#if 0

-- deleted a bunch of history -- 12-May-99 tc@DinkumSoftware.com

-- deleted a bunch of history -- 27-Apr-99 thru 17-jul-99

-- deleted a bunch of history -- 26-jul-99 thru 27-mar-00

-- deleted a bunch of history -- 05-Apr-00 thru 07-may-00

-- deleted a bunch of history -- 08-may-00 thru 31-jul-00

-- deleted a bunch of history -- 01-Aug-00 thru 04-Oct-00

-- deleted a bunch of history -- 19-Oct-00 thru 29-Oct-01

-- deleted a bunch of history -- 09-Nov-01 thru 02-Jan-02

-- deleted a bunch of history -- 02-Jan-02 thru 29-jul-02

-- deleted a bunch of history -- 02-Aug-02 thru 20-Dec-02

-- deleted a bunch of history -- 18-jan-03 thru 21-Dec-03

-- deleted a bunch of history -- 31-jan-04 thru 08-Dec-04

-- deleted a bunch of history -- 05-Jan-05 thru 22-Dec-07

02-Jan-08 pfurey@webbresearch.com   SN#3369 Added when_utc to surface behavior.

12-Jan-08 pfurey@webbresearch.com   SN#3370 Added u_hover_bpump_delta_value(cc)

Removed bpump_delta_value from

drift_at_depth argument list

29-Jan-08 pfurey@webbresearch.com

for smith@spawar.navi.mil   SN#3371 Added sensors for sscsd proglet.

07-Feb-08 pfurey@webbresearch.com   SN#3372 Added x_cycle_time, u_low_power_cycle_time

and u_allowable_devsched_msecs

22-feb-08 tc@DinkumSoftware.com SN#3372 Added cc_final_est_time_til_inflection(s)

m_time_til_wpt(s) cc_est_time_til_inflection(s)

23-feb-08 tc@DinkumSoftware.com SN#3373 Added bviper b_arg: min_reqd_quiet_time(s)

24-feb-08 tc@DinkumSoftware.com SN#3374 Added bviper b_arg: post_inflection_holdoff(s)

26-feb-08 tc@DinkumSoftware.com SN#3375 Added bviper b_arg: allow_sample_at_surface(bool)

27-feb-08 tc@DinkumSoftware.com SN#3376 bug fix, bviper b_args:when_* had no units

14-Mar-08 pfurey@webbresearch.com SN#3377 Changed default value of

u_max_altimeter(m) from 30 to 100

15-Mar-08 pfurey@webbresearch.com SN#3378 Added u_low_power_hd_fin_ap_gain (_igain), and

x_hd_fin_ap_gain (_igain)

17-Mar-08 pfurey@webbresearch.com SN#3379 Added sensor: sci_motebb_logout(nodim) to

for ahails@mote.org motebb list of parameters.

20-Mar-08 pfurey@webbresearch.com SN#3380 Changed the default value of

u_de_avg_oil_vol_err_alpha from

0.05 to 0.0 as a temp bug work-around.

12-apr-08 dpingal@webbresearch.com SN#3381 added proglet bb2flV3

03-jun-08 dpingal@webbresearch.com SN#3382 fixed cal factors, units for bb2flV3

07-Jun-08 peter@fureysoftware.com SN#3383 Added sensors for FIRe proglet

23-jun-08 dpingal@webbresearch.com SN#3384 added ohf

14-jul-08 dpingal@webbresearch.com SN#3385 added proglet bb2flV4

14-jul-08 dpingal@webbresearch.com SN#3386 added proglet bb2flV5

15-sep-09 fmarcelino@webbresearch.com SN#3387 added c_iridium_phone_num_alt,

c_iridium_failover_retries, m_iridium_attempt_num

and m_iridium_current_num (MANTIS 255)

14-Oct-08 fnj@webbresearch.com SN#3387A Added u_sci_cycle_time

2008.10.23 tc@DinkumSoftware.com SN#3388 UTM nav bug

Renamed some sensors for consistency

X_LMC_UTM_VEHICLE_ZONE_DIGIT => X_LMC_UTM_VEH_ZONE_DIGIT

X_LMC_UTM_VEHICLE_ZONE_CHAR => X_LMC_UTM_VEH_ZONE_CHAR

X_LAST_UTM_EASTING_CORRECTION => X_LMC_UTM_VEH_EASTING_CORRECTION

X_LAST_UTM_NORTHING_CORRECTION => X_LMC_UTM_VEH_NORTHING_CORRECTION

22-dec-08 dpingal@webbresearch.com SN#3389 fixed annotation, merged all reread log file sensors into one

22-dec-08 dpingal@webbresearch.com SN#3390 added m_argos_timestamp, changed u_reqd_depth_at_surface to 2

15-jan-09 dpingal@webbresearch.com SN#3391 added c_recovery_on

2009.01.23 tc@DinkumSoftware.com SN#3392 deleted a bunch of history. 05-Jan-05 thru 22-Dec-07

2009.01.26 tc@DinkumSoftware.com SN#3393 Added c_iridium_reread_config_files(button)

c_iridium_lead_zeros_alt(nodim)

2009.02.06 tc@DinkumSoftware.com SN#3384 Typo in comment

16-Feb-09 fnj@webbresearch.com SN#3389B Added u_sci_dbd_sensor_list_xmit_control

22-jan-08 dpingal@webbresearch.com SN#3394 added x_low_power_status,

c_coulomb_on, u_coulomb_debug, m_coulomb_amphr,

m_coulomb_current, u_coulomb_timeout, u_iridium_force_port

06-mar-09 dpingal@webbresearch.com SN#3395 added min_depth, max_depth to sample behavior

17-mar-09 fmarcelino@webbresearch.com SN#3396 Modified c_iridium_current_num to initialize it to 0 which is now

the primary number.

2009.01.22 tc@DinkumSoftware.com SN#3397 Added gbus and coulomb device driver sensors

C_COULOMB_ON, U_COULOMB_DEBUG,

M_COULOMB_AMPHR, M_COULOMB_CURRENT,

M_COULOMB_AMPHR_RAW, M_COULOMB_CURRENT_RAW

Added gbus digifin_v2 sensors

U_DIGIFIN_V2_DEBUG,

Moved some fin sensors in file to digifin_v2:

f_fin_safety_max(rad), c_fin(rad), m_fin(rad)

#

2009.03.18 fmarcelino@webbresearch.com SN#3398 Changed c_iridium_failover_retries to

u_iridium_failover_retries and upped the default

value to 5

2009.03.31 fmarcelino@webbresearch.com SN#3399 Added m_iridium_rssi sensor to store the iridium

signal strength.

2009.04.01 fmarcelino@webbresearch.com SN#3400 Changed m_iridium_rssi to m_iridium_signal_strength

and changed the default value to -1.0

2-Apr-09 dpingal@webbresearch.com SN#3401 removed a large number of unused sensors

(now commented out)

6-Apr-09 dpingal@webbresearch.com SN#3402 Added ballast control b_args to drift_at_depth so they

can be tuned

9-Apr-09 dpingal@webbresearch.com SN#3403 Added bbam proglet, fixed sensors_in b_args, made sure

that all proglets are included

14-Apr-09 fnj@webbresearch.com SN#3404 Fixed minor conflict in merge from HEAD to SCIENCE_DATA_LOGGING_BRANCH.

2009.04.22 fmarcelino@webbresearch.com SN#3404 Added lithium Ion Power device sensors

2009.05.18 fmarcelino@webbresearch.com SN#3405 Added u_motor_debug. Added leak detect sensors to

support new hardware which can now be independently

checked with the new hardware.

Also added new sensors for new vehicle temp driver

New sensors are veh_temp_*

2009.05.18 fmarcelino@webbresearch.com SN#3405 Added u_motor_debug. Added leak detect sensors to

support new hardware which can now be independently

checked with the new hardware.

Also added new sensors for new vehicle temp driver

New sensors are veh_temp_*

01-Jun-09 fnj@webbresearch.com SN#3405 Added x_science_logging_state.

07-Jul-09 fnj@webbresearch.com SN#3406 Added m_mission_start_time.

08-Jul-09 fnj@webbresearch.com SN#3407 Added m_science_readiness_for_consci.

13-Jul-09 fnj@webbresearch.com SN#3408 Changed c_science_printout from 2 to 0.

2009.07.22 pfurey@webbresearch.com SN#3406 Added sensors for uModem proglet.

2009.07.30 fmarcelino@webbresearch.com SN #3407 Added u_alt_filter_enabled to enable/disable median

filtering.

2009.08.12 fmarcelino@webbresearch.com SN #3408 Added f_digifin_movement_retry_max to handle digifin retry

attempts before issuing warnings

2009-08-18 fnj@webbresearch.com SN#3407 Added SCI_X_SENT_DATA_FILES.

2009.08.20 pfurey@webbresearch.com SN#3409 Added extra output sensors to FIRe proglet

2009-08-24 fnj@webbresearch.com SN#3408 Added sci_m_disk_usage, sci_m_disk_free, sci_x_disk_files_removed.

2009-08-31 fnj@webbresearch.com SN#3409 Added c_science_send_all. Fixed doco for c_science_on.

2009.09.09 pfurey@webbresearch.com SN#3410 Replaced enum with nodim units for sci_badd_error

and sci_FIRe_error, was causing logging error.

2009.09.10 pfurey@webbresearch.com SN#3411 Added sensors for rinkoII proglet.

2009.09.18 pfurey@webbresearch.com SN#3412 Added sensors for dvl proglet and

b_arg: intersample_depth for sample behavior.

2009-09-22 fnj@webbresearch.com Reconciled merge conflicts.

2009.09.24 fmarcelino@webbresearch.com SN#3413 Added sensors f_coulomb_calibration_factor and f_clock_source

2009.09.24 pfurey@webbresearch.com SN#3414 Added m_avg_depth_rate(m/s) and

u_avg_depth_rate_alpha(nodim)

2009-09-30 fnj@webbresearch.com Reconciled merge conflicts.

2009-10-29 fmarcelino@teledyne.com SN#3415 added b_arg: remaining_charge_min(%) and

b_arg: remaining_charge_sample_time(sec)

2009.10.19 dpingal@webbresearch.com SN#3415 Added m_science_readiness_for_consci.

2009.10.21 dpingal@webbresearch.com SN#3416 Added u_ballast_pumped_stop_distance(cc),

u_battpos_stop_distance(in).

added bb2flsV6 proglet

Changed c_iridium_phone_num_alt value to production dockserver

2009.10.22 dpingal@webbresearch.com SN#3417 Changed bb2flsV6 output chl->cdom

2009.11.05 pfurey@webbresearch.com SN#3418 Added sensors for fpitch_pump driver, dynamic control,

and new behavior argument values for pitch control.

2009-11-17 fnj@omegatech.hatchescreek.com SN#3419 Added sci_m_spare_heap, sci_m_min_free_heap, sci_m_min_spare_heap.

2009-12-01 pfurey@webbresearch.com SN#3420 Added sensor: m_science_sync_time(timestamp)

2009-12-02 pfurey@webbresearch.com SN#3421 Added sensor: u_att_rev_ignore_warnings(bool)

2009-12-03 fmarcelino@teledyne.com SN#3422 Added m_coulomb_amphr_total(amp-hrs) persistant amp-hours total

2009-12-22 fnj@omegatech.hatchescreek.com SN#3423 Added u_science_send_time_limit_adjustment_factor.

2009-12-29 dpingal@teledyne.com Fixed comment for c_recovery_on

2009-12-30 pfurey@webbresearch.com SN#3424 Removed u_abort_c_battpos(in)

2010.01.12 pfurey@webbresearch.com SN#3425 Added nth_yo_to_sample sample argument.

2010-01-15 fnj@webbresearch.com SN#3426 Changed default value of u_sci_cmd_max_consci_time from 1200 to 3600.

2010-02-09 pfurey@webbresearch.com SN#3427 Added b_arg: end_action(enum) to goto_list behavior.

2010-02-01 pfurey@webbresearch.com SN#3428 Changed values for f_pitch_fluid_pumped_cal_m

and f_pitch_fluid_pumped_cal_b.

Added f_thermal_reqd_acc_pres(bar) and updated values

for some engpres.c sensors, thr_reqd_pres_mul(nodim),

and max_pumping_charge_time(sec). Changed definition

and value of eng_pressure_mul(nodim).

2010-02-12 pfurey@webbresearch.com SN#3429 Removed MTHR_AWAITING_AIR from the legal values

of m_thermal_pump(enum)

2010-02-22 tc@DinkumSoftware.com SN#3430 Added C/M_AVBOT_POWER/ENABLE to masterdata.

2010-02-25 pfurey@webbresearch.com SN#3431 Removed u_allowable_devsched_msecs.

2010-03-12 fmarcelino@teledyne.com SN#3432 Added sensors m_gps_uncertainty(nodim) and m_gps_num_satellites(nodim)

2010-03-12 fmarcelino@teledyne.com SN#3433 added b_arg: strobe_on and m_strobe_ctrl

2010-03-15 pfurey@webbresearch.com SN#3434 Added/removed args to/from drift_at_depth behavior.

2010-03-22 pfurey@webbresearch.com SN#3435 Added u_sound_speed(m/s) and u_angle_of_attack(rad).

2010-03-19 pfurey@webbresearch.com SN#3436 Added xs_fluid_pumped(cc).

2010-04-08 fmarcelino@teledyne.com SN#3437 Changed initialization of m_iridium_attempt_num to 1

2010-04-08 pfurey@webbresearch.com SN#3438 Added simulation sensors for do_thermal_oil().

2010-04-23 pfurey@webbresearch.com SN#3439 Added sensors for flbbrh pnd flur proglets.

2010-05-24 pfurey@webbresearch.com SN#3440 Added sensors for bb2flsV7 and flbbcd proglets.

2010-06-17 pfurey@webbresearch.com SN#3441 Added sensors for dmon proglet.

2010-07-01 pfurey@teledyne.com SN#3442 Added sensors for c3sfl proglet.

2010-08-12 pfurey@teledyne.com SN#3443 Updated/added/removed some drift_at_depth

arguments and sensors.

2010-08-10 fmarcelino@teledyne.com SN#3444 Added sensors for suna proglet.

2010-08-23 pfurey@teledyne.com SN#3445 Added target_altitude, alt_time, and bpump_delay

to drift_at_depth. Added x_target_hover_depth

and x_avg_hover_depth.

2010-08-24 fmarcelino@teledyne.com SN#3446 Added f_coulomb_battery_capacity for mantis #722

2010-09-07 pfurey@teledyne.com SN#3447 Updated drift_at_depth argument defaults.

2010-09-14 pfurey@teledyne.com SN#3448 Changed the default for the abend b_arg:

max_allowable_busy_cpu_cycles(cycles)

from -1 to 75 (see Mantis issue #767).

2010-09-21 www.DinkumSoftware.com/tc SN#3449 X_AVBOT_DISABLED

2010-09-22 www.DinkumSoftware.com/tc SN#3450 SCI_AVBOT_PROGLET_IS_INSTALLED

2010-09-30 fmarcelino@teledyne.com SN#3451 Added C_LOGGER_CTRL_TIMEOUT (Mantis #793)

2010-10-01 pfurey@teledyne.com SN#3452 Added u_dvl_pd_data_stream_select(enum) and

sci_dvl_ensemble_offset(nodim).

2010-10-06 pfurey@teledyne.com SN#3453 Added u_dvl_wd_data_out(nodim).

2010-10-14 fmarcelino@teledyne.com SN#3454 Added f_device_reinit_timeout(min).

2010-11-09 pfurey@teledyne.com SN#3455 Updated f_coulomb_calibration_factor(%)

from 0.68 to 0.068

2010-11-16 fmarcelino@teledynec.om SN#3456 Changed units of the sci_suna_* sensors

2010-12-02 pfurey@teledyne.com SN#3457 Added m_est_time_to_surface(sec),

when_utc_on_surface(bool),

m_avg_upward_inflection_time(sec),

m_avg_downward_inflection_time(sec),

m_avg_climb_rate(m/s), and

m_avg_dive_rate(m/s).

Changed u_avg_depth_rate_alpha(nodim) from 0.96 to 0.5

2010-12-15 pfurey@teledyne.com SN#3458 Added comatose b_arg: start_sci_wants_quiet(bool)

and sci_wants_quiet(bool).

2010-12-28 fmarcelino@teledyne.com SN#3459 Added u_suna_bootup_time

2010-01-31 dpingal@teledyne.com SN#3460 Bug#874, changed value for
u_max_water_depth_lifetime

from 3 to 1

2010-02-01 dpingal@teledyne.com SN#3461 Bug#866, changed value for
f_coulomb_calibration_factor

from .068 to .05

2011-02-02 pfurey@teledyne.com SN#3462 Added sci_bbam_sim_is_installed().

2011-02-07 pfurey@teledyne.com SN#3463 Added sci_auvb_sim_is_installed().

2011-02-09 pfurey@teledyne.com SN#3464 Added sensors for satpar proglet.

2011-02-24 lcooney@teledyne.com SN#3465 Added sensors for autoballast control and

b_args for yo, dive_to, climb_to. See /doco/how-it-works/autoballast.txt

2011-03-02 lcooney@teledyne.com SN#3466 Changed value of c_delta_bpump_ballast(X) from 25 to -1

and added sensor m_delta_bpump_ballast_adj

2011-03-14 pfurey@teledyne.com SN#3467 Increased b_arg: vacuum_max(inHg) from 11.0 to 12.0

2011-04-07 pfurey@teledyne.com SN#3468 Added sensors for vsf proglet.

2011-04-12 lcooney@teledyne.com SN#3469 Added u_calc_angle_of_attack

2011-04-13 pfurey@teledyne.com SN#3470 Added sensors for oxy4330f proglet.

2011-05-03 www.DinkumSoftware.com/tc SN#3471 added u_avbot_debug for avbot 485 usage

2011-05-05 lcooney@teledyne.com SN#3472 Added c_autoballast_state which replaces c_autoballast_converged and

c_autoballast_init. Added u_autoballast_abort to choose whether or not to

abort on convergence error. Removed m_delta_bpump_ballast_adj

2011-05-04 fmarcelino@teledyne.com SN#3473 Added s_coulomb_relative_charge

2011-04-08 dpingal@teledyne.com SN#3474 Changed defauly values for lots of b_args to make glider

fly OK with defaults

2011-06-02 pfurey@teledyne.com SN#3475 Added sensors for gamma_rad5 proglet.

2011-06-14 pfurey@teledyne.com SN#3476 Added sensors for bsipar proglet.

2011-06-29 pfurey@teledyne.com SN#3477 Added x_hover_active(bool) and x_deactivate_hover(bool)

2011-07-15 fmarcelino@teledyne.com SN#3478 Changed f_clock_source default to 1

2011-07-19 pfurey@teledyne.com SN#3479 Added m_bpump_fault_bit(bool) and

m_de_pump_fault_count(nodim) (Mantis #945).

Changed f_de_oil_vol_safety_max(cc) from

300 to 290 (Mantis #974)

2011-09-08 pfurey@teledyne.com SN#3480 Changed units for bsipar scale factor.

2011-09-12 dpingal@teledyne.com SN#3481 Added energy monitoring for pitch and fin

2011-09-12 lcooney@teledyne.com SN#3482 Added the following sensors for heading cntrl: m_hdg_derror, u_hd_fin_ap_dgain,

u_low_power_hd_fin_ap_dgain, x_hd_fin_ap_dgain

x_heading_deadband, x_heading_rate_deadband, u_fly_deep_in_shallow,x_heading_reversal.

Added the following sensors for pitch servo memory:

x_battpos_achieved, u_use_pitch_servo_memory, c_dive_battpos, c_climb_battpos

u_battpos_avg_num_min, u_battpos_ap_deadband

Added when_secs to set_heading behavior

Updated comments for x_hd_fin_ap_ran

2011-09-14 lcooney@teledyne.com SN#3483 Changed u_pitch_ap_gain from -2.86 to -5.0

Changed u_pitch_max_delta_battpos from 0.020 to 0.2

Removed U_BATTPOS_AP_DEADBAND

2011-09-14 lcooney@teledyne.com SN#3484 For yo/climb-to behavior, removed use_sc_model b_arg.

For yo/dive-to behavior, added wait_for_ballast b_arg.

Added sensor c_wait_for_ballast

2011-09-19 lcooney@teledyne.com SN#3485 Changed f_min_ballast from 150.0 to 250.0

2011-09-28 fmarcelino@teledyne.com SN#3486 Added x_last_commanded_fin_pos(rad)

2011-10-13 lcooney@teledyne.com SN#3487 Changed u_pitch_ap_gain from -5.0 to -3.0,

[u/x]_hd_fin_ap_gain from 1.0 to 1.5,

[u/x]_hd_fin_ap_igain from 0.03 to 0.02,

[u/x]_hd_fin_ap_dgain from -1.0 to -4.0

2011-10-14 lcooney@teledyne.com SN#3488 Changed x_heading_rate_deadband to u_heading_rate_deadband,

x_heading_deadband to u_heading_deadband

2011-11-18 pfurey@teledyne.com SN#3489 Added x_clothesline_state(enum) and changed c_de_oil_vol(cc)

from 270 to 260.

2011-12-12 pfurey@teledyne.com SN#3490 Added sensor: x_surface_active(nodim) (Mantis #1113).

2011-12-21 pfurey@teledyne.com SN#3491 Added sensors for flbb proglet.

2011-12-22 dpingal@teledyne.com SN#3492 Added comments for Control Center b_arg metadata

2012-01-03 dpingal@teledyne.com SN#3493 Added comments for Control Center sensor metadata

2012-01-03 pfurey@teledyne.com SN#3494 Added u_dvl_ensemble_timeout(sec)

2012-12-07 / thruster_devel: 2012-01-04 dpingal@teledyne.com SN#3495 Minor syntax fixes to Control Center metadata,

added c_thruster_on

2012-01-06 fmarcelino@teledyne.com SN#3504 Added sensors u_vr2c_serial_num_0, u_vr2c_serial_num_1 and

sci_vr2c_state

2012-12-07 / thruster_devel: 2012-01-13 lcooney@teledyne.com SN#3496 Thruster support: Added [dive/climb]_use_thruster/thruster_value to yo,

drift_at_depth, dive and climb behaviors. Added cc_[final]_thruster_[mode/value],

#
dc_c_thruster_on

2012-12-07 / thruster_devel: 2012-01-18 lcooney@teledyne.com SN#3497 Added drift_at_depth b_arg: enable_steering(bool) and

sensor x_enable_steering_during_hover (set by enable_steering)

2012-01-25 fmarcelino@teledyne.com SN#3495 Added CTD41CP2 sensors.

2012-02-08 dpingal@teledyne.com SN#3496 Fixed default value of yo:d_pitch_value

2012-12-07 / thruster_devel: 2012-02-09 fmarcelino@teledyne.com SN#3505 Added u_vr2c_status_interval_0 and _1 and u_vr2c_profile_0 and _1

2012-12-07 / thruster_devel: 2012-02-16 lcooney@teledyne.com SN#3498 Added u_thruster_inflection_holdoff, x_thruster_state, and c_hover_battpos

2012-03-15 fmarcelino@teledyne.com SN#3497 Added sci_badd_mmp_is_installed

2012-12-07 / thruster_devel: 2012-Mar-19 lcooney@teledyne.com SN#3499 Added sensors for get_est_horz_speed_thruster():

m_thruster_est_speed, f_thruster_v[0,1], f_thruster_i[0-2]

x_thruster_has_current_sense and m_thruster_current

f_thruster_min_v and f_thruster_max_v

2012-12-07 / thruster_devel: 2012-Mar-26 lcooney@teledyne.com SN#3500 Added drift_at_depth b_arg wait_for_pitch,

X_PITCH_AP_GAIN, U_PITCH_AP_GAIN_THRUSTER, X_PITCH_AP_DEADBAND, U_PITCH_AP_DEADBAND_THRUSTER

U_THRUSTER_HD_FIN_AP_GAIN(IGAIN,DGAIN)

#

2012-12-07 / thruster_devel: 2012-03-30 dpingal@teledyne.com SN#3501 Added m_thruster_current

2012-12-07 / thruster_devel: 2012-03-30 dpingal@teledyne.com SN#3502 Added m_thruster_report

2012-04-12 fmarcelino@teledyne.com SN#3498 Added behavior arguments to badd_b behavior:

c_badd_autobaud(bool), c_baud_attemp_min(enum),

c_baud_attempt_max(enum) and c_autobaud_max_BER(nodim)

2012-04-18 fmarcelino@teledyne.com SN#3499 Added sensors to supporting behavior aguments for badd

2012-04-23 pfurey@teledyne.com SN#3500 Replaced oxy4330f with oxy4 proglet

2012-04-27 fmarcelino@teledyne.com SN#3501 Added c_badd_transaction_num and associated b_arg

2012-05-03 pfurey@teledyne.com SN#3502 Changed the default values of

u_dvl_pd_data_stream_select from 6 to 0

u_dvl_num_errors_before_restart from 5 to 1

u_dvl_ensemble_timeout from 60 to 20

2012-05-07 fmarcelino@teledyne.com SN#3502 Added b_arg: when_wpt_dist to badd_b behavior.

2012-05-09 fmarcelino@teledyne.com SN#3503 Added b_arg: c_badd_channel_probe_test and sensor.

2012-01-06 fmarcelino@teledyne.com SN#3504 Added sensors u_vr2c_serial_num_0, u_vr2c_serial_num_1 and

sci_vr2c_state

2012-02-09 fmarcelino@teledyne.com SN#3505 Added u_vr2c_status_interval_0 and _1 and u_vr2c_profile_0 and _1

2012-06-11 pfurey@teledyne.com SN#3506 Added sensor: u_dvl_bl_min_layer_size(dm)

sensor: u_dvl_bl_near_layer_boundary(dm)

sensor: u_dvl_bl_far_layer_boundary(dm)

sensor: u_dvl_bk_water_mass_layer_mode(enum)

Removed sensor: u_dvl_bottom_track_mode(enum)

2012-06-27 pfurey@teledyne.com SN#3507 Changed c_ctd41cp_num_fields_to_send from 3 to 4 (Mantis#1339)

2012-12-07 / thruster_devel: 2012-07-05 lcooney@teledyne.com SN#3505 Added drift_at_depth b_args start_dist_from_target, use_bpump_servo

Added drift_at_depth servo sensors u_depth_ap_[d]gain_[deep/shallow]

Added yo/climb b_arg stop_when_air_pump

Added airpump sensors m_vacuum_change_since_air_pump_on, m_vacuum_air_bag_inflated, m_vacuum_air_pump_on,

u_vacuum_air_bag_inflated, u_max_depth_for_air_pump_est

Added thruster sensors u_ap_thruster_delta_cmd and u_ap_thruster_depth_rate_deadband

u_ap_thruster_depth_rate_period, x_ap_thruster_depth_rate_period

Added m_surface_depth_reached and u_pitch_surface

Added thruster-assisted abort parameters: abend b_arg use_thruster_for_ascent, u_thruster_abort_inflection_holdoff,

x_use_thruster_for_abort_ascent

2012-07-09 pfurey@teledyne.com(for ahails@mote.org) SN#3508 Replaced references to obsolete

proglet MoteBB with new proglet MoteOPD. Added

c_moteopd_debug(bool) and c_moteopd_data_overtime(sec).

Chg default sensor: c_moteopd_on(sec) to -1

2012-08-24 pfurey@teledyne.com SN#3509 Removed hs2 proglet to make supersci.app smaller.

Added c_obsolete_on as a place holder for removed proglets

in science_super.c and sample.c

2012-09-04 pfurey@teledyne.com SN#3510 Removed whpar proglet

2012-09-05 pfurey@teledyne.com SN#3511 Removed ohf proglet

2012-09-06 pfurey@teledyne.com SN#3512 Removed avbot proglet

2012-09-07 pfurey@teledyne.com SN#3513 Removed whgpbm proglet

2012-09-18 fmarcelino@teledyne.com SN#3514 Added f_suna_firmware_cfg sensor.

2012-10-17 fmarcelino@teledyne.com SN#3515 Changed c_badd_target_id to -1

2012-11-07 fmarcelino@teledyne.com SN#3515 Mantis #1392

2012-11-19 lacooney@alum.mit.edu SN#3516 Added PID steering sensors: u_hd_fin_ap_scale_by_max, u_hd_fin_ap_deadband_reset

2012-12-03 lacooney@alum.mit.edu SN#3517 Added u_autoballast_end_on_converge

2012-12-03 dpingal@teledyne.com SN#3518 Mantis 1453 - Removed dvl setup sensors

2012-12-05 dpingal@teledyne.com SN#3519 Added u_dvl_single_pd0_file

2012-12-07 / thruster_devel: 2012-09-14 lacooney@alum.mit.edu SN #3506 Renamed drift_at_depth b_arg bpump_servo to depth_ctrl (now offer several options for depth control

Added pitching depth servo control from bclaus@mun.ca: and sensors:

u_hover_depth_pitch_limit, u_hover_depth_p_gain, u_hover_depth_d_gain, u_hover_depth_pitch_deadband,

u_hover_depth_pitch_offset, x_avg_depth_pitch_battpos_offset, u_avg_depth_pitch_battpos_alpha,

u_avg_depth_pitch_battpos_deadband, u_hover_depth_pitch_max_time

Renamed u_depth_ap_[d]gain_[deep/shallow] to u_hover_bpump_ap_[d]gain_[deep/shallow]

Renamed [u/x]_ap_thruster_depth_rate_period to [u/x]_thruster_ap_period

Added pitch servo d gain: x_pitch_ap_dgain, u_pitch_ap_dgain, u_pitch_ap_dgain_thruster and m_pitch_derror

thruster gain scale: u_pitch_ap_scale_thruster_gain and u_max_thruster_speed

Added u_autoballast_end_on_converge

Added u_depth_rate_thr_avg_num and m_depth_rate_thr_avg_final, m_depth_error

Updated x_thruster_state to include more information

Added u_hd_fin_ap_deadband_reset

u_hd_broll_ap_deadband_reset, u_hd_broll_ap_scale_by_max, u_hd_broll_ap_dgain

2012-12-07 / thruster_devel: 2012-10-25 lacooney@alum.mit.edu SN#3507 Added x_hover_depth_p_gain, u_hover_depth_gain_scale_[m,b], dad b_arg depth_gain_scale

Removed u_pitch_ap_scale_thruster_gain

Changed many drift_at_depth sensors to b_args: depth_pitch_limit, depth_p_gain, depth_d_gain,

depth_pitch_deadband, depth_pitch_offset, depth_pitch_max_time

2012-12-07 / thruster_devel: 2012-11-02 dpingal@teledyne.com SN#3515 Merged thruster_devel into main branch

2012-12-07 / thruster_devel: 2012-11-05 dpingal@teledyne.com SN#3516 Couple new experimental DVL settings

2012-12-07 / thruster_devel: 2012-11-08 lacooney@alum.mit.edu SN#3508 Added drift_at_depth sensors:

m_depth_ierror, m_depth_derror, x_hover_depth_i_gain, x_hover_depth_d_gain, u_hover_depth_p_gain_min

x_hover_depth_pitch_limit, x_hover_depth_ap_ran, x_hover_depth_pitch_is_maxed

u_hover_depth_[run_time, deadband, rate_deadband, inflection_holdoff, hardover_holdoff,

scale_by_max, deadband_reset, limit_gain_x_error, absolute, abort_after_y_misses]

Changed x_hover_depth_p_gain and from 0.15 to -0.15 (sign of m_depth_error changed)

Added drift_at_depth b_arg depth_i_gain and removed b_arg depth_pitch_offset

Added u_hd_fin_ap_scale_by_max, u_hd_broll_ap_deadband_reset, u_hd_broll_ap_scale_by_max, u_hd_broll_ap_dgain

Added thruster sensors: u_avg_thruster_current_num, m_avg_thruster_current, u_max_thruster_current, m_thruster_current_spike

Added u_avg_thruster_speed_num and m_avg_thruster_speed

2012-12-07 / thruster_devel: 2012-11-26 lacooney@alum.mit.edu SN#3517 Added m_avg_thruster_depth

2012-12-07 dpingal@teledyne.com SN#3520 Merged thruster branch into tip

2012-12-07 lacooney@alum.mit.edu SN#3521 Added u_secs_surface_depth_reached,

m_thruster_power, m_thruster_voltage, m_thruster_amphr, m_thruster_watthr

Added additional use_thruster/thruster_value parameter (2: percent max)

2012-12-11 fmarcelino@teledyne.com SN#3522 Changed u_dvl_single_pd0_file default to 0

2012-12-14 lacooney@alum.mit.edu SN#3523 Changed values of following (M#0001460): u_pitch_ap_gain_thruster -3 to -2, c_thruster_current_cal 0.037 to 0.0593

f_thruster_i0 0.1506 to 0.04352, f_thruster_i1 2.8022 to 1.289, f_thruster_i2 -2.4233 to -0.5438

#
u_thruster_hd_fin_ap_gain 1.0 to 0.5, u_thruster_hd_fin_ap_dgain 0 to -4.0

2012-12-14 lacooney@alum.mit.edu SN#3523 Update to comment for c_stop_when_air_pump, use_thruster_for_ascent

2012-12-28 fmarcelino@teledyne.com SN#3524 Removed all VR2C input sensors as they're no longer used (Mantis #1448)

2013-02-07 fmarcelino@teledyne.com SN#3525 Added echosndr853 sensors.

2013-02-22 dpingal@teledyne.com SN#3526 Added m_thruster_raw

2013-03-01 lacooney@alum.mit.edu SN#3527 Added use_thruster = 4 mode for dive, climb, yo, and drift_at_depth behaviors,

and sensors u_ap_thruster_power_deadband, u_ap_thruster_power_p_gain, m_thruster_power_error

Added m_avg_thruster_power_drift

Added [x,u]_increase_vacuum_time

Modified the following: M_AVG_THRUSTER_CURRENT ==> M_AVG_THRUSTER_POWER, U_AVG_THRUSTER_CURRENT_NUM ==> U_AVG_THRUSTER_POWER_NUM

M_THRUSTER_CURRENT_SPIKE ==> M_THRUSTER_POWER_SPIKE. U_MAX_THRUSTER_CURRENT ==> F_THRUSTER_POWER_MAX

Modified comments for x_thruster_state to include use_thruster=4

c_autoballast_state now includes more information on failed convergence

Changed u_low_power_hd_fin_ap_[i]gain: _gain from 1.0 to 0.5, igain from 0.004 to 0.0001 M#1507

2013-04-05 dpingal@teledyne.com SN#3528 Passive retraction changes: added

x_ballast_passive_retraction_count, f_ballast_passive_retraction_delay,

changed default f_ballast_pumped_battery_spike_trigger to 3.0

#

2013-04-23 lacooney@alum.mit.edu SN#3529 Added surface behavior b_arg: thruster_burst,

Added sensors: u_thruster_burst_volts, u_thruster_burst_secs,

u_thruster_delta_max, c_thruster_surface_[secs,depth], c_thruster_depth_rate_[secs,depth], x_why_lens_completed

u_thruster_power_limit_cmd, u_thruster_power_delta_max

Changed value of u_increase_vacuum_time from 30 to 0, c_thruster_current_cal from 0.0593 to 0.038

2013-04-30 lacooney@alum.mit.edu SN#3530 Updated values of drift_at_depth b_args depth_gain_scale (0 -> 1),

depth_i_gain (0 -> -0.0001), depth_d_gain (0 -> 0.1)

f_thruster_max_v (9 -> 9.7), u_thruster_power_limit_cmd (70 -> 80), u_thruster_power_delta_max (7 -> 3)

u_vacuum_air_bag_inflating (0.5 -> 0.3)

f_thruster_i0 (0.04352->0.1473), f_thruster_i1(1.289-> 0.9018), f_thruster_i2(-0.5438->-0.2083)

u_hover_depth_gain_scale_m (0.270->0.429), u_hover_depth_gain_scale_b (-0.189->-0.430)

f_thruster_power_max (10.0->14.5), f_thruster_power_min(1.0->0.5)

2013-05-07 dpingal@teledyne.com SN#3531 Removed c_thruster_report

#2014-03 mz6580@mun.ca SN#3530 Deleted oxy3835 and glbps but added TRITECH SONAR

When you edit this file, increment MASTERDATA_SN by one.

This serial number is used to detect whether edit_struct.exe was run

before the software was compiled.

#endif

#define MASTERDATA_SN 3530

#if 0

--

prefix meanings:

m_ measured

c_ commanded

u_ user defined before run time

f_ Set in factory, do not change unless you know what you are doing

x_ Do not ever set this. Typically computed at run-time.

s_ simulated state variables

Testing only!!!!!

sensor: x_thrvalve_num_of_identical_readings_in_a_row(nodim)   0

--

Sensor values being passed to science over the clothesline have a

decimal precision limit of 6 places. As a workaround for sensor values

with very small values (<< 0, high decimal precision), say

u_bb2c_beta532_factor (0.000007494) we developed the following concepts:

"Mnodim" which signifies that the true value of the sensor has been

multiplied by 1.e6 and therefor must be divided by 1.e6 on the science side.

"Tnodim" which signifies that the true value of the sensor has been

multiplied by 1.e13 and therefor must be divided by 1.e13 on the science side.

```
# ------------------------------------------------------------------------

# Some general glider specific characteristics

sensor: f_max_working_depth(m)   30.0  #! visible = True; min = 2.0; max = 1000.0

                                # How deep glider can work

                                # NOTE: set this to 194m if you want a regular

                                # electric glider to bottom out at 200m


sensor: f_nominal_dive_rate(m/s) 0.19       # clips 0-1

sensor: f_nominal_pitch(rad)     0.4363     # 25 degs, clips 0-90 degs

sensor: f_device_reinit_timeout(min) 2.0 # The amount of time to elapse before the glider

                                 # attempts to bring non super-critical devices

                                 # into back into service when a mission aborts

                                 # (in minutes)



# SENSORS


# --- Configuration, Read Only at reset time

sensor: f_enable_picozoom(bool) 1.0  #  0=> never enable picozomm

                                #  1=> enable it if M_FREE_HEAP is > F_AUTO_PICOZOOM_HEAP_REQD

                                #  2=> always enabled Picozoom

sensor: f_auto_picozoom_heap_reqd(bytes) 100000 # heap required to autoenable picozoom
```

```
# --- Set at init time
sensor: x_hardware_ver(nodim) -3.0 # hardware rev
                        #  128  RevE
                        #   -2  initial value, i.e. before set
                        #   -1  error reading jumpers
                        #    0  early board without jumpers --or--
                        #       Board has jumpers, none set

# --- Set/used in gliderdos
sensor: x_software_ver(nodim) 0.0 # current software version
sensor: x_in_gliderdos(bool)  0.0 # true->in glider as opposed to a mission
sensor: x_are_in_lab(bool)    0.0 # true->started with -lab command line switch
sensor: x_are_running_onetime_sequence(bool) 0.0 # true -> onetime.seq active

sensor: u_max_time_in_gliderdos(sec) 600.0          #! visible = True
                                    # in, run "sequence" after this much time
                                    # in gliderdos without receiving a keystroke
                                    # disabled in -lab mode
                                    # disabled if <= 0

                # these are used
sensor: u_max_sequence_repetitions(nodim)       100   # in, upper limit on # repetitions allowed
                                     #     in a sequence specifier listed in a
                                     #     sequence command (e.g., sequence foo.mi(100)
```

sensor: u_max_total_sequenced_missions(nodim) 100 # in, upper limit on total missions sequenced

sensor: u_max_allowed_lastgasp_aborts(nodim) 1 # in, how many lastgasp.mi aborts to allow

before returning to GliderDos

sensor: u_sequence_max_time_in_gliderdos(s) 900 # in, how long to stay in Gliderdos after

a lastgasp.mi abort

sensor: u_stale_gps_msg_time(s) 600

sensor: u_stale_gps_msg_period(s) 300 # in, In gliderdos msg delivered every u_stale_gps_msg_period

seconds if its been u_stale_gps_msg_time since

the last gps fix.

-1 (on either sensor) disables (no msg every delivered)

intended to alert shore side control to do a

"callback" when operating over iridium.

the msg: "NOTE:GPS fix is getting stale: X secs old"

sensor: m_1meg_persistor(bool) 0 # out, 1 if M_FREE_HEAP > U_HEAP_REQUIRED_FOR_1MEG_PERSISTOR

sensor: u_heap_required_for_1meg_persistor(bytes) 500000 # in, heap required for 1 MB persistor

sensor: m_why_started(enum) 255 # out, how GliderDos started

128 -> External (the reset button)

64 -> Power-On

32 -> Software Watchdog

16 -> Dbl Bus Fault

```
                                        # 4   -> Loss of Clock
                                        # 2   -> RESET instruction
                                        # 1   -> Test Submodule
                                        # 255 -> Uninitialized

sensor: c_heap_measurement_period(mins)          -1   # how often to measure the heap, <= 0 disables

sensor: m_min_spare_heap(bytes)                  -1   # out, minimum spare heap seen
sensor: sci_m_min_spare_heap(bytes)              -1   # and for science
sensor: m_min_free_heap(bytes)                   -1   # out, minimum free heap seen
sensor: sci_m_min_free_heap(bytes)               -1   # and for science

# --- Set in outer control loop,
# main_per.c
sensor: u_cycle_time(sec) 4.0            # in, num of secs/cycle on glider processor
sensor: u_low_power_cycle_time(sec) -1.0   # in, num of secs/cycle on glider processor
                                # during low power mode (dive/climbs),
                                # <=0 disables low power mode
sensor: u_sci_cycle_time(sec) 1.0        # in, num of secs/cycle on science processor

# calculated cycle time
sensor: x_cycle_time(sec) 4.0  # either u_cycle_time or u_low_power_cycle_time
sensor: x_low_power_status(nodim) 4.0  # why not low power?

sensor: u_max_sensor_logs_per_cycle(nodim) 4  # in, max high density sensor records
```

```
                                # per dbd/sbd logging cycle, valid
                                # range is 2 - 15
sensor: m_present_time(timestamp) 0    # out, secs since 1970 @ start of cycle
sensor: m_mission_start_time(timestamp) 0  # out, secs since 1970 @ start of mission
sensor: m_present_secs_into_mission(sec) 0 # out, secs since mission started
sensor: m_cycle_number(nodim) 0          # out, cycles since mission started

sensor: x_cycle_overrun_in_ms(msec) 0 # out, set every cycle
                                # the number of milliseconds that the
                                # cycle actually was compared to
                                # U_CYCLE_TIME
sensor: u_allowable_cycle_overrun(msec) 1000  # how large x_cycle_overrun_in_ms can
                                # before saying are_device_drivers_called_normally()
                                # For reasons that aren't clear to me, we are overrunning
                                # every cycle by 250ms.. someone should figure out why
                                # 14-Jun-05 tc@DinkumSoftware.com

   # These measure time in ms of various states
sensor: x_lc_time(msec)   0  # layered control
sensor: x_dc_time(msec)   0  # dynamic control
sensor: x_ds_time(msec)   0  # device scheduler
sensor: x_sp_time(msec)   0  # sensor processing
sensor: x_log_time(msec)  0  # log_data()
sensor: x_dead_time(msec) 0  # idle at end of loop
```

```
  # This is for the weighted average of the post device scheduler
  # processing time (sensor processing and logging)
sensor: x_avg_msecs_of_post_ds_processing_reqd(msec)   1000 # start here to speed up stabilization
sensor: u_avg_msecs_of_post_ds_processing_alpha(nodim) 0.75 # 0 - 1 (more weight to recent values)

# --- Strobe light sensor
sensor: c_strobe_ctrl(bool)          0 # boolean controller for the strobe light.
                                       # 0 = Off
                                       # 1 = On
sensor: m_strobe_ctrl(bool)          0 # boolean measurement for the strobe light.
                                       # 0 = Off
                                       # 1 = On

# --- layered_control.c

sensor: x_mission_num(nodim)         0 # out, YYDDxx the current or last mission number
                                       # Old style, before switch to DBD scheme
                                       # Kept for argos

sensor: x_mission_status(enum)  -3 # out, current (or last) mission status
sensor: x_old_mission_status_1(enum)   -3 # out, old,    status from prior missions
sensor: x_old_mission_status_2(enum)   -3 # out, older,  status from prior missions
sensor: x_old_mission_status_3(enum)   -3 # out, oldest, status from prior missions

                                  # New DBD style mission numbering
```

sensor: x_dbd_mission_number(nodim)   0.0 # out, mmmm of mmmmssss.dbd

sensor: x_dbd_segment_number(nodim)   0.0 #   ssss of mmmmssss.dbd

All these sensors "reflect" the values in struct command

See commands.h definition of XXX_mode_t for "mode" values

The cc_XXX variables are updated many times during a cycle

during the behavior resolution process in layered_control

The cc_final_XXX variables are updated once per cycle after

all the behaviors are resolved.

sensor: cc_heading_mode(enum) -1 # out, cmd->heading_mode

sensor: cc_heading_value(X)   0 #   argument for heading_mode

sensor: cc_pitch_mode(enum)   -1 # out, cmd->pitch_mode

sensor: cc_pitch_value(X)   0 #   argument for pitch_mode

sensor: cc_bpump_mode(enum)   -1 # out, cmd->bump_mode

sensor: cc_bpump_value(X)   0 #   argument for bpump_mode

sensor: cc_thruster_mode(enum)   -1 # out, cmd->thruster_mode

sensor: cc_thruster_value(X)   0 #   argument for thruster_mode

sensor: cc_threng_mode(enum) -1 # out, cmd->threng_mode

sensor: cc_inflection_mode(enum) -1 # out, cmd->inflection_mode

sensor: cc_depth_state_mode(enum) -1   # out, cmd->depth_state_mode

sensor: cc_mission_status_mode(enum) -3 # out, cmd->mission_status_mode

sensor: cc_is_comatose(bool)   0 # out, cmd->is_comatose

sensor: cc_time_til_inflect(s)   -1 # out, <0 ==> invalid

sensor: cc_final_heading_mode(enum) -1 # out, cmd->heading_mode

sensor: cc_final_heading_value(X) 0 # argument for heading_mode

sensor: cc_final_pitch_mode(enum) -1 # out, cmd->pitch_mode

sensor: cc_final_pitch_value(X) 0 # argument for pitch_mode

sensor: cc_final_thruster_mode(enum) -1 # out, cmd->thruster_mode

sensor: cc_final_thruster_value(X) 0 # argument for thruster_mode

sensor: cc_final_bpump_mode(enum) -1 # out, cmd->bump_mode

sensor: cc_final_bpump_value(X) 0 # argument for bpump_mode

sensor: cc_final_threng_mode(enum) -1 # out, cmd->threng_mode

sensor: cc_final_inflection_mode(enum) -1 # out, cmd->inflection_mode

sensor: cc_final_depth_state_mode(enum) -1 # out, cmd->depth_state_mode

sensor: cc_final_mission_status_mode(enum) -3 # out, cmd->mission_status_mode

sensor: cc_final_is_comatose(bool) 0 # out, cmd->is_comatose

sensor: cc_final_time_til_inflect(s) -1 # out, <0 ==> invalid

behavior specific

  # behavior specific, have not sorted it all out

sensor: c_depth(m) -1 # ascend.c, descend.c, glider_yo.c layer_control.c

  # surface

sensor: u_max_num_files_to_xmit_at_once(nodim) 30 #! visible = True; min = 1.0; max = 60.0

in, max files batched in sending

files from glider to shore

sensor: m_free_heap(bytes) -1 # out, the amount of free heap space

sensor: sci_m_free_heap(bytes) -1 # and for science

sensor: m_spare_heap(bytes) -1 # out, projected amt of heap if every

big consumer is activated.

sensor: sci_m_spare_heap(bytes) -1 # and for science

sensor: x_in_surface_dialog(nodim) 0 # out, non-zero means surface behavior

is in surface dialog and others

specifically behavior abend should

not try to read any chars. This is

a bitfield, with bit assigned to each

surface behavior by their behavior number

bit = 1 << (behavior_num-1)

sensor: x_num_bang_cmds_done(nodim) 0 # incremented every time a !cmd execute in

a surface dialogue, see secs_after_bang_cmd()

sensor: x_sent_data_files(nodim) 0 # set to the number of glider log files sent via last zmodem batch

set to 0 on failure.

sensor: sci_x_sent_data_files(nodim) 0 # set to the number of science log files sent via last zmodem batch

set to 0 on failure.

sensor: x_surface_active(nodim) 0 # (>0 means active) set to the surface behavior id that is active.

Only one surface behavior can be active at a given time. The first

surface behavior that goes active blocks other surface behaviors

until it leaves the active state.

Lens penetration sensors (c_stop_when_air_pump)

diveclimb.c

sensor: m_surface_depth_reached(bool) 0 # True if we've reached u_reqd_depth_at_surface.

sensor: u_pitch_surface(rad) 0.087 # In, expected pitch if we are at the surface/air bag is inflated. 5 deg

sensor: u_secs_surface_depth_reached(sec) 60 # In, if it has been this many seconds since m_surface_depth_re

sensor: x_why_lens_completed(nodim) 0 # Out, Why the 'c_stop_when_air_pump' surface behavior is complete

1: Reached depth and saw vacuum change (M_VACUUM_AIR_BAG_INFLATED > 0.0)

2: Reached depth and saw vacuum change (M_VACUUM_CHANGE_SINCE_AIR_PUMP_ON >= U_VACUUM_AIR_BAG_INFLATING)

and M_PITCH <= U_PITCH_SURFACE

3: Reached depth and time since the surface depth was reached (M_SURFACE_DEPTH_REACHED) > U_SECS_SURFACE_DEPTH_REACHED

vacuum.c

sensor: m_vacuum_change_since_air_pump_on(inHg) 0 # Change in vacuum (m_vacuum - m_vacuum_air_pump_on)

sensor: m_vacuum_air_bag_inflated(inHg) 0 # If m_vacuum_change_since_air_pump_on > u_vacuum_air_bag_inflated. Indicates that air bag is likely inflated

sensor: m_vacuum_air_pump_on(inHg) -1.0 # Initial value of m_vacuum when air pump is turned on AND depth < u_max_depth_for_air_pump_est

sensor: u_vacuum_air_bag_inflated(inHg) 1.5 # For m_vacuum_change_since_air_pump_on greater than this value, it is very likely that the air bag has inflated

sensor: u_vacuum_air_bag_inflating(inHg) 0.3 # For m_vacuum_change_since_air_pump_on greater than this value, it is very likely that the air bag has started to inflate

sensor: u_max_depth_for_air_pump_est(m) 8.0 # The maximum depth that we assume the air pump will inflate. Used only for determining m_vacuum_air_pump_on.

hydro_smp

sensor: dhs_valid(bool) 0 #non-zero means remaining sensors are valid

sensor: dhs_start_time(abstime) 0 #secs since 1970 GMT

sensor: dhs_duration(s) 0

sensor: dhs_gain(dB) 0

sensor: dhs_channel(nodim) 0

sensor: dhs_xmit_files(nodim) 0

sensor: dhs_silence_lvl(nodim) 0

sensor: dhs_sampling(bool) 0 # is set true when data collection in process

yo

sensor: c_reread_mafiles(bool) 0 # 1 -> reread mafile during a mission

sensor: c_climb_target_depth(m) -1.0 # in, value of b_arg for climb target depth

sensor: c_dive_target_depth(m) -1.0 # in, value of b_arg for dive target depth

drift_at_depth

Don't change intial values of computed x_hover_XXX sensors!

sensor: x_target_hover_depth(m) 0.0 # deduced from b_args: target_depth and target_altitude

sensor: x_avg_hover_depth(m) 0.0 # exponential mean of m_depth while hovering

sensor: x_hover_ballast(cc) 0.0 # adjusted hover_bpump_value for maintaining

neutral buoyancy at drift_at_depth target depth

sensor: m_avg_thruster_depth(m) 0.0 # out, if thruster is commanded, the average thruster depth for this segment.

sensor: m_avg_thruster_power_drift(watt) 0.0 # out, if thruster is commanded, the average thruster power for this segment.

sensor: x_avg_hover_ballast(cc) 0.0 # exponential mean of calculated

neutral ballast

sensor: u_avg_hover_ballast_alpha(nodim) 0.05 # more weight for longterm mean

used for maintaining a depth vs neutral buoyancy lookup table

sensor: x_hover_ballast_shallow(cc) 0.0 # the shallowest neutral buoyancy pumped

sensor: x_hover_ballast_deep(cc) 0.0 # the deepest neutral buoyancy pumped

sensor: x_hover_depth_shallow(m) 0.0 # the shallowest target drift depth

sensor: x_hover_depth_deep(m) 0.0 # the deepest target drift depth

sensor: x_hover_active(bool) 0 # drift_at_depth is in BS_HOVER substate

sensor: x_deactivate_hover(bool) 0 # a way to stop drift_at_depth

sensor: x_enable_steering_during_hover(bool) 0 # If true, allow steering when commanded to hover.

Set by drift_at_depth b_arg enable_steering

The bpump servo mode and pitching depth control mode specify two types of depth control methods,

as defined in drift_at_depth b_arg depth_ctrl.

They both use m_depth_error:

sensor: m_depth_error(m/s) 0.0 # Out, m_depth - commanded depth

sensor: m_depth_ierror(m) 0.0 # Out, integrated error

sensor: m_depth_derror(m/s^2) 0.0 # Out, time derivate of error

bpump servo mode (b_arg depth_ctrl = 1)

delta bpump = -Kp*(target_depth - glider_depth) + Kd*depth_rate

sensor: u_hover_bpump_ap_gain_shallow(cc/m) 1.0 #proportional gain, shallow bpump

sensor: u_hover_bpump_ap_dgain_shallow(cc-s/m) 1.0 #derivative gain, shallow bpump

sensor: u_hover_bpump_ap_gain_deep(cc/m) 1.0 #proportional gain, deep bpump

sensor: u_hover_bpump_ap_dgain_deep(cc-s/m) 1.0 #derivative gain, deep bpump

pitching depth control mode (b_arg depth_ctrl = 2)

pitch cmd = -(Kp*m_depth_error + Ki*m_depth_ierror + Kd*m_depth_derror)

sensor: x_hover_depth_p_gain(X) -0.15 # proportional gain Kp: should always be < 0. Set based upon DEPTH_GAIN_SCALE:

If b_arg DEPTH_GAIN_SCALE is true,

X_HOVER_DEPTH_P_GAIN = U_HOVER_DEPTH_GAIN_SCALE_M * M_THRUSTER_EST_SPEED + U_HOVER_DEPTH_GAIN_SCALE_B

If b_arg DEPTH_GAIN_SCALE is false,

X_HOVER_DEPTH_P_GAIN = b_arg DEPTH_P_GAIN

sensor: u_hover_depth_gain_scale_m(X) 0.429

sensor: u_hover_depth_gain_scale_b(X) -0.430

sensor: u_hover_depth_p_gain_min(X) -0.01 # In, limits the minimum value of X_HOVER_DEPTH_P_GAIN if scale is used

sensor: x_hover_depth_i_gain(X) 0 # Out, set by b_arg DEPTH_I_GAIN

sensor: x_hover_depth_d_gain(X) 0 # Out, set by b_arg DEPTH_D_GAIN

sensor: x_hover_depth_pitch_limit(rad) 0.174 # Out, x_hover_depth_pitch_limit = b_arg DEPTH_PITCH_LIMIT - X_PITCH_AP_DEADBAND

sensor: u_hover_depth_run_time(sec) -1 # How often to "run" the loop

<= 0, every cycle

> 0, this many seconds

sensor: u_hover_depth_pitch_deadband(X) -1

sensor: u_hover_depth_pitch_rate_deadband(X) -1

sensor: u_hover_depth_inflection_holdoff(sec) -1.0 # do not set. Required for PID controller structure,

but not used for pitching depth control

How long to not integrate after pitched at limit

sensor: u_hover_depth_hardover_holdoff(sec) 120.0 # in, how long to keep zeroing the integrated

error after fin is "hard over".

<= 0 causes no holdoff time, i.e. starts integrating

immediately after fin is NOT hardover.

sensor: u_hover_depth_scale_by_max(bool) 0 # Whether or not we scale the command by x_hover_depth_pitch_limit

sensor: u_hover_depth_deadband_reset(bool) 0 # in, If true, then reset the integrator if we are not changing pitch due to being in the deadband

If false, then hold the integral term constant while in deadband.

various clipping limits

sensor: u_hover_depth_limit_gain_x_error(rad) 1000.0 # Limits the gain*error term, (flattens gain curve)

Set it large to disable it.

sensor: u_hover_depth_limit_absolute(rad) 1000 # limits final C_PITCH value to beween +/- this value.

Set it large to disable it.

Note: this is also limited to the

pitch limit X_HOVER_DEPTH_PITCH_LIMIT

sensor: u_hover_depth_abort_after_y_misses(nodim) -1 # in, how many missed depth measurements

before we aborting the mission

<= 0 never abort

1 abort on first miss

>= 2 abort when miss this many times in a row

sensor: x_hover_depth_ap_ran(bool) -10 # Updated on a cycle where pitching depth control executed

-1 First time initialization

0 ; called, but chose not to command motor

1 ; did not run cause no fresh input

2 ; did not run cause too close to inflection

3 ; did not run cause not time to run yet

4 ; "ran", controlled motor

5 ; did not run cause within deadband

sensor: x_hover_depth_pitch_is_maxed(bool) 0 # true implies pitch is maxed

used for estimating trim offset

sensor: x_avg_depth_pitch_battpos_offset(in) 0.0 # exponential mean of calculated trim offset

sensor: u_avg_depth_pitch_battpos_alpha(nodim) 0.05 # more weight for longterm mean

sensor: u_avg_depth_pitch_battpos_deadband(m) 0.1 # depth error deadband

#must be above this to compute avg

--- dynamic control.c

For use in abort sequences, see doco/abort-sequences.txt

sensor: u_abort_min_burn_time(sec) 600 # Never drop the weight before this time

sensor: u_abort_max_burn_time(sec) 14400 # Always drop the weight after this time

sensor: u_abort_turn_time(sec) 300 # Max time it takes glider to "turn around vertically"

sensor: x_inflecting(bool) 0 # out, true implies in an inflection

sensor: m_tot_num_inflections(nodim) 0 # out, running count of number of inflections

sensor: m_last_yo_time(sec) 0.0 # out, twice the time between last inflections

sensor: m_avg_yo_time(sec) 60.0 # out, twice the average time between inflections

exponential average of m_last_yo_time

sensor: m_num_half_yos_in_segment(nodim) # out, number of dive/climbs since last surface

0 on first dive after surfacing

incremented on each inflection

sensor: u_fly_deep_in_shallow(bool) 0 #! visible = True

in, flying a deep glider in shallow water, so we want pitch control

to not wait for m_is_de_pump_moving. Adapt dynamic_control.c accordingly.

--- diveclimb.c

Start: sensors for autoballast/speed control. See /doco/how-it-works/autoballast.txt

Start: autoballast configuration sensors (i.e. sensors a user may wish to change before running a mission):

sensor: u_autoballast_end_on_converge(bool) 0 #in. If true, end autoballast adjustments to c_[climb/dive] once converged. If false, keep running autoballast.

sensor: u_autoballast_abort(bool) 0 #in, if autoballast fails to converge, then abort if true. if false, c_autoballast_state will change to 3 and continue with the last ballast amounts

sensor: c_autoballast_state(enum) 0 # in/out, describes the current state of autoballast. The user may also change its value to force autoballast to change state

0=uninitialized.

1=initialized, still converging

2=converged successfully

3=converged unsuccessfully: abs(c_dive_bpump) or c_climb_bpump > X_BALLAST_PUMPED_MAX/X_DE_OIL_VOL_MAX

4=converged unsuccessfully: c_climb_bpump - c_dive_bpump < c_autoballast_volume (d_bpump_value)

5=converged unsuccessfully: may be a bit more complicated. Consider examining .mlg

sensor: c_dive_bpump(X) -1000.0 # in/out, amt of ballast used in a dive in autoballast control.

sensor: c_climb_bpump(X) 1000.0 # in/out, amt of ballast used in a climb in autoballast control.

sensor: f_scale_pitch(X) 3.0 # in, value to scale pitch deadband in the wait_for_pitch routine

sensor: f_speed_min(m/s) 0.05 #in, minimum allowable speed input. Protects against excessively low c_speed_min b_args that could cause stall

sensor: f_min_ballast(X) 250.0 # in, the minimum amt of total ballast, limits c_autoballast_volume. Provided as a safety against unreasonably low b_arg

This value is the smallest total drive that we've historically used, however, if the user is comfortable you could be set

this value to be smaller.

sensor: f_max_ballast(X) 400.0 # in, the maximum amt of total ballast, limits c_autoballast_volume. Provided as a safety against unreasonably high b_arg

If using a deep engine capable of larger displacement, it's conceivable that you might want to use a larger value.

If so, feel free to set this value higher.

sensor: f_min_pump(X) 10.0 #in, the minimum amt of delta ballast for use in a climb or dive .

End: autoballast configuration sensors

Start: autoballast measurement sensors

sensor: m_dive_tot_time(s) -1.0 # out, amount of time to complete dive

sensor: m_climb_tot_time(s) -1.0 # out, amount of time to complete climb

sensor: m_dive_depth(m) -1.0 # out, depth that dive actually achieves (to determine if veh has inflected early.)

sensor: c_speed_ctrl(bool) 0 # out, Gets set to true by software if speed control (SM_SPEED, dynamic_control) was used in this dive or climb.

End: autoballast measurement sensors

Start: autoballast storage sensors (sensors defined in b_args, stored in these sensors)

sensor: c_autoballast_volume(X) 1000.0 #in/out, stores the user specified total amt of ballast. Read in as b_arg_bpump_value and recorded to this sensor

sensor: c_wait_for_pitch(bool) 1 #in, if true, wait for pitch/batt pos dynamics to settle before enabling speed control. Set by yo b_arg, and stored in this sensor.

sensor: c_wait_for_ballast(sec) 100.0 #in, wait this many seconds after ballast pump has stopped moving (after inflection only)

#before enabling speed control. Set by yo b_arg, and stored in this sensor.

sensor: f_depth_rate_method(enum) 3 # in, method of filtered depth rate to use for speed control. Set by yo b_arg, and stored in this sensor.

0= raw m_depth_rate

1= m_depth_rate_subsample

2 = tbd.

3 = running average. uses m_depth_rate_avg_final

tbd other methods:

filter out unreasonable depth rates, combos of those above, etc.

sensor: c_delta_bpump_speed(X) 50.0 #in/out, amount of ballast to add to bpump in order to reach desired speed. Should always be positive. SM_SPEED takes care of the sign

Set by yo b_arg, and stored in this sensor.

sensor: c_delta_bpump_ballast(X) -1.0 #in/out, amount of ballast to add to bpump in order to converge on ballast.

#If < deadband, diveclimb.c will set it to deadband

Set by yo b_arg or changed by autoballast routine, and stored in this sensor.

sensor: c_time_ratio(X) 1.1 # in, ratio of climb/dive times that must be maintained for speed control. Set by yo b_arg, and stored in this sensor.

sensor: c_use_sc_model(bool) 0 #out, if using model of veh for SM_SPEED. Always set to 0 for now until the model is designed. Read in as b_arg_bpump_value and recorded to this sensor

sensor: c_speed_max(m/s) -100.0 # in/out, fastest depth rate allowed for SM_SPEED control. Gets set separately by user b_arg for dive and climb

sensor: c_speed_min(m/s) -100.0 # in/out, slowest depth rate allowed for SM_SPEED control. Gets set separately by user b_arg for dive and climb

End: autoballast storage sensors

End: sensors for autoballast/speed control.

sensor: c_speed(m/s) -1 # out, horizontal speed, <0 means no speed specified

sensor: dc_c_ballast_pumped(cc) 0 # out, what dynamic control wants ballast to be

sensor: f_neutral_ballast(cc) 0 # in, amt of ballast for neutral (~0)

sensor: c_pitch(rad) 0 # out, commanded pitch, <0 to dive

sensor: dc_c_battpos(in) 0 # out, what dynamic control wants fore/aft battery to be

sensor: dc_c_fluid_pumped(cc) 0 # out, what dynamic control wants fore/aft fluid to be

sensor: dc_c_thermal_updown(enum) # out, what dynamic_control wants thermal engine to do

sensor: dc_c_de_updown(enum) # out, what dynamic_control wants deep electric engine to do

sensor: dc_c_oil_volume(cc) 0 # out, what dynamic control wants oil volume to be

sensor: f_neutral_oil_volume(cc) 0 # in, amt of oil volume for neutral (~0)

also used in g_shell.c: GCmdBallast()

Generic location stuff, see coord_sys.h for description

sensor: x_lmc_utm_zone_digit(byte) 0 # The utm zone of lmc (0,0)

sensor: x_lmc_utm_zone_char(byte) 0 # ditto, 0->A 1->B etc

sensor: x_utm_to_lmc_00(nodim) 0 # matrix such that: lmc = [] * utm + off

sensor: x_utm_to_lmc_01(nodim) 0 # |x| |00 01| (|e| |x0|)

sensor: x_utm_to_lmc_10(nodim) 0 # | | = | | * (| | + | |)

sensor: x_utm_to_lmc_11(nodim) 0 # |y| |10 11| (|n| |y0|)

sensor: x_utm_to_lmc_x0(nodim) 0

sensor: x_utm_to_lmc_y0(nodim) 0

#The first pair are to record current vehicle UTM zone

#The next six convert (northing,easting) -> (x,y).

#All of these are computed when the origin in LMC is established.

#And the last two are used to correct for lon UTM zone and equator crossings.

These store the vehicles zone (for detecting boundry crossing)

and the correction for computing vehicle's lat/lon from lmc position

sensor: x_lmc_utm_veh_zone_digit(byte) 0 # The utm zone of the vehicle

sensor: x_lmc_utm_veh_zone_char(byte) 0 # ditto, 0->A 1->B etc

sensor: x_lmc_utm_veh_easting_correction(m) 0 # needed for crossing lon UTM zones

sensor: x_lmc_utm_veh_northing_correction(m) 0 # needed for crossing equator

Generic heading related stuff

sensor: c_heading(rad) 0   # out, commanded heading

sensor: c_roll(rad) 0   # out, commanded roll

sensor: dc_c_battroll(rad) 0 # out, what dynamic control wants roll battery to be

sensor: f_battroll_offset(rad) 0.0 # in, added to c_roll to handle off center batteries

sensor: m_hdg_error(rad)   0 # out, m_heading - c_heading

sensor: m_hdg_ierror(rad-sec) 0 # out, integrated m_hdg_error

sensor: m_hdg_derror(rad/sec) 0 # out, rate of change of m_hdg_error

```
 # Waypoint control
sensor: u_use_current_correction(nodim) 1   #! visible = True
                                #  0 calculate, but do not use m_water_vx/y
                                #  1 use m_water_vx/y to navigate AND aim

sensor: c_wpt_x_lmc(m)   0 # in, command waypoint in lmc units
sensor: c_wpt_y_lmc(m)   0 #
sensor: x_hit_a_waypoint(bool) 0 # set by behavior when reach a waypoint
sensor: x_last_wpt_x_lmc(m)    0 # set by behavior when reach a waypoint
sensor: x_last_wpt_y_lmc(m)    0

 # Heading autopilot variables
 # Mostly parameteric inputs to control autopilot
 # The X_ guys are working variables
 # See doco/how-it-works/heading_autopilot.txt

  # servo on Heading by adjusting fin
    # controls/knobs: the user might change these
    # read glider/doco/how-it-works/heading_autopilot.txt
sensor: u_hd_fin_ap_gain(1/rad)      1.50 # The "gain" of controller: 57 deg proportional band
                              # 1/57 deg
sensor: u_hd_fin_ap_igain(1/rad-sec) 0.02
sensor: u_hd_fin_ap_dgain(sec/rad) -4.00
                            # percent C_FIN = (-U_HD_FIN_AP_GAIN  * M_HDG_ERROR) +
                            #              (-U_HD_FIN_AP_IGAIN * M_HDG_IERROR)+
```

```
#               (-U_HD_FIN_AP_DGAIN * M_HDG_DERROR)

# For u_low_power_cycle_time(s) = 30
sensor: u_low_power_hd_fin_ap_gain(1/rad)      0.5  #
sensor: u_low_power_hd_fin_ap_igain(1/rad-sec) 0.0001 #
sensor: u_low_power_hd_fin_ap_dgain(sec/rad) 0.0 #

# For thruster installed.
sensor: u_thruster_hd_fin_ap_gain(1/rad)      0.5   #
sensor: u_thruster_hd_fin_ap_igain(1/rad-sec) 0.004 #
sensor: u_thruster_hd_fin_ap_dgain(sec/rad) -4.0 #

# These get set to either u_hd_fin_ap_gain(igain,dgain) or
# u_thruster_hd_fin_ap_gain(igain,dgain) if thruster installed or
# u_low_power_hd_fin_ap_gain(igain,dgain) depending on the value of x_cycle_time.
# Initially set to the default values of u_hd_fin_ap_gain(igain,dgain).
sensor: x_hd_fin_ap_gain(1/rad)       1.50
sensor: x_hd_fin_ap_igain(1/rad-sec)  0.03
sensor: x_hd_fin_ap_dgain(sec/rad) -4.00

sensor: u_hd_fin_ap_run_time(secs)  -1  # How often to "run" the loop
                                        # <= 0, every cycle
                                        #  > 0, this many seconds
sensor: u_hd_fin_ap_scale_by_max(bool) 1 # Whether or not we scale the command by X_FIN_MAX
```

What to do around inflections

sensor: u_hd_fin_ap_inflection_holdoff(sec) -1.0 # in, controls steering around inflections

-1 always steer/integrate_errors during inflection

>=0 don't steer/integrate errors:

during inflection --AND--

for this many secs after START of inflection

Deadbands on heading error and rate error. If heading error and rate error are both within these deadbands, don't move the fin.

To disable completely (i.e. ignore if in deadband or not), set either value to -1.

To turn off one deadband but not the other, set the one you want to turn off to a very high value,

i.e. want to turn off deadband on rate error: set u_heading_rate_deadband 1000.0

sensor: u_heading_deadband(rad) 0.087 #in, deadband for heading error M_HDG_ERROR

sensor: u_heading_rate_deadband(rad/s) 0.0087 #in, deadband for heading rate error M_HDG_DERROR

sensor: u_hd_fin_ap_deadband_reset(bool) 0 # in, If true, then reset the integrator if we are not moving the fin due to being in the deadband

If false, then hold the integral term constant while in deadband.

How long to not integrate after big course changes

sensor: u_hd_fin_ap_hardover_holdoff(sec) 120.0 # in, how long to keep zeroing the integrated

error after fin is "hard over".

<= 0 causes no holdoff time, i.e. starts integrating

immediately after fin is NOT hardover.

various clipping limits

sensor: u_hd_fin_ap_limit_gain_x_error(rad) 1000.0 # Limits the gain*error term, (flattens gain curve)

Set it large to disable it.

sensor: u_hd_fin_ap_limit_absolute(rad) 1000 # limits final C_FIN value to beween +/- this value.

Set it large to disable it.

Note: this is also limited to the

fin safety limit X_FIN_MAX.

sensor: u_hd_fin_abort_after_y_misses(nodim) 5.0 # in, how many missed attitude measurements

before we aborting the mission

<= 0 never abort

1 abort on first miss

>= 2 abort when miss this many times in a row

state: user shouldn't change, they are outputs only

sensor: x_hd_fin_ap_ran(bool) -10 # Updated on a cycle where heading autopilot executed

-1 First time initialization

0 ; called, but chose not to command motor

1 ; did not run cause no fresh input

2 ; did not run cause too close to inflection

3 ; did not run cause not time to run yet

4 ; "ran", controlled motor

5 ; did not run cause within deadband

sensor: x_hd_fin_ap_is_hardover(bool) 0 # true implies fin is "hardover"

sensor: x_heading_reversal(rad) 2.96 # in, if x_heading_reversal <= heading error <= (2*pi-x_heading_reversal), then

we are trying to reverse directions

servo on Heading by adjusting battery roll

Note: These all are parallels of X_hd_fin_XXX.

See those variables for a description.

The Version 1 of battery steering wasn't tested

on a battery steered glider when implemented. These

settings probably have to be changed.

sensor: u_hd_broll_ap_gain(1/rad) 1.00

sensor: u_hd_broll_ap_igain(1/rad-sec) 0.03

sensor: u_hd_broll_ap_dgain(1/rad-sec) 0.00 #Never tested

sensor: u_hd_broll_ap_run_time(secs) -1.0

sensor: u_hd_broll_ap_inflection_holdoff(sec) -1.0

sensor: u_hd_broll_ap_hardover_holdoff(sec) 400.0

sensor: u_hd_broll_ap_deadband_reset(bool) 0

sensor: u_hd_broll_ap_limit_gain_x_error(rad) 1000

sensor: u_hd_broll_ap_limit_absolute(rad) 1000

sensor: u_hd_broll_abort_after_y_misses(nodim) 3.0

sensor: u_hd_broll_ap_scale_by_max(bool) 1 # Whether or not we scale the command by X_BATTROLL_MAX

sensor: x_hd_broll_ap_ran(bool) -10

sensor: x_hd_broll_ap_is_hardover(bool) 0

Pitch autopilot variables

specify the curve relating vehicle pitch to battery postion

pitch(rad) = F_PITCH_BATTPOS_CAL_M(rad/in) * battpos(in) + F_PITCH_BATTPOS_CAL_B(in)

note: signs on pitch/battpos are documented under C_PITCH and C_BATTPOS

values for amy from lake seneca

sensor: f_pitch_battpos_cal_m(rad/in) -1.2565 # input

sensor: f_pitch_battpos_cal_b(in) 0.055 # input

specify the curve relating vehicle pitch to fluid pumped

# pitch(rad) = F_PITCH_FLUID_PUMPED_CAL_M(rad/cc) * fluid_pumped(cc) +
F_PITCH_FLUID_PUMPED_CAL_B(cc)

note: signs on pitch/fluid_pumped are documented under C_PITCH and C_FLUID_PUMPED values

sensor: f_pitch_fluid_pumped_cal_m(rad/cc) -0.0043 # input

sensor: f_pitch_fluid_pumped_cal_b(cc) 0.0 # input

 # Mostly parameteric inputs to control servo

 # The X_ guys are working variables

 # Cloned from heading_autopilot, See doco/heading_autopilot.txt

sensor: u_max_pitch_ap_period(sec) 60 # 16 AutoPilot "runs" at least this often

sensor: u_min_pitch_ap_period(sec) 2 # AutoPilot "runs" no more than this often

sensor: x_pitch_ap_period(sec) 0 # Actual computed time until next running of autopilot

sensor: x_pitch_ap_ran(bool) 0 # Updated on a cycle where pitch autopilot executed

Pitch servo control consists of a Proportional(P) - Derivative(D) controller

percent delta C_BATTPOS =

-X_PITCH_AP_GAIN * M_PITCH_ERROR

+X_PITCH_AP_DGAIN * M_PITCH_DERROR

Proportional gain for pitch servo control (value should always be < 0):

sensor: x_pitch_ap_gain(1/rad) -3.0 # Set based upon operating conditions:

If thruster is not commanded,

X_PITCH_AP_GAIN = U_PITCH_AP_GAIN

If thruster is commanded

X_PITCH_AP_GAIN = U_PITCH_AP_GAIN_THRUSTER

sensor: u_pitch_ap_gain(1/rad) -3.0 # Proportional gain of controller for no thruster

sensor: u_pitch_ap_gain_thruster(1/rad) -2.0 # The "gain" of controller if thruster is commanded

Derivative gain for pitch servo control (value should always be > 0):

sensor: x_pitch_ap_dgain(s/rad) 1.0 # Set based upon operating conditions:

If thruster is not commanded, X_PITCH_AP_DGAIN = U_PITCH_AP_DGAIN

If thruster is commanded, X_PITCH_AP_GAIN = U_PITCH_AP_DGAIN_THRUSTER

sensor: u_pitch_ap_dgain(s/rad) 1.0 # X_PITCH_AP_DGAIN = U_PITCH_AP_DGAIN if thruster is not commanded

sensor: u_pitch_ap_dgain_thruster(s/rad) 1.0 # X_PITCH_AP_GAIN = U_PITCH_AP_DGAIN_THRUSTER if thruster is commanded

sensor: x_pitch_ap_deadband(rad) 0.0524 # set to u_pitch_ap_deadband_thruster if thruster is commanded,

otherwise set to u_pitch_ap_deadband

The deadband + or - from C_PITCH,

We do not make corrections if

abs(M_PITCH_ERROR) < X_PITCH_AP_DEADBAND

sensor: u_pitch_ap_deadband(rad) 0.0524 # 3 deg. Deadband for thruster not commanded

sensor: u_pitch_ap_deadband_thruster(rad) 0.0524 # 3 deg. Deadband for thruster commanded

sensor: u_pitch_max_delta_battpos(in) 0.20 # 40% of deadband

in, max delta battpos to apply

a really big number sets no limit and is safe

somebody else clips later on

sensor: u_pitch_max_delta_fluid_pumped(cc) 4.0 # 40% of deadband

in, max delta fluid_pumped to apply

a really big number sets no limit and is safe

somebody else clips later on

sensor: u_pitch_correction_time_mult(nodim) 0.50 # What fraction assumed correction time we wait before

running again.

sensor: u_pitch_deadband_time_mult(nodim) 2.0 # How much we increase the time til next attempt if

we are in the dead band.

sensor: m_pitch_error(rad) 0 # out, difference between m_pitch - c_pitch

sensor: m_pitch_derror(rad/s) 0 # out, time derivative of m_pitch_error

sensor: x_battpos_achieved(enum) 0 #out, Describes the state of the initial battpos searching for pitch servo mode.

0=Haven't gotten to desired starting position C_CLIMB/DIVE_BATTPOS.

1=Gotten to desired starting position, but haven't collected enough samples within pitch db

2=Achieved state 1, and collected enough samples. Calc a running average of battpos

and store this in C_DIVE/CLIMB_BATTPOS

sensor: u_use_pitch_servo_memory(bool) 1 #in, if true, enable pitch servo memory

--- sensor_processing.c

sensor: x_sensor_processing_ran(bool) 0 # out, updated on every cycle

Used to compute integration times

sensor: m_tot_horz_dist(km) 0.0 # out, How far we have moved underwater

sensor: x_current_target_altitude(m) -1.0 # default is none, height above

bottom glider is currently

diving/climbing to

sensor: u_print_engine_status(sec) -1.0 # controls printing of thermal/deep electric status

<0 do not print >0 print status that often

compute_depth_stuff()

NOTE: Raw depth data (m_depth) is noisy. Depth rate for purposes of depth state evaluation

(m_depth_rate_subsampled) will be based on subsampled depth data (m_depth_subsampled)

in order to minimize false reversals, false motion, and false stalls.

We take a long enough interval between depth measurements for subsampling, so that

depth state does not have a significant occurrence of false reversals, false motion,

and false stalls.

sensor: f_depth_subsampling_rate(sec) -1 # in, time rate of subsampled depth (will be APPROX!)

0 ==> no subsampling

<0 ==> autodetect based on is_deep

sensor: f_depth_subsampling_rate_default_deep(sec) 23 # auto value for deep

sensor: f_depth_subsampling_rate_default_shallow(sec) 0 # auto value for shallow

sensor: m_depth_subsampled(m) 0 # out, subsampled depth measurement

sensor: m_depth_rate(m/s) 0 # out, rate of change of depth, >0 is down

sensor: m_depth_rate_subsampled(m/s) 0 # out, subsampled depth rate measurement

sensor: m_avg_depth_rate(m/s) 0 # out, avg rate of change of depth, >0 is down

sensor: m_avg_climb_rate(m/s) 0 # out, avg rate of change of depth when climbing

sensor: m_avg_dive_rate(m/s) 0 # out, avg rate of change of depth when diving

sensor: u_avg_depth_rate_alpha(nodim) 0.25 # in, time constant for exponential averaging of

m_depth_rate ==> m_avg_depth_rate

1==> no averaging, i.e.

m_avg_depth_rate = m_depth_rate

smaller numbers (>0) ==> longer time constant

Calculate an average depth rate (m_depth_rate_avg_final) over a finite number of samples (c_depth_rate_running_avg_num).

The running average is stored as m_depth_rate_running_avg, the current sample number is stored as m_depth_rate_running_avg_n

See calc_running_avg() in sensor_processing.c

sensor: m_depth_rate_avg_final(m/s) 0.0 # Final value of the calculation. Use this!

sensor: m_depth_rate_running_avg(m/s) 0.0 # out, a running average calculation, used in diveclimb.c and sensor_processing.c

sensor: m_depth_rate_running_avg_n(enum) 0 # out, identifies the data sample # "n" of m_depth_rate_running_avg

sensor: c_depth_rate_running_avg_num(enum) 10 # the number of data samples to collect for the m_depth_rate_running_avg calculation.

sensor: u_reqd_depth_at_surface(m) 2 #! visible = True; min = 1.0; max = 10.0

in, depths less than this considered "at surface"

sensor: u_hovering_frac_nom_dive_rate(nodim) 0.25 # in, fraction of f_nominal_dive_rate

used as threshold for hovering

clips to 0-1

sensor: m_depth_state(enum) 0 # based on m_depth_rate and u_surface_depth

matches CC_DEPTH_STATE_MODE (enum depth_state_mode_t)

compute_surface_estimate()

These run 0 to 1 and are estimates we are at the surface

sensor: m_surface_est_cmd(nodim) 0 # commanded to surface

sensor: m_surface_est_ctd(nodim)   0   # ctd pressure => depth

sensor: m_surface_est_gps(nodim)   0   # gps talking to satellite

sensor: m_surface_est_fw(nodim)   0   # freewave has carrier

sensor: m_surface_est_irid(nodim) 0   # iridium has carrier

sensor: u_surface_est_time_constant(secs) 30 # m_surface_est_XXX expontially decayed

by this when corresponding condition is false

sensor: m_surface_est_total(nodim)   0 # sum of above m_surface_est_XXX ....

sensor: u_surface_est_threshold(nodim) 1.5 # and are compared to this

in order to set...

sensor: m_appear_to_be_at_surface(bool) 0 # The final result

sensor: m_certainly_at_surface(bool)   0 # true if got a gps fix, or freewave/iridium carrier

on this cycle.

compute_altitude_stuff()

sensor: u_alt_reduced_usage_mode(bool) 1 #! visible = True

in, default is on, 0 -> off

reduced usage mode turns on

altimeter only when necessary

sensor: x_alt_time(sec) 0   # out, calculated c_alt_time value

<0 altimeter off, =0 as fast as possible,

>0 that many seconds between measurements

sensor: m_altitude(m) 0 # out, height above the bottom

sensor: m_altitude_rate(m/s) 0 # out, rate of change of altitude, <0 is down

sensor: m_altimeter_status(enum) 0 # out, 0 is good reading

non-zero means rejected

see sensor_processing.h for codes

sensor: u_min_altimeter(m) 2.0 # in, altimeter reading must be between these(inclusive)

sensor: u_max_altimeter(m) 100.0 # the maximum range of the altimeter

sensor: m_aground_water_depth(m) -1 # out, set by behavior dive_to when it crashes

into bottom

sensor: m_water_depth(m) -1.0 # out, m_depth + m_altitude.

-1 ==> unknown

sensor: u_max_water_depth_lifetime(yos) 1.0 #! visible = True

in, how long we can use m_depth in absence

of measured data

sensor: u_max_bottom_slope(m/m) 3.0 # in, max slope of bottom. <0 disables all filters

max change in altitude/horizontal movement

sensor: u_min_water_depth(m) 0 # in, altimeter reading + M_DEPTH must be between these

sensor: u_max_water_depth(m) 2000 # inclusive

compute_alt_measure_delay()

```
sensor: u_alt_measure_secs_prior_inflection(sec) 15.0 # seconds prior to
                                        # inflection to start
                                        # measuring continuously
                                        # min legal value is 15.0 secs
sensor: u_alt_measure_fraction(nodim) 0.5  # must be > 0 and < 1, fraction
                                # of time till inflection to measure
                                # altitude, used in reduced-usage mode

# compute_heading_rate()
sensor: m_hdg_rate(rad/sec) 0  # rate of change of heading


# compute_vehicle_velocity()
sensor: m_speed(m/s)  0          # out, vehicle horizontal speed THRU WATER
sensor: m_is_speed_estimated(bool) 0 # out, Tells if m_speed is computed from
                        # M_DEPTH_RATE and M_PITCH -or-
                        # estimated. If M_PITCH is too small, estimate is
                        # from M_MISSION_AVG_SPEED_DIVING/CLIMBING.
                        # If thruster is installed and M_PITCH or M_DEPTH_RATE
                        # are too small, estimate comes from input voltage/current.


sensor: m_avg_speed(m/s)  0  # out, avg vehicle horizontal speed THRU WATER
                    # used only computing C_HEADING to way point
sensor: u_avg_speed_alpha(nodim) 0.001  # in, time constant for exponential averaging of
```

```
                    #   m_speed ==> m_avg_speed
                    #     1==> no averaging, i.e. m_avg_speed = m_speed
                    #     smaller numbers (>0) ==> longer time constant
sensor: u_calc_angle_of_attack(bool) 1 #! visible = True
                                        # in, whether or not to make an angle of attack calculation
sensor: u_angle_of_attack(rad) 0  # The angle of attack is used in the speed calculation
                    # and is a function of pitch. In reality, the glide angle
                    # is slightly steeper than the pitch. The difference, the
                    # angle of attack, allows the wings (and body) to generate
                    # lift to transfer vertical to horizontal velocity. The
                    # angle of attack is generally small (2 degrees or so)
                    # but still can account for errors in horizontal
                    # speed of 2-3 cm/s.

sensor: m_mission_avg_speed_diving(m/s)   0    # out, running average of computed m_speed
sensor: m_mission_avg_speed_climbing(m/s) 0    # since start of mission.  Used to estimate
                        # M_SPEED when M_PITCH is too small (< 11 deg)
sensor: u_coast_time(s)             7.5     # in, how long it takes the gliders
                        # horizontal speed to go to 0 due to drag
                        # Used when estimating M_SPEED by linearly
                        # reducing M_MISSION_AVG_SPEED_* to 0 over
                        # this time
                        # <0 ==> disables the damping
                    # Note: see sensor_processing.c:damp_horz_speed()
                    #      for justification of this time
```

sensor: m_vx_lmc(m/s) 0 # out, vehicle horizontal velocity OVER GROUND

sensor: m_vy_lmc(m/s) 0

get_est_horz_speed_thruster()

sensor: m_thruster_est_speed(m/s) 0.0 #Out, estimated forward speed (in body-frame) due to thruster

sensor: u_max_thruster_speed(m/s) 1.5 # Max estimated thruster speed. Limits M_THRUSTER_EST_SPEED and scales U_PITCH_AP_GAIN_THRUSTER

sensor: u_avg_thruster_speed_num(enum) 10 # In, number of samples to use for m_avg_thruster_speed

sensor: m_avg_thruster_speed(m/s) 0.0 #Out, average horizontal speed due to thruster

There are two set of coefficients used to estimate forward speed due to the thruster.

The coefficients are determined off-line, based upon a 52kg glider,

12V 4W Re-Max17 with 29:1 gearbox and 9x7 off the shelf prop.

Eventually we'll have a program that calculates the coefficients based upon vehicle drag estimates and motor/propeller set.

f_thruster_v[0,1] relate estimated input voltage to forward speed,

for thruster motor controller without current sensing (x_thruster_has_current_sense = 0)

m_thruster_est_speed = f_thruster_v1*input_voltage + f_thruster_v0

sensor: f_thruster_v1(m/s-volts) 0.0631 # In, 1st-order coefficient

sensor: f_thruster_v0(m/s) 0.0288 # In, 0th-order coefficient

f_thruster_i[0,1,2] relate measured motor current to forward speed,

for thruster motor controller with current sensing (x_thruster_has_current_sense = 1)

m_thruster_est_speed = i2*current^2 + i1*current + i0

sensor: f_thruster_i0(m-s) 0.1473

sensor: f_thruster_i1(m/s-amp) 0.9018

sensor: f_thruster_i2(m/s-amp-amp) -0.2083

compute_water_velocity() See doco/water-velocity-caclulation.txt

sensor: m_water_vx(m/s) 0 # in/out How fast the water is going. LMC coord. sys.

sensor: m_water_vy(m/s) 0 # used as input here (if u_use_current_correction is true)

sensor: m_initial_water_vx(m/s) 0 # out, initial computation of m_water_vx/y

sensor: m_initial_water_vy(m/s) 0 #

sensor: m_final_water_vx(m/s) 0 # out, initial computation of m_water_vx/y

sensor: m_final_water_vy(m/s) 0 #

sensor: m_water_delta_vx(m/s) 0 # out, change in water_vx/vy this segment

sensor: m_water_delta_vy(m/s) 0 #

both computed in compute_water_velocity() when get gps fix.

sensor: x_prior_seg_water_vx(m/s) 0 # in/out water speed used for navigation on prior segment

sensor: x_prior_seg_water_vy(m/s) 0

sensor: u_max_water_speed(m/s) 2.8 # in, 5 knots

magnitude of (m_water_vx,m_water_vy) clipped to this

These are part of the state machine used in computing water velocity

See doco/water-velocity-calculation.txt for writeup

sensor: x_dr_state(enum) 0.0 # out, mission_start=0, underwater=1,awaiting_fix=2,

awaiting_postfix=3, awaiting_dive=4

sensor: m_dr_time(sec) -1.0 # out, how long underwater, subject to currents

sensor: m_dr_surf_x_lmc(m) 0 # Dead Reckoned location when surface

sensor: m_dr_surf_y_lmc(m) 0

sensor: m_dr_fix_time(sec) -1.0 # out, surface drift time til first gps fix

sensor: m_gps_fix_x_lmc(m) 0 # location of first gps fix

sensor: m_gps_fix_y_lmc(m) 0

sensor: m_dr_x_ini_err(m) 0 # out, m_gps_fix_x/y_lmc - m_dr_surf_x/y_lmc

sensor: m_dr_y_ini_err(m) 0

sensor: m_dr_postfix_time(sec) -1.0 # out, surface drift time til later gps fix that is

used to correct for surface drift during

m_dr_fix_time

sensor: m_gps_postfix_x_lmc(m) 0

sensor: m_gps_postfix_y_lmc(m) 0 # Location used to measure surface drift

sensor: m_dr_x_postfix_drift(m) 0 # out, m_gps_postfix_x/y_lmc - x_gps_fix_x/y_lmc

sensor: m_dr_y_postfix_drift(m) 0

sensor: m_dr_x_ta_postfix_drift(m) 0 # out, m_dr_x/y_postfix_drift * time adjusted value

sensor: m_dr_y_ta_postfix_drift(m) 0

sensor: m_dr_x_actual_err(m) 0 # out, m_dr_x/y_ini_err - timeadj(m_dr_x/y_postfix_drift)

sensor: m_dr_y_actual_err(m) 0

compute_lmc_position()

sensor: m_x_lmc(m) 0 # vehicle position in Local Mission Coordinates

sensor: m_y_lmc(m) 0 # (0,0) at mission start Y axis is magnetic north

sensor: x_lmc_xy_source(enum) 0 # out, how m_x/y_lmc was computed this cycle

>= 0 means an (x,y) was computed

3 gps (surface)

2 dead reckon(uw)

1 dr estimated speed (uw)

0 inited to (0,0) first cycle of mission

-1 not computed cause at surface and no gps fix this cycle

-2 not computed cause no DR data (cycle overrun?)

-10 indicates software error, you should never see this

compute_waypoint_metrics()

sensor: m_dist_to_wpt(m) # out, How far to (c_wpt_x_lmc,c_wpt_y_lmc)

sensor: m_vmg_to_wpt(m/s) # out, Velocity Made good to (c_wpt_x_lmc,c_wpt_y_lmc)

sensor: m_time_til_wpt(s) # out, m_dist_to_wpt / m_vmg_to_wpt

```
# translate_to_latlon()
sensor: m_lat(lat)   69696969  # vehicle position in latitude
sensor: m_lon(lon)   69696969  # vehicle position in longitude
sensor: c_wpt_lat(lat) 0 # current waypoint in latitude
sensor: c_wpt_lon(lon) 0 # current waypoint in longitude
sensor: x_last_wpt_lat(lat) # last achieved waypoint
sensor: x_last_wpt_lon(lon)

# compute_comms_stuff
sensor: u_stable_comms_reqd_secs(sec)  60.0 # in, continous seconds of carrier detect
                        #     required to have stable comms
sensor: m_stable_comms(bool) 0.0          # out, true-> comms are stable, i.e. we have
                        #      had m_console_cd for reqd number of secs
                        #      in a row
sensor: u_zmodem_verbosity(nodim)      29.0 # in, controls output to config\zmodem.log
                        # the higher the number, the more output
                        # see zmdebug.h for a description
# compute_time_to_surface
sensor: m_est_time_to_surface(sec)     0.0  # An estimate of the time to climb to
                        # the surface from the current depth.
# compute_avg_inflection_time
sensor: m_avg_upward_inflection_time(sec)   12.0 # exponential average of inflections
sensor: m_avg_downward_inflection_time(sec) 12.0 # start with reasonable guess

# --- device driver level
```

sensor: m_device_drivers_called_abnormally(nodim) 0 # non-zero means time base is suspect because

glider busy, after data transmission, etc

It is results of:

devsched.c:device_drivers_called_normally()

It is a bit-field, there is a bit set for

each of the possible reasons. See top of

devsched.c for definitions (#define DDCA_xxx)

sensor: m_device_oddity(nodim) -1.0 # These set to the device number of offending device

sensor: m_device_warning(nodim) -1.0 # whenever it generates error/warning/oddity

sensor: m_device_error(nodim) -1.0

sensor: f_max_time_per_device_ctrl(msec) 500 # In, default max allowable time for

a device driver to run. oddities

generated if this time execeeded

sensor: f_noise_floor(volts) 0.050 # Electrical noise in system

Used to compute how often motor

velocities are computed and checked

sensor: f_crush_depth(m) 225.0 # When the glider gets crushed

sensor: f_time_to_burn_wire(sec) 20.0 # How long it takes burn wire to drop weight

sensor: m_at_risk_depth(m) 221.0 # When have to start burning the wire to drop the

in order to drop the weight before f_crush_depth

```
                        #  when diving at f_nominal_dive_rate
                        #  default=    225m - 20s * 0.19 m/s =


# common to all motors
sensor: u_motor_debug(nodim)          0   # bitmask:
                              # 0x0000000000000001          1    print motor travel stats at end of motion


sensor: u_comatose_enabled(bool)      0.0   #! visible = True
                              # in, true->enables comatose mode
sensor: u_comatose_deadband_mult(nodim) 10.0 # in, how much to increase motor deadbands
                              #     when in comatose mode


sensor: u_motor_fs_travel_mult(nodim)  2.0 # in, used to compute worst case motor travel time
                             # = U_MOTOR_FS_TRAVEL_MULT *
                             #   2 * F_<motor>_SAFETY_MAX / F_<motor>NOMINAL_VEL


sensor: f_motor_analyze_deadband(nodim) 1800.0 # enables computation and printing of
                               # all motor positioning stats, i.e. diffence
                               # between C_xxx(commanded) and M_xxx(measured)
                               # <= 0    no action, zero stats
                               # >  0    accumulate stats (min, mean, max, standev)
                               #         every F_MOTOR_ANALYZE_DEADBAND calls...
                               #           print and zero stats
```

ballast/buoyancy pump: motor.c motor_drivers.

sensor: c_ballast_pumped(cc) 0 #in >0 pumps ballast overboard, goes up

sensor: m_ballast_pumped(cc) #out,

sensor: f_ballast_pumped_stall_retry(sec) 10.0 # in, how long to wait for retry if

pump jams, not moving fast enuf

sensor: x_ballast_pumped_max(cc) 226 # out, Maximum OPERATIONAL limit

sensor: x_ballast_pumped_deadband(cc) 0.0 # out, how close is good enuf

= f_ballast_pumped_deadz_width * f_ballast_pumped_db_frac_dz

sensor: x_ballast_pumped_passive_retraction_depth(m) 200.0 # Maintains shallowest depth

where battery spike occured

sensor: x_ballast_passive_retraction_count(int) 0 # How many times we hit the brakes

sensor: f_ballast_passive_retraction_delay(ms) 10 # How long for

sensor: m_is_ballast_pump_moving(bool) 0 # out, t-> motor is moving

sensor: m_ballast_pumped_vel(cc/sec) 0 # out, measured motor speed

sensor: m_ballast_pumped_energy(joules) 0 #out, How much energy to pump water on last command

= pressure * volume when extending

sensor: m_tot_ballast_pumped_energy(kjoules) 0 #out, totalized m_ballast_pumped_energy

sensor: u_ballast_pumped_microposition(bool) 0 # T==> microposition the motor

sensor: u_ballast_pumped_micropos_rt(msec) 250 # "run time" >0 max allowable microposition time

```
sensor:   u_ballast_pumped_micropos_wp(nodim)  0.01 #"when pulse" 0-1  when start pulsing the
motor
                                        #  0 immediately, 0.5 when half way there, 1 never
sensor:   u_ballast_pumped_micropos_dc(nodim)  10   # "duty cycle" 1-N  once pulsing,
                                        #    pulse motor 1 cycle out of this many



 # max = safety_max - deadzone
 sensor: f_ballast_pumped_safety_max(cc) 268.0  # in, damage to glider
 sensor: f_ballast_pumped_deadz_width(cc)  42.0  # in, sets x_ limit
 sensor: f_ballast_pumped_db_frac_dz(nodim) 1.0   # deadband as fraction of dead zone
 sensor: f_ballast_pumped_nominal_vel(cc/sec) 132.0 # in, nominal speed
 sensor: f_ballast_pumped_reqd_vel_frac(nodim) 0.25 # in, fraction of nominal
                                        # required before saying not
                                        # moving fast enuf
 sensor:   u_ballast_pumped_stop_distance(cc)  0    # how long it takes pump to stop

# This battery voltage spike relative to m_battery triggers the driver to use
# passive retraction, disengaged brake with pump power off.
sensor: f_ballast_pumped_battery_spike_trigger(volts) 3.0



 # Specs linear relationship between sensor units (cc) and the
 # voltage we actually read out of the AD for position
 # pumped(cc) = pumped_cal_m(cc/Volt) * volts + pumped_cal_b(cc)
```

sensor: f_ballast_pumped_cal_m(cc/Volt) 366.93 # in, slope

sensor: f_ballast_pumped_cal_b(cc) -412.19 # in, y-intercept

Battery (fore/aft) position: motor.c motor_drivers.

sensor: c_battpos(in) 0 # in, >0 vehicle dives (nose down)

the battery is moved forward

sensor: c_dive_battpos(in) 0 # out, battpos to immediately move to at start of dive. Set by either DM_PITCH_SERVO or in diveclimb.c

sensor: c_climb_battpos(in) 0 # out, battpos to immediately move to at start of climb. Set by either DM_PITCH_SERVO or in diveclimb.c

sensor: c_hover_battpos(in) 0 # out, battpos to immediately move to at start of hover/drift_at_depth. Set by DM_PITCH_SERVO

sensor: u_battpos_avg_num_min(enum) 5 #in, pitch servo keeps track of a running avg of battpos that result in

pitch angle within db. This sensor defines the minimum number of samples

required to transition to x_battpos_achieved=2 and record the avg battpos

to c_climb/dive_battpos

sensor: m_battpos(in) # out

sensor: x_battpos_max(in) # out, Maximum OPERATIONAL limit

sensor: x_battpos_deadband(in) 0.0 # out, how close is good enuf

= f_battpos_deadzone_width * f_battpos_db_frac_dz

sensor: m_is_battpos_moving(bool) 0 # out, t-> motor is moving

sensor: m_battpos_vel(in/sec) 0 # out, measured motor velocity

```
sensor: u_battpos_microposition(bool)   1  # T==> microposition the motor
sensor:   u_battpos_micropos_rt(msec)   1000  # "run time"  >0 max allowable microposition time
sensor:   u_battpos_micropos_wp(nodim)  0.01 #"when pulse" 0-1  when start pulsing the motor
                                #  0 immediately, 0.5 when half way there, 1 never
sensor:   u_battpos_micropos_dc(nodim)  10   # "duty cycle" 1-N  once pulsing,
                                #    pulse motor 1 cycle out of this many
sensor:   u_battpos_stop_distance(in) 0      # stop distance



 # max = safety_max - deadzone
 # x_battpos_max = f_safety_max_battpos - f_deadzone_width_battpos


 sensor: f_battpos_safety_max(inches) 0.45  # in, damage to glider
 sensor: f_battpos_deadzone_width(inches) 0.068 # Sets x_ limit
 sensor: f_battpos_db_frac_dz(nodim)      1.0   # deadband as fraction of dead zone
 sensor: f_battpos_nominal_vel(inches/sec)  0.16 # nominal speed
 sensor: f_battpos_reqd_vel_frac(nodim) 0.25 # in, fraction of nominal
                                 # required before saying not
                                 # moving fast enuf
 sensor: u_battpos_ap_deadband(inches) 0.068 # deadband of battpos driver, used to immediately
                                                                                 # go to a
battpos in pitch servo mode.
 # Specs linear relationship between sensor units (inches) and the
 # voltage we actually read out of the AD for position
 # battpos(inches) = _cal_m(inches/Volt) * volts + _cal_b(inches)
 sensor: f_battpos_cal_m(inches/Volt)  0.571 # slope
```

```
 sensor: f_battpos_cal_b(inches)      -0.506 # y-intercept


 sensor: u_pitch_energy_cal_m(nodim)    1.0
 sensor: u_pitch_energy_cal_b(volts)    0.0
 sensor: m_pitch_energy(joules)         0.0


# fpitch_pump (fluid pumped fore/aft): fpitch_pump.c
sensor: c_fluid_pumped(cc)           0 # in, >0 vehicle dives (nose down)
                                     #     the fluid is pumped forward
sensor: m_fluid_pumped(cc)           # out
sensor: x_fluid_pumped_max(cc)       # out, Maximum OPERATIONAL limit
sensor: x_fluid_pumped_deadband(cc) 0.0 # out, how close is good enuf
                  # = f_fluid_pumped_deadzone_width * f_fluid_pumped_db_frac_dz
sensor: m_is_fpitch_pump_moving(bool) 0 # out, t-> pump is moving
sensor: m_fluid_pumped_vel(cc/sec)    0 # out, measured fluid pumped velocity
sensor: m_fluid_pumped_fwd_hall_voltage(volts) # out, voltage from forward hall sensor
sensor: m_fluid_pumped_aft_hall_voltage(volts) # out, voltage from aft hall sensor


# With all the fluid in the aft reservoir take the difference of:
# m_fpitch_pump_fwd_hall_voltage - m_fpitch_aft_hall_voltage
sensor: f_fluid_pumped_voltage_offset(volts) -1.27 # fully retracted (fwd_volts-aft_volts)


# max = safety_max - deadzone
# x_fluid_pumped_max = f_fluid_pumped_safety_max - f_fluid_pumped_deadzone_width
```

sensor: f_fluid_pumped_safety_max(cc) 160.0 # in, damage to glider

sensor: f_fluid_pumped_deadzone_width(cc) 20.0 # Sets x_ limit

sensor: f_fluid_pumped_db_frac_dz(nodim) 0.5 # deadband as fraction of dead zone

sensor: f_fluid_pumped_nominal_vel(cc/sec) 20.0 # nominal speed

sensor: f_fluid_pumped_reqd_vel_frac(nodim) 0.2 # in, fraction of nominal

required before saying not

moving fast enuf

#

Specs linear relationship between sensor units (cc) and the

voltage we actually read out of the AD for position

fluid_pumped(cc) = _cal_m(cc/Volt) * volts + _cal_b(cc)

sensor: f_fluid_pumped_cal_m(cc/Volt) 134.4 # slope

sensor: f_fluid_pumped_cal_b(cc) -168.0 # y-intercept

battery roll, motor.c motor_drivers.

battroll

sensor: c_battroll(rad) 0 # in, >0 puts stbd wing down

the battery is rotated ClockWise (CW)

when looking fwd

sensor: m_battroll(rad) # out

sensor: x_battroll_max(rad) # out, Maximum OPERATIONAL limit

sensor: x_battroll_deadband(rad) 0.0 # out, how close is good enuf

```
                # = f_battroll_deadzone_width * f_battroll_db_frac_dz
sensor: m_is_battroll_moving(bool) 0   # out, t-> motor is moving
sensor: m_battroll_vel(rad/sec)    0  # out, measured motor velocity

sensor: u_battroll_microposition(bool) 0  # T==> microposition the motor
sensor:   u_battroll_micropos_rt(msec)   250  # "run time"  >0 max allowable microposition time
sensor:   u_battroll_micropos_wp(nodim)  0.01 #"when pulse" 0-1  when start pulsing the motor
                        #  0 immediately, 0.5 when half way there, 1 never
sensor:   u_battroll_micropos_dc(nodim)  10   # "duty cycle" 1-N  once pulsing,
                        #    pulse motor 1 cycle out of this many


 # max = safety_max - deadzone
 sensor: f_battroll_safety_max(rad) 0.52  # in, damage to glider
 sensor: f_battroll_deadzone_width(rad) 0.088 # in, Sets x_ limit
 sensor: f_battroll_db_frac_dz(nodim)      1.0   # deadband as fraction of dead zone
 sensor: f_battroll_nominal_vel(rad/sec) 0.09 # in, nominal speed
 sensor: f_battroll_reqd_vel_frac(nodim)0.25 # in, fraction of nominal
                        # required before saying not
                        # moving fast enuf


 # Specs linear relationship between sensor units (rads) and the
 # voltage we actually read out of the AD for position
 # battroll(rad) = _cal_m(rad/Volt) * volts + battroll_cal_b(rad)
```

```
 sensor: f_battroll_cal_m(rad/Volt)  0.950 # slope
 sensor: f_battroll_cal_b(rad)       -1.22 # y-intercept


# fin, motor.c motor_drivers
# These moved (in this file) to digifin_v2: c_fin, m_fin,
sensor: f_fin_offset(rad) 0.0 # in, added to c_fin to trim (after autopilot)

sensor: x_fin_max(rad) # out, Maximum OPERATIONAL limit
sensor: x_fin_deadband(rad)  0.0  # out, how close is good enuf
                # = f_fin_deadzone_width * f_fin_db_frac_dz
sensor: m_is_fin_moving(bool) 0   # out, t-> motor is moving
sensor: m_fin_vel(rad/sec)    0  # out, measured motor velocity

sensor: u_fin_microposition(bool) 1  # T==> microposition the motor
sensor:   u_fin_micropos_rt(msec)   750  # "run time"  >0 max allowable microposition time
sensor:   u_fin_micropos_wp(nodim)  0.01 #"when pulse" 0-1  when start pulsing the motor
                        #  0 immediately, 0.5 when half way there, 1 never
sensor:   u_fin_micropos_dc(nodim)  5    # "duty cycle" 1-N  once pulsing,
                        #    pulse motor 1 cycle out of this many

################################################
# start of readbacks which apply only to
# Lithium Ion Power Driver
################################################
```

debugging control - only effective for Lithium Ion Power Driver (LIPD)

sensor: u_lithium_battery_debug(nodim) 0 # Bit-mapped debug control register - add desired elements together

b0 1 real time trace all rcvd packets

b1 2 real time trace all gliderbus_transact errors

b2 4 real time trace all lipd_do_transaction errors

b3 8 fake a good return from gliderbus_transact errors

b4 16 fake a good return for missing "$" beginning of packet

b5 32 fake a good return for packet parse errors

b6 64 fake a good return for ill-formed checksums

b7 128 fake a good return for checksum mismatches

b8 256 print a trace of faked good returns

b9 512 make response packet timeout only a device oddity

b10 1024 make all non-hopeless device errors into warnings

b11 2048 make all non-hopeless device errors and warnings into oddities

b12 4096 make all non-hopeless device errors, warnings, and oddities into non-entities

b13 8192 never turn off gliderbus power

b14 16384 print number of phases attempted during lipd_ctrl execution

b15 32768 print duration of lipd_ctrl execution

b16 65536 unassigned

b17 131072 unassigned

b18 262144 unassigned

b19 524288 unassigned

b20 1048576 unassigned

b21 2097152 unassigned

b22 4194304 unassigned

b23 8388608 unassigned

b24 16777216 unassigned

b25 33554432 unassigned

b26 67108864 unassigned

b27 134217728 unassigned

b28 268435456 unassigned

b29 536870912 unassigned

b30 1073741824 unassigned

b31 2147483648 unassigned

sensor: c_lithium_battery_on(sec) 0 # required by gb_devdrvr paradigm

statistics

sensor: m_lithium_battery_relative_charge(%) 0 # Relative cumulative charge

sensor: m_lithium_battery_time_to_discharge(mins) 0 # cumulative time to discharge

sensor: m_lithium_battery_time_to_charge(mins) 0 # cumulative time to charge

sensor: m_lithium_battery_status(nodim) 0 # cumulative LIPD status

##

end of readbacks which apply only to LIPD

(Lithium Ion Power Driver)

##

##

start of readbacks which apply only to digifin

These moved (in this file) to digifin_v2: m_digifin_rawposition(nodim)

##

status

sensor: m_digifin_status(nodim) 0 # bit mapped status

sensor: m_digifin_motorstep_counter(nodim) 0 # total count of steps moved

primary data

sensor m_digifin ---------- # share use win fin_motor driver

sensor: m_digifin_rawposition---------- # share use with digifin_v2 driver

statistics

sensor: m_digifin_boot_counter(nodim) 0 # number of PIC power cycles

sensor: m_digifin_uptime_secs(nodim) 0 # number of seconds into PIC power cycle

sensor: m_digifin_uptime_secs_delta(nodim) 0 # delta of number of seconds into PIC power cycle

sensor: m_digifin_num_pkts_rcvd_by_pic_total(nodim) 0 # total number of packets received by PIC (per power cycle)

sensor: m_digifin_num_pkts_rcvd_by_pic_total_delta(nodim) 0 # delta of total number of packets received by PIC

sensor: m_digifin_num_pkts_rcvd_by_pic_with_error(nodim) 0 # number of packets received with error by PIC (per power cycle)

sensor: m_digifin_num_pkts_rcvd_by_pic_with_error_delta(nodim) 0 # delta of number of packets received with error by PIC

sensor: m_digifin_num_pkts_rcvd_by_pic_good(nodim) 0 # number of packets received good by PIC (per power cycle)

sensor: m_digifin_num_pkts_rcvd_by_pic_good_delta(nodim) 0 # delta of number of packets received good by PIC

sensor: m_digifin_motorstep_counter(nodim) 0 # energy consumption metric (per power cycle)

sensor: m_digifin_motorstep_counter_delta(nodim) 0 # delta of energy consumption metric

sensor: m_digifin_recapture_counter(nodim) 0 # number of times recapture performed (per power cycle)

sensor: m_digifin_recapture_counter_delta(nodim) 0 # delta of number of times recapture performed

sensor: m_digifin_factorycal_counter(nodim) 0 # number of times factory cal performed (per power cycle)

sensor: m_digifin_factorycal_counter_delta(nodim) 0 # delta of number of times factory cal performed

sensor: m_digifin_startupcal_counter(nodim) 0 # number of times startup cal performed (per power cycle)

sensor: m_digifin_startupcal_counter_delta(nodim) 0 # delta of number of times startup cal performed

sensor: m_digifin_demandcal_counter(nodim) 0 # number of times demand cal performed (per power cycle)

sensor: m_digifin_demandcal_counter_delta(nodim) 0 # delta of number of times demand cal performed

sensor: m_digifin_activecal_counter(nodim) 0 # number of times active cal set (per power cycle)

sensor: m_digifin_activecal_counter_delta(nodim) 0 # delta of number of times active cal set

sensor: m_digifin_leakdetect_counter(nodim) 0 # number of times leak detected (per power cycle)

sensor: m_digifin_leakdetect_counter_delta(nodim) 0 # delta of number of times leak detected

sensor: m_digifin_motorfault_counter(nodim) 0 # number of times motor fault registered (per power cycle)

sensor: m_digifin_motorfault_counter_delta(nodim) 0 # delta of number of times motor fault registered

sensor: m_digifin_phases_attempted(nodim) 0 # number of I/O phases attempted by digifin_ctrl

sensor: m_digifin_phases_failed(nodim) 0 # number of I/O phases failed by digifin_ctrl

sensor: m_digifin_phases_busy(nodim) 0 # number of I/O phases by digifin_ctrl with busy response

sensor: m_digifin_phases_good(nodim) 0 # number of I/O phases good by digifin_ctrl

calibration

sensor: m_digifin_calbehavior(nodim) 0 # 0 -> old behavior, 1 -> new behavior

sensor: m_digifin_factorycal_portstop(nodim) 0 # port stop for factory cal in A/D counts

sensor: m_digifin_factorycal_stbdstop(nodim) 0 # stbd stop for factory cal in A/D counts

sensor: m_digifin_startupcal_portstop(nodim) 0 # port stop for startup cal in A/D counts

sensor: m_digifin_startupcal_stbdstop(nodim) 0 # stbd stop for startup cal in A/D counts

sensor: m_digifin_demandcal_portstop(nodim) 0 # port stop for demand cal in A/D counts

sensor: m_digifin_demandcal_stbdstop(nodim) 0 # stbd stop for demand cal in A/D counts

sensor: m_digifin_activecal_midpoint(nodim) 0 # active cal midpoint in A/D counts

sensor: m_digifin_activecal_type(nodim) 0 # last cal type: 0-1023 -> set to specified value

65535 -> set from last factory cal

65534 -> set from last startup cal

65533 -> set from last demand cal

65532 -> set from current position

leak detect

sensor: m_digifin_leakdetect_reading(nodim) 0 # leak detect reading in A/D counts

sensor: m_digifin_leakdetect_threshold(nodim) 0 # leak detect threshold in A/D counts

firmware properties

sensor: m_digifin_bootloader_version(nodim) 0 # bootloader version

sensor: m_digifin_firmware_version(nodim) 0 # firmware version

sensor: m_digifin_firmware_stored_checksum(nodim) 0 # firmware checksum stored in flash

sensor: m_digifin_firmware_calculated_checksum(nodim) 0 # firmware checksum calculated at boot time

debugging

sensor: m_digifin_pic_debug(nodim) 0 # general-use debug register

sensor: f_digifin_movement_retry_max(nodim) 3 # Number of times digifin will attempt to

retry to move the fin to a commanded position

before it issues a warning (-1 = infinite retry, never give warning).

##

end of readbacks which apply only to digifin

##

basically fixed parameters for digifin

sensor: u_digifin_response_timeout(msec) -1 # timeout from issuing command to receiving response from digifin

(-1 means using gliderbus default value)

sensor: f_digifin_busy_timeout_secs(sec) 45 # max # of secs the digifin can respond to our cmds

with "busy" responses before we generate an error

(note: this is in one continuous run of all "busy"

response packets with no non-"busy" response packets)

sensor: f_digifin_startup_wait(sec) 10 # number of secs after turning digifin power on

until we start talking to it

sensor: f_digifin_status_mask(nodim) 524031 # every bit 0-18 set except b8 = 256 (leakdetect rdg chgd)

sensor: f_digifin_status_force(nodim) 2 # only bit set b1 = 2 (position changed)

mechanism for issuing special commands to digifin

sensor: c_digifin_write_reg(nodim) 0 # in; digifin register to write to

sensor: c_digifin_read_reg(nodim) 0 # in, digifin register to read from

sensor: c_digifin_cmd_data(nodim) 0 # in; data for digifin command

sensor: m_digifin_resp_data(nodim) 0 # out; data from digifin response

sensor: m_digifin_cmd_done(nodim) 0 # in/out; flag for command completed; T ==> completed

sensor: m_digifin_cmd_error(nodim) 0 # out; T ==> error running special command

debugging control - only effective for digifin

sensor: u_digifin_debug(nodim) 0 # Bit-mapped debug control register - add desired elements together

b0 1 real time trace all rcvd packets

b1 2 real time trace all gliderbus_transact errors

b2 4 real time trace all digifin_do_transaction errors

b3 8 fake a good return from gliderbus_transact errors

b4 16 fake a good return for missing "$" beginning of packet

b5 32 fake a good return for packet parse errors

b6 64 fake a good return for ill-formed checksums

b7 128 fake a good return for checksum mismatches

b8 256 print a trace of faked good returns

b9 512 make response packet timeout only a device oddity

b10 1024 make all non-hopeless device errors into warnings

b11 2048 make all non-hopeless device errors and warnings into oddities

b12 4096 make all non-hopeless device errors, warnings, and oddities into non-entities

b13 8192 never turn off gliderbus power

b14 16384 print number of phases attempted during digifin_ctrl execution

b15 32768 print duration of digifin_ctrl execution

b16 65536 unassigned

b17 131072 unassigned

b18 262144 unassigned

b19 524288 unassigned

b20 1048576 unassigned

b21 2097152 unassigned

b22 4194304 unassigned

b23 8388608 unassigned

b24 16777216 unassigned

b25 33554432 unassigned

b26 67108864 unassigned

b27 134217728 unassigned

b28 268435456 unassigned

b29 536870912 unassigned

b30 1073741824 unassigned

b31 2147483648 unassigned

debugging values - only apply to digifin

reset these by setting c_fin_debug_reset to true (this is edge detected and automatically set back to false)

```
  # max = safety_max - deadzone

  # These moved (in this file) to digifin_v2: f_fin_safety_max

  sensor: f_fin_deadzone_width(rad) 0.020 # in, Sets x_ limit (motor_fin and digifin_v2)

  sensor: f_fin_db_frac_dz(nodim)      1.0   # deadband as fraction of dead zone (motor_fin and
digifin_v2)

  sensor: f_fin_nominal_vel(rad/sec) 0.0981 # in, nominal speed

  sensor: f_fin_reqd_vel_frac(nodim) 0.25   # in, fraction of nominal

                                  # required before saying not

                                  # moving fast enuf



  # Specs linear relationship between sensor units (rads) and the

  # voltage we actually read out of the AD for position

  # fin(rad) = _cal_m(rad/Volt) * volts + fin_cal_b(rad)

  sensor: f_fin_cal_m(rad/Volt)  0.6461 # slope

  sensor: f_fin_cal_b(rad)       -.7904 # y-intercept



# de_pump.c


# Inputs:


sensor: c_de_oil_vol(cc)                    260.0  # >0, goes up


sensor: u_min_de_oil_flux(cc/sec)               0.10 # if below, error
```

sensor: u_de_oil_vol_check_time(sec) 0.0 # monitoring rate while stable

0 = every cycle

sensor: u_secs_for_oil_vol_stabilization(secs) 30.0 # <=0 disables, wait time for any gas in system

to stabilize after ascents

sensor: u_de_avg_oil_vol_err_alpha(nodim) 0.0 # 0 - 0.05 (0.05 = more weight to long term average)

set these to illegal values to insure them getting set in autoexec.mi

sensor: f_de_oil_vol_pot_voltage_min(volts) -20.0 # raw AD voltage of fully retracted pot

sensor: f_de_oil_vol_pot_voltage_max(volts) -20.0 # raw AD voltage of fully extended pot

sensor: f_de_oil_vol_in_system(cc) 650.0 # volume of internal oil reservoir

sensor: f_de_oil_vol_safety_max(cc) 290.0 # shouldn't go beyond this

sensor: f_de_oil_vol_deadz_width(cc) 30.0 # sets x_ limit

sensor: f_de_oil_vol_db_frac_dz(nodim) 0.667 # deadband as fraction of dead zone

sensor: f_de_max_secs_for_updown_to_finish(secs) 540.0 # 9 minutes (~ how

long it takes

to retract 650cc

of oil at surface)

sensor: x_de_pump_disable(bool) 0 # t-> disable the de_pump driver,

needed to run GliderDos tvalve command

Outputs:

sensor: m_de_oil_vol(cc) 0.0 # calibrated from m_de_oil_vol_pot_voltage

sensor: m_de_oil_vol_pot_voltage(volts) 0.0 # raw voltage from AD

```
sensor: m_is_de_pump_moving(bool)          0   # t-> motor is moving

sensor: m_de_pump_fault_count(nodim)       0   # incremented when bit_BPUMP_FAULT
                                               # is set on start-up and gets cleared
                                               # after a successful re-start or
                                               # after an abort (3 tries).


sensor: x_de_oil_vol_deadband(cc)          0.0 # how close is good enough
                                               # = f_de_oil_vol_deadz_width *
                                               #   f_de_oil_vol_db_frac_dz
# max = safety_max - deadz_width
sensor: x_de_oil_vol_max(cc)               0.0 # Maximum OPERATIONAL limit


# Needed to adjust voltage limits in de_pump_chore to account for pump
# and valve power off time latencies, and gas in the system
sensor: x_de_oil_vol_ierr_on_ascent(cc)    0.0 # sum(measured - commanded)
sensor: x_de_oil_vol_ierr_on_descent(cc)   0.0 # sum(measured - commanded)
sensor: x_de_avg_oil_vol_ierr_on_ascent(cc)  0.0 # avg(sum(measured - commanded))
sensor: x_de_avg_oil_vol_ierr_on_descent(cc) 0.0 # avg(sum(measured - commanded))


# Keeps tract of the oil flux in the deep electric
sensor: x_de_oil_flux(cc/sec)              0.0 # positive = pumping, negative = retracting


sensor: x_de_ignore_tvalve_oddity(bool)    0   # t-> don't log tvalve oddity after
                                               # de_pump chore
```

threng.c

sensor: c_thermal_updown(enum) 0.0 # in

CTHRENG_DONT_USE(-1) Disable this driver (thrvalve still active)

CTHRENG_UP_CHARGE(0) Go thru an UP, CHARGE cycle

CTHRENG_DOWN(1) DOWN

sensor: m_thermal_updown(enum) 3.0 # out

MTHRENG_CHARGE(0) Stable in the charge position

MTHRENG_DOWN(1) Stable In the down position

MTHRENG_MOVING(2) Moving between states

MTHRENG_NOT_IN_USE(3) Higher level driver disabled

MTHRENG_ERROR(-1) Something bad happened, someone should abort

sensor: u_thermal_valve_time_in_up_pos(s) 60.0 # in, how long thermal valve says in up position

before being automatically moved to charge

sensor: u_thermal_valve_time_in_down_pos(s) 300 # in, 5 minutes in seconds

how long the valve must be in the down

position before allowed to go to up position

Used to prevent "double charges". Ignored

for safety sake if glider is deeper than the

minimum of f_max_working_depth or f_at_risk_depth

thrvalve.c

sensor: f_thermal_valve_time_over_slot(msec) 150 # millisecs the thermal valve hole is over the sensor

sensor: c_thermal_valve(enum) # in, THRVALVE_UP(1),THRVALVE_CHARGE(2), THRVALVE_DOWN(3)

sensor: m_thermal_valve(enum) # out, THRVALVE_UNKNOWN(0), THRVALVE_UP(1),
THRVALVE_MOVING_TO_UP(-1)

THRVALVE_CHARGE(2), THRVALVE_MOVING_TO_CHARGE(-2),

THRVALVE_DOWN(3), THRVALVE_MOVING_TO_DOWN(-3)

sensor: m_is_thermal_valve_moving(bool) # out, true if valve is moving

sensor: x_thermal_valve_move_backwards(bool) 0 # In, non-zero means move valve backwards

DO NOT MANUALLY set this, it is maintained

by gliderdos TVALVE command. Only used in -lab.

sensor: u_thermal_valve_check_time(sec) 180 # how often check valve position

<= 0 to disable

tcm3.c

sensor: f_tcm3_cal_points(nodim) 50 # Default number of sample points in calibration

sensor: m_tcm3_stddeverr(uT) -1 # The compass samples magnetic field

standard deviation error.

sensor: m_tcm3_xcoverage(%) -1 # Percentage of how much of the X magnetometer

axis was covered by the sampling.

sensor: m_tcm3_ycoverage(%) -1 # Percentage of how much of the Y magnetometer

axis was covered by the sampling.

sensor: m_tcm3_zcoverage(%) -1 # Percentage of how much of the Z magnetometer

axis was covered by the sampling.

sensor: m_tcm3_magbearth(uT) -1 # The calculated Earth's magnetic field

magnitude from the calibration samples.

sensor: m_tcm3_is_calibrated(bool) 0 # The compass calibration status flag.

sensor: m_tcm3_poll_time(ms) 0 # Time after open_uart() call we poll for data

sensor: m_tcm3_recv_start_time(ms) 0 # Time after open_uart() call we start receiving data

sensor: m_tcm3_recv_stop_time(ms) 0 # Time after open_uart() call we stop receiving data

attitude.c/attitude_tcm3.c/attitude_rev.c

sensor: c_att_time(sec) 0 # in, time spacing for attitude checks

<0 is off, =0 as fast as possible

otherwise secs between measurements

sensor: c_att_recall(msec) -1.0 # in, <=0 no subcycle measurements

>0 millisecs between subcycle measurements

(c_att_time must be 0 to enable)

sensor: u_att_rev_ignore_warnings(bool) 1 # Only on the Revolution, ignore warnings by default.

sensor: m_roll(rad) 0 # out, >0 is port wing up

sensor: m_pitch(rad) 0 # out, >0 is nose up

sensor: m_heading(rad) 0 # out

sensor: m_vehicle_temp(degC) 0 # out

oceanpres.c

sensor: c_pressure_time(sec) 1 # in, <0 is off, =0 as fast as possible

>0 num seconds betweens measurements

sensor: c_pressure_recall(msec) -1 # in, <=0 no subcycle measurements

>0 millisecs between subcycle measurements

c_pressure_time must be 0 to enable

sensor: m_pressure_voltage(volts) # out, measured, averaged or median filtered from 20 raw samples of AD

sensor: m_pressure(bar) # out, measured NOT clipped:

<0 not good, glider above the surface,

0 surface

>0 glider below surface in water

sensor: m_depth(m) 0 # out, calculated clips at 0

0 surface

>0 glider below surface in water

sensor: u_use_ctd_depth_for_flying(bool) 0 #! visible = True

true=> use ctd measurement for m_depth

implemented as emergency workaround for

broken ocean pressure

sensor: m_depth_rejected(bool) 0 # out, true if depth measurement is filtered

1 ==> thinks glider at surface

U_DEPTH_RATE_FILTER_SUB_SUR_DEP ==> M_DEPTH

2 ==> thinks glider is NOT surface

no M_DEPTH is output

sensor: u_depth_rate_filter_sub_sur_dep(m) 0.05 # used for M_DEPTH when m_pressure rejected at

the surface

sensor: u_depth_rate_filter_factor(nodim) 4.0 # <=0 disables bad depth filter,

otherwise multiplies

f_nominal_dive_rate by this

value to create the cutoff value

for an acceptable depth

rate of change

sensor: x_measured_depth(m) 0.0 # The last published M_DEPTH where M_DEPTH_REJECTED is 0

i.e. actually came from pressure sensor in lieu of a

made up value at the surface. Depth rate filter compares

current "depth" being evaluated against this

sensor: u_pressure_autocal_min_time_between(secs) 180 # minimum interval time

between auto calibrations

sensor: u_pressure_autocal_enabled(bool) 1 # 0=turned off, 1=turned on

sensor: u_pressure_autocal_deadband(bar) 0.025 # re-calibrate when drift is

beyond + or - this

sensor: u_pressure_autocal_max_allowed(bar) 0.2 # print oddity when drift is

beyond + or - this, don't

re-calibrate

sensor: u_pressure_autocal_performed(bool) 0 # becomes 1 when auto re-calibration is done

becomes 2 when manual re-calibration is done

becomes -1 when excessive pressure drift is detect:

(no calibration is done!)

sensor: x_pressure_manual_cal_now(bool) 0 # non-zero causes manual (non-auto) re-calibration

set to 1 by GliderDos>zero_pressure_sensor

set to 0 by ocean_pressure device driver when

manual re-calibration is done

inputs, config stuff FS-->full scale

sensor: u_bar_per_meter(bar/m) 0.1 # Converts m_pressure to m_depth

sensor: f_ocean_pressure_full_scale(bar) 13.8 # pressure @ FS volts

sensor: f_ocean_pressure_min(volts) 0.20 # voltage for 0 pressure

sensor: f_ocean_pressure_max(volts) 2.40 # voltage for FS pressure

sensor: u_pressure_median(bool) 0 # T ==> perform median filtering (new behavor),

F ==> perform averaging only (old behavior)

sensor: u_pressure_median_k(nodim) 1 # standard deviation mutiplier for median filtering

sensor: u_pressure_median_iter(nodim) 1 # number of iterations for median filtering

(minimum for this sensor is clipped at 1)

sensor: u_pressure_median_median(bool) 0 # T ==> after median filtering, use median of remaining samples for pressure measurement

F ==> after median filtering, use mean of remaining samples for pressure measurement

sensor: u_pressure_median_debug(enum) 0 # bit-mapped debug control (values are additive):

0 = no debug functionality

1 = (b0) debug trace for statistics

2 = (b1) debug trace for oops

4 = (b2) record raw samples

8 = (b3) debug trace for timing

raw samples for debugging the averaging/median filtering code

(only if enabled by u_pressure_median_debug)

sensor: m_pressure_raw_voltage_sample0(volts) # first raw AD sample

sensor: m_pressure_raw_voltage_sample1(volts) # second raw AD sample

sensor: m_pressure_raw_voltage_sample2(volts) # third raw AD sample

sensor: m_pressure_raw_voltage_sample3(volts) # fourth raw AD sample

sensor: m_pressure_raw_voltage_sample4(volts) # fifth raw AD sample

sensor: m_pressure_raw_voltage_sample5(volts) # sixth raw AD sample

sensor: m_pressure_raw_voltage_sample6(volts) # seventh raw AD sample

sensor: m_pressure_raw_voltage_sample7(volts) # eighth raw AD sample

sensor: m_pressure_raw_voltage_sample8(volts) # ninth raw AD sample

sensor: m_pressure_raw_voltage_sample9(volts) # tenth raw AD sample

sensor: m_pressure_raw_voltage_sample10(volts) # eleventh raw AD sample

sensor: m_pressure_raw_voltage_sample11(volts) # twelfth raw AD sample

sensor: m_pressure_raw_voltage_sample12(volts) # thirteenth raw AD sample

sensor: m_pressure_raw_voltage_sample13(volts) # fourteenth raw AD sample

sensor: m_pressure_raw_voltage_sample14(volts) # fifteenth raw AD sample

sensor: m_pressure_raw_voltage_sample15(volts) # sixteenth raw AD sample

sensor: m_pressure_raw_voltage_sample16(volts) # seventeenth raw AD sample

sensor: m_pressure_raw_voltage_sample17(volts) # eighteenth raw AD sample

sensor: m_pressure_raw_voltage_sample18(volts) # nineteenth raw AD sample

sensor: m_pressure_raw_voltage_sample19(volts) # twentieth raw AD sample

engpres.c (driver name: thermal_acc_pres)

sensor: c_thermal_acc_pres_time(sec) 1 # in, <0 is off, =0 as fast as possible

>0 num seconds between measurements

sensor: c_thermal_acc_pres_recall(msec) -1.0 # in, <=0 no subcycle measurements

```
                                # >0 millisecs between subcycle measurements
                                # c_thermal_acc_pres_time must be 0 to enable
sensor: m_thermal_acc_pres_voltage(volts) 0 # out, raw voltage from AD
sensor: m_thermal_acc_pres(bar)          0 # out, calibrated from m_thermal_acc_pres_voltage


  # inputs, volts/pressure config stuff  FS-->full scale
  sensor: f_thermal_acc_pres_full_scale(bar) 220.0 # pressure @ FS volts
  sensor: f_thermal_acc_pres_min(volts) 0.160 # voltage for 0 pressure
  sensor: f_thermal_acc_pres_max(volts) 1.767 # voltage for FS pressure

sensor: m_thermal_acc_vol(cc)       # out, computed oil volume from m_thermal_acc_pres
  # inputs, volume/pressure config stuff
  #      specs PV=k relationship between pressure and volume
  #      m_thermal_acc_vol(cc) = f_thermal_acc_vol_cal_v0(cc) *
  #                        (1 - f_thermal_acc_vol_cal_p0(bar)/m_thermal_acc_pres(bar))
  sensor: f_thermal_acc_vol_cal_v0(cc)  1340.0 # in, invariant volume with piston full out
                                # 800cc from ext tank
                                # 540cc accumulator (25 in^3)
  sensor: f_thermal_acc_vol_cal_p0(bar) 137.8948 # in, initial pressure with piston full out
                                 # 2000psi=>137.8948

sensor: m_thermal_enuf_acc_vol(bool)  0  # out, reflects state of switch that measure
                                #      adequate thermal displacement.
                                # 0==> not enuf    !=0 ==> enuf
```

```
                                #
sensor: f_thermal_reqd_acc_pres(bar) 200.0 # in, threshold pressure for thermal charge
                                # minimum reqd value = 186.0 (as of 2010.01.14)
sensor: x_thermal_reqd_acc_vol(cc)   416.1048 # out, the volume of oil in accumulator when
                                #      switch says we have enuf


 # thrpump.c
sensor: c_thermal_pump(enum)  0   # in, commanded state of thermal pump:
                        #    CTHRPUMP_OFF            0.0
                        #    CTHRPUMP_ON_WITH_CHECKS   1.0
                        #    CTHRPUMP_ON_REGARDLESS    2.0  note: only in -lab
sensor: m_thermal_pump(enum)  0   # out, actual state of thermal pump:
                        #    MTHR_PUMP_OFF               0.0
                        #    MTHR_PUMP_ON                1.0
                        #    MTHR_AWAITING_NOT_ENUF_VOLUME   -1.0
                        #    MTHR_AWAITING_REQD_PITCH       -2.0
                        #    MTHR_AWAITING_VALVE           -3.0
sensor: u_thermal_pump_reqd_pitch(rad) -0.1745  # in,   how far down glider must be pointing in
                                #      to use the pump   (-0.1745rad => -10deg)
sensor: x_thermal_pump_start_in(sec)   -1.0     # in/out, advisory time until thermal pump is engaged


# altimeter.c
sensor: c_alt_time(sec) 0 # in, time spacing for altimeter pings
                # <0 is off, =0 as fast as possible
                # >0 that many seconds betweens measurements
```

sensor: c_alt_recall(msecs) -1.0 # in, <=0 no subcycle sampling

>0 millisecs between subcycle measurements

c_alt_time must be 0 to enable

sensor: f_altimeter_model(enum) 0 # in, which altimeter is installed:

0 Benthos, sample 400ms after power on

1 AirMar(mod1), sample 3.2 to 5 sec after power on

-1 experimental, sample u_exp_alt_pwr_stb_time secs

after power on

sensor: u_exp_alt_pwr_stb_time(s) 0 # in, only looked at if f_altimeter_model == -1

control when to sample experimental altimeter

>0 the seconds to wait before reading altimeter

0 Never power off the altimeter, i.e. leave

it powered on all the time.

sensor: u_exp_alt_correction(m) 0 # in, only looked at if f_altimeter_model == -1 (experimental)

used to compensate for fixed offsets in altimeters

M_RAW_ALTITUDE(m) = M_RAW_ALTITUDE(m) + U_EXP_ALT_CORRECTION(m)

sensor: u_sound_speed(m/s) 1500.0 # User may tune this nominal value for sound speed in seawater.

Altimeters are calibrated assuming a 1500 m/s speed of sound.

Tuning this value will scale the output by (1500.0/u_sound_speed).

sensor: u_alt_min_post_inflection_time(sec) 10.0 # num secs after inflection before we take data

sensor: u_alt_min_depth(m) 2.0 #! visible = True

how deep vehicle must be to use altitude

sensor: u_alt_reqd_good_in_a_row(nodim) 3 # how many in a row we require before accepting reading

sensor: u_alt_filter_enabled(bool) 1 #! visible = True

enable median filter depth for altitude.

sensor: m_raw_altitude(m) # out, height above bottom, unfiltered

sensor: m_raw_altitude_rejected(bool) # out, true if altimeter did not supply reading

sensor: m_altimeter_voltage(volts) # out, voltage read from the A/D

watchdog.c

sensor: c_weight_drop(bool) 0 # in, non-zero->drop the weight

sensor: u_tickle_on_gps(bool) 1 # in, non-zero reset watchdog on every gps fix

sensor: u_tickle_on_console_cd(bool) 1 # in, non-zero reset watchdog if have freewave

sensor: x_hardware_cop_timeout(hours) -1 # out, reflects state of jumper

-1 can't tell, >=RevE will be 2 or 16

sensor: m_cop_tickle(bool) 1 # out, set to 1 whenever COP is tickled

sensor: m_tot_on_time(days) 0 # out, How long we have been powered on

sensor: m_bpump_fault_bit(bool) 0 # out, reflects state of bit_BPUMP_FAULT

airpump.c

sensor: c_air_pump(enum) 0 # in, <0 turns it off regardless

0 turns it off unless thermal or deep electric engine needs it

>0 turns it on

```
sensor: u_thermal_min_time_in_esc_pos(s) 1800.0 # in, for thermal only
                        # the number of seconds the air pump solenoid must
                        # stay in escape position before it is automatically
                        # returned to fill position.  Note: The glider must also
                        # NOT be at the surface for the valve to be automatically
                        # moved to fill position for thermal or electric.

sensor: m_air_pump(bool) 0   # out, whether it is on or not
sensor: m_air_fill(bool) 0   # out, T->air pump solenoid in fill position
                        #      F->air pump solenoid in escape position



# battery.c
sensor: u_battery_time(sec) 0 # in, Time between battery measurements
                         # <0 is off, =0 as fast as possible
                         # >0 num seconds betweens measurements
sensor: u_battery_recall(msecs) -1.0 # <=0 no subcycle measurements
                             # >0 millisecs between subcycle measurements
                             # u_battery_time must be 0 to enable
sensor: m_battery_inst(volts) 12   # out, Instantaneous battery voltage
sensor: m_battery(volts)      12   # out, Average Battery voltage
sensor: u_battery_alpha(nodim) 0.1 # in, The weighting factor to produce the average.
                             #     Should be between 0 and 1.
                             #     1 ==> no averaging at all
```

```
                #     smaller numbers mean more averaging
                #M_BATTERY =    U_BATTERY_ALPHA  * M_BATTERY_INST +
                #           (1-U_BATTERY_ALPHA) * M_BATTERY
```

```
# vacuum.c
sensor: u_vacuum_time(sec) 0 # in, Time between vacuum measurements
                # <0 is off, =0 as fast as possible
                # >0 that many seconds betweens measurements
sensor: x_increase_vacuum_time(bool) 0 # Whether or not to temporarily increase the vacuum sampling frequency to u_increase_vacuum_time
sensor: u_increase_vacuum_time(sec) 0 # If we want to temporarily increase the vacuum sampling frequency, then set it to this value
sensor: u_vacuum_recall(msec) -1 # in, <=0 no subcycle measurements
                # >0 millisecs between subcycle measurements
                # u_vacuum_time must be 0 to enable
sensor: m_vacuum(inHg) # out, Internal glider pressure
sensor:    u_vacuum_cal_m(inHg/Volt) -14.4059  # Factory Calibration data
sensor:    u_vacuum_cal_b(inHg)      31.64615  #      inHg = m V + b
```

```
# leakdetect.c
sensor: c_leakdetect_time(s) 0.0 # in, Time between leakdetect measurements
                #   <0 is off, =0 as fast as possible
                #   >0 that many seconds betweens measurements
sensor: c_leakdetect_recall(msec) -1.0 # in, <=0, no subcycle measurements
```

>0 millisecs between subcycle measurements

c_leakdetect_time must be 0 to enable

sensor: f_leakdetect_threshold(volts) 2.0 # in, Any M_LEAKDETECT_VOLTAGE below this is considered

a leak. This threshold is for both aft leakdetect and

forward leakdetect (if exists)

sensor: m_leakdetect_voltage(volts) 0.0 # out Voltage that was read out of the aft leak detect

The lower the voltage, the worse the leak.

sensor: m_leak(bool) 0.0 # non-zero ==> m_leakdetect_voltage_aft < f_leakdetect_threshold

sensor: m_leakdetect_voltage_forward(volts) 0.0 # out Voltage that was read out of the forward leak detect

The lower the voltage, the worse the leak.

sensor: m_leak_forward(bool) 0.0 # non-zero ==> m_leakdetect_voltage_forward < f_leakdetect_threshold

veh_temp.c

sensor: c_veh_temp_time(s) 0.0 # in, Time between vehicle temperature measurements

<0 is off, =0 as fast as possible

>0 that many seconds betweens measurements

sensor: c_veh_temp_recall(msec) 0.0 # in, <=0, no subcycle measurements

>0 millisecs between subcycle measurements

c_leakdetect_time must be 0 to enable

sensor: f_veh_temp_threshold(c) 38.0 # in, Any M_VEH_TEMP at or above this is considered

```
                        # an overheat.

sensor: m_veh_temp(c) -1.0          # out temperature that was read out from the board

sensor: m_veh_overheat(bool) -1.0   # non-zero ==> m_veh_temp >= f_veh_temp_threshold



# pinger.c
sensor: u_pinger_rep_rate(sec) 0    #in, secs between primary depth pings
                        # 0 turns it off

sensor: u_pinger_max_depth(m)  0    #in, Secondary ping at 1 second when m_depth
                        #    is >= this depth. (assuming nominal
                        #    8 second u_pinger_rep_rate)

sensor: u_ping_n_enabled(bool) 0    # if non-zero enable "ping N times"
                        # functionality, 0 turns it off for
                        # "quiet missions"

sensor: c_pinger_on(bool)      0    # in, non-zero means ping N times once



# gps.c
sensor: c_gps_on(enum) 0 # in, <0-> off always 0->off, but surface autoon, 1->gps take fixes
                # >1 take fixes + diag output [see gps.h]
```

sensor: u_gps_reqd_valid_fixes(nodim) 6 # in, reqd number of valid fixes since power on

before we publish as m_gps_lat/lon

sensor: m_gps_on(bool) 0 # out, >0 means gps is actually turned on

sensor: m_gps_lat(lat) 69696969 # out DDMM.MMMM >0 ==> North <0 ==> South

sensor: m_gps_lon(lon) 69696969 # out DDMM.MMMM >0 ==> East <0 ==> West

sensor: m_gps_x_lmc(m) 0 # out position in local mission coordinates

sensor: m_gps_y_lmc(m) 0 # out

sensor: m_gps_status(enum) 69 # out, updated with status of gps after received a line

sensor: m_gps_full_status(enum) 69 # out, updated with status of gps after every attempt to

to read characters from the gps

0 is good fix, m_gps_lat/lon update

>0 no fix see gps.h for list of why

sensor: m_gps_ignored_lat(lat) 69696969 # out, first few ignored gps fixes here

sensor: m_gps_ignored_lon(lon) 69696969 # published when m_gps_status == GPS_STATUS_FIRST_IGNORED_VALID(1)

sensor: m_gps_invalid_lat(lat) 69696969 # out, published on A lines

sensor: m_gps_invalid_lon(lon) 69696969

sensor: m_gps_toofar_lat(lat) 69696969 # out, published if too far from DR point

sensor: m_gps_toofar_lon(lat) 69696969 # M_GPS_STATUS == GPS_STATUS_TOOFAR_FIX(3)

sensor: m_gps_dist_from_dr(m) 69696969 # out, how far fix is from dead reckoned position

sensor: x_gps_reasonable_radius(m) 69696969 # out, how far fix CAN BE from dead reckoned position

= U_GPS_REASONABLE_FACTOR *

(U_GPS_UNCERTAINITY + secs_since_last_valid_gps_fix *

(U_MAX_WATER_SPEED + nominal glider horizontal speed))

sensor: u_gps_reasonable_factor(nodim) 1.0 # in, see equation above

sensor: u_gps_uncertainity(m) 30.0 # in, see equation above

This data is read from gps and published

sensor: m_gps_utc_day(byte) 0 # 1-31 Date/Time of position

sensor: m_gps_utc_month(byte) 0 # 1-12

sensor: m_gps_utc_year(byte) 0 # 00, 01, ... until Y3K

sensor: m_gps_utc_hour(byte) 0 # 0-23

sensor: m_gps_utc_minute(byte) 0 # 0-59

sensor: m_gps_utc_second(nodim) 0 # 0-59.xxxxxx

sensor: m_gps_speed(m/s) 0 # speed over ground

sensor: m_gps_heading(rad) 0 # magnetic heading

sensor: m_gps_mag_var(rad) 0 # mag_heading = true_heading + mag_var

mag_var>0 ==> variation is West (like on cape cod)

sensor: m_gps_uncertainty(nodim) 69696969 # out, Horizontal dilution of precision 0.5 to 99.9

sensor: m_gps_num_satellites(nodim) 69696969 # out, Number of satellites in use, 00 to 12

sensor: m_system_clock_lags_gps(sec) 0 # lagtime between persistor and gps clock

sensor: m_avg_system_clock_lags_gps(sec) 0 # exponential mean of above lagtime

sensor: u_alpha_system_clock_lags_gps(nodim) 0.05 # weight in exponential mean

generic time syncing sensor, called in surface.c and g_shell.c

sensor: u_max_lag_before_syncing_time(sec) 12 # sync_time when avg lag exceeds 12 secs

generic sensor to record syncing offsets, called in g_shell.c

sensor: x_system_clock_adjusted(sec) 0 # records the last sync_time offset

argos.c

sensor: c_argos_on(enum) 0 # <0 PTT is always turned off, even at surface

0 PTT powered off, but can be auto turned on at surface

>0 PTT is powered on and transmitting:

1 no diagnostic output

2 output xmitted chars to MLOG/TERM

3 output xmitted/recvd chars to MLOG/TERM

sensor: m_argos_on(bool) 0 # out, >0 means argos is actually turned on

sensor: m_argos_sent_data(bool) 0 # out, > 0 means data was sent to PTT

sensor: m_argos_is_xmitting(bool) 0 # out, > 0 means PTT is radiating

sensors to support new PTT format, along with legacy stuff

sensor: x_argos_type(enum) 0 # 0 SmartCAT (legacy)

1 X-CAT (external PIC)

sensor: f_argos_format(enum) 0 # 0 rev0 legacy (32 byte)

1 rev1 Mar05 (31 byte)

sensor: m_argos_timestamp(timestamp) 0 # last time argos was powered off

ctd.c

sensor: c_profile_on(sec) 0 # in, <0 is off, =0 as fast as possible

>0 that many seconds betweens measurements

sensor: c_profile_recall(msec) 2000 # in, <=0 no subcycle measurements

millisecs between subcycle measurements

c_profile_on must be 0 to enable

sensor: m_water_cond(S/m) 3 # out, conductivity

sensor: m_water_temp(degC) 10 # out

sensor: m_water_pressure(bar) 0 # out

avbot-devdrvr.c

A linux add-on cpu

#sensor: c_avbot_power(bool) 0 # in, power supplied to linux cpu

#sensor: m_avbot_power(bool) 0 # out, ditto

#sensor: c_avbot_enable(bool) 0 # in, linux cpu enabled to control

#sensor: m_avbot_enable(bool) 0 # out, ditto

#sensor: x_avbot_disabled(bool) 0 # out, true => hardware bit (currently persistor bit 29)

shorted to ground.

clears c_avbot_enable

#sensor: sci_avbot_proglet_is_installed(bool) 0 # in, t-> avbot_proglet installed on science.

```
#sensor: u_avbot_debug(nodim)   0       # required by gbus (485)



# iridium.c
sensor: c_iridium_on(enum) 1 # in
                        #  <0 turns it off
                        #   0 turns it off
                        #   1 turns it on, becomes 2nd console when connected
                        #   2 turns it on, no 2nd console
                        #   3 turns it on in "send data" mode
                        #   4 turns it on in "echo data" mode
                        #  >4 turns it off



sensor: c_iridium_reread_config_files(button)    0.0   # Set to force reread of:
                                        #    iridinit.* and loginexp.*
                                        # code sets it back to 0
                                        #   1 ==> read,parse and       USE
                                        #   2 ==> read,parse and DO NOT use
                                        #        (use for syntax checking)



      # Phone number+prefix, assuming 508 548-2446 target
      # For a commercial card:   0015085482446
      # For a military card:   006975085482446
```

```
    # You should put YOUR number in autoexec.mi


# Main number
sensor: c_iridium_lead_zeros(nodim)         2 # number of leading zeros in phone number
                                    #   typically 2 for both commercial or military
sensor: c_iridium_phone_num(digits) 15085482446 #  WRC phone number !no spaces!


# ALT number (RESOLVES MANTIS #255)
sensor: c_iridium_lead_zeros_alt(nodim)         2 # number of leading zeros in phone number
sensor: c_iridium_phone_num_alt(digits) 15085482446 # WRC phone number !no spaces!



    # Used to manage which phone number to use
sensor: u_iridium_failover_retries(nodim) 5 #! visible = True
                                # Maximum number of retries before failing over to
                                                                        # other number
sensor: m_iridium_attempt_num(nodim) 0 # keeps track of the number of retries for the
                                                                        # current number (Should be
initialized to 1)
sensor: c_iridium_current_num(enum) 0 # 0 - IRIDIUM_PHONE_NUM_PRIMARY
                                                                        # 1 -
IRIDIUM_PHONE_NUM_ALTERNATE



# How long to wait for modem to respond at various times
```

```
sensor: c_iridium_atok_timeout(sec)    30  # how long to wait for OK after AT
                                           # should be immediate if phone is attached
sensor: c_iridium_register(sec) 30 # minimum time for iridium to register after
                                   #  powerup. We do not try to dial for this many secs.
sensor: c_iridium_await_connect_max(mins) 5 # how long we will wait for a response
                                            #  after dialing the iridium phone number.
                                            #  When exceeded the iridium power is cycled.
                                            # Zero or negative means wait forever.

sensor: c_iridium_no_char_timeout(mins) 10  # How long to wait for a character at all other times
                                            # This is internally to clipped to never be less than 5 minutes
                                            # unless you are in lab_mode.  This is catch all to force an iridium
                                            # error (and a redial) if it ever gets "stuck"

sensor: c_iridium_power_on_delay(sec) 3 # min time between power on and sending AT
                                        # internally clipped to maximum of c_iridium_register secs
sensor: c_iridium_redial_delay(sec) 1 # delay time between redials. Values less than
                                      #  the cycle time (nominally two seconds)
                                      #  will delay till next cycle (i.e. 2 seconds)
```

sensor: c_iridium_time_til_callback(sec) 0.0 # Set this non-zero to have iridium

hang up and call back in that many seconds.

Call back is canceled if anyone sets C_IRIDUM_ON

sensor: u_iridium_max_time_til_callback(sec) 1800.0 # Maximum legal value for

C_IRIDIUM_TIME_TIL_CALLBACK

sensor: c_iridium_redials_per_on_off(nodim) 1 # how often we cycle the iridium

power when trying to connect. Min 1, Max 10.

sensor: c_iridium_cmd_echo(enum) 1 # 0 = do not echo modem commands; 1 = do echo

sensor: m_iridium_on(bool) 0.0 # out 0 it's off, 1 it's on

sensor: m_iridium_connected(bool) 0 # out 1==> modem is connected

sensor: m_iridium_console_on(enum) 0. # out. 0 = iridium console off, 1 = on

sensor: m_iridium_status(enum) 99.0 # out MODEM_NO_CARRIER = 0

MODEM_OK, = 1

MODEM_CONNECT, = 2

MODEM_ERROR, = 3

MODEM_NO_ANSWER, = 4

MODEM_BUSY, = 5

MODEM_NO_DIALTONE, = 6

LOGGING_IN = 7

LOGGED_ON = 8

MODEM_AWAITING_OK = 10,

MODEM_AWAITING_CONNECTION, = 11

MODEM_TIMEOUT, = 12

MODEM_UNKNOWN = 99,

NO_CHARS_TIMEOUT = 100,

sensor: m_iridium_waiting_registration(bool) # out, 1 ==> waiting for phone to register

sensor: m_iridium_waiting_redial_delay(bool) # out, 1 ==> waiting to redial

sensor: m_iridium_signal_strength(nodim) -1.0 # iridium received signal

strength indication (RSSI)

sensor: m_iridium_redials(nodim) 0.0 # out, number of redials since phone was on

sensor: m_iridium_dialed_num(nodim) 0.0 # out, number of times phone was dialed

incremented on every dial attempt

it is never reset

sensor: m_iridium_call_num(nodim) 0.0 # out, is incremented on every connection,

it is never reset

sensor: u_iridium_force_port(bool) 0 # in, iridium always uses J26 if true

nose.c

sensor: c_recovery_on(bool) 0 # In, nonzero deploys recovery system

thruster.c/thruster_g1.c

sensor: c_thruster_on(%) 0   # In, zero = off, 100 = all it's got

sensor: m_thruster_raw(int) 0 # out, pwm setting

sensor: x_thruster_has_current_sense(bool) 1 #Out. If thruster motor controller has current sensing. Set by thruster.c or thruster_g1.c

sensor: m_thruster_current(amp) 0 # Out, measured current.

sensor: c_thruster_current_cal(nodim) 0.038 # A / count cal for thruster current

sensor: u_thruster_cmd_change_min(%) 1 # In, minimum change between thruster commands to be considered a new set of cmds

sensor: u_thruster_delta_max(%) 2.5 #In, limit the change in command to be less than this for use_thruster = TM_PERCENT/TM_PERCENT_MAX to avoid spikes

sensor: u_avg_thruster_power_num(nodim) 10 # Number of samples to use for m_avg_thruster_power

sensor: m_avg_thruster_power(watt) 0 # Out, average measured power

sensor: f_thruster_power_max(watt) 14.5 # In, maximum allowable average thruster power

sensor: f_thruster_power_min(watt) 0.5 # In, minimum allowable average thruster power, for use_thruster=4

sensor: m_thruster_power_spike(enum) 0 # Running tally of power spikes above f_thruster_power_max

sensor: m_thruster_voltage(volts) 0 #Out, estimated input voltage to thruster. = C_THRUSTER_ON*M_BATTERY_INST/100

sensor: m_thruster_power(watt) 0 #Out m_thruster_voltage * m_thruster_current

sensor: m_thruster_amphr(amp-hrs) 0 #Out, integrated current

sensor: m_thruster_watthr(watt-hrs) 0 #Out, integrated power

sensor: x_use_thruster_for_abort_ascent(bool) 0 # Set by abend b_arg use_thruster_for_ascent. Whether or not to use the thruster in case of an abort to avoid hovering

sensor: u_min_thruster_abort_ascent_rate(m/s) -0.05 # In, minimum ascent rate before thruster kicks in. Must be < 0

surface.c thruster parameters

sensors to specify thruster burst behavior (b_arg: thruster_burst)

sensor: u_thruster_burst_volts(volts) 6 # In, command voltage for thruster

sensor: u_thruster_burst_secs(sec) 15 # In, turn on thruster for this long

dynamic_control.c thruster parameters

sensor: dc_c_thruster_on(%) 0 # In/out, What dynamic control wants c_thruster_on to be

sensor: u_thruster_inflection_holdoff(sec) 60.0 #In, how long to wait after inflection/commanded depth state change.

sensor: u_thruster_abort_inflection_holdoff(sec) 200.0 #In, how long to wait after the abort occured before using thruster (if x_use_thruster_for_abort_ascent is true).

sensor: x_thruster_state(enum) -1 # Out, why dc_c_thruster_on was commanded. Describes the state of the thruster controller

0: Thruster mode not enabled

1: Constant thruster command, use_thruster = 1 (command % of glider voltage) and use_thruster = 2 (command % of max voltage)

The following states apply to use_thruster=3 (command depth rate) and use_thruster=4 (command power)

2: Increased thruster: (M_DEPTH_RATE_THR_AVG_FINAL < command) (use_thruster=3)

or (M_AVG_THRUSTER_POWER < command) (use_thruster=4)

3: Decreased thruster: (M_DEPTH_RATE_THR_AVG_FINAL > command) (use_thruster=3)

or

4: Did not adjust thruster: (M_DEPTH_RATE_THR_AVG_FINAL within command and U_AP_THRUSTER_DEPTH_RATE_DEADBAND limits) (use_thruster=3)

or (M_AVG_THRUSTER_POWER within command and U_AP_THRUSTER_POWER_DEADBAND limits) (use_thruster=4)

5: Did not adjust thruster, not enough time has elapsed since last command (X_THRUSTER_AP_PERIOD) (use_thruster=3)

6: Did not adjust thruster: (not enough depth rate samples U_DEPTH_RATE_THR_AVG_NUM) (use_thruster=3)

or (not enough power samples U_AVG_THRUSTER_POWER_NUM) (use_thruster=4)

The following states apply to all thruster modes

7: Off, because inflecting or within the u_thruster_inflection_holdoff time period.

8: Off, because measured pitch is in the wrong direction (i.e. m_pitch < 0 on a climb)

9: On, Thruster burst mode (surface behavior: b_arg: thruster_burst

sensor: f_thruster_min_v(volts) 3.0 # In, minimum voltage to run the thruster

sensor: f_thruster_max_v(volts) 9.7 # In, maximum voltage to run the thruster

sensor: u_ap_thruster_delta_cmd(%) 10 # how much to increment thruster command to maintain depth rate/power

Parameters specific to use_thruster = power.

We use the thruster to maintain a desired input power

sensor: u_ap_thruster_power_deadband(watt) 0.5 # In, allow power between +/- this amount

sensor: m_thruster_power_error(watt) 0 # Out, cmd - meas

sensor: u_ap_thruster_power_p_gain(nodim) 2.5 # In, if > 0, then delta_thruster command = u_ap_thruster_power_p_gain * m_thruster_power_error

Otherwise, = u_ap_thruster_delta_cmd

If the new power feedback command is to be increased above u_thruster_power_limit_cmd, then

we limit the delta command by u_thruster_power_delta_max to avoid spikes for large commands

sensor: u_thruster_power_limit_cmd(%) 80 #In,

sensor: u_thruster_power_delta_max(%) 3 #In,

Parameters specific to use_thruster = depthrate.

We use the thruster to maintain a desired depth rate

sensor: u_ap_thruster_depth_rate_deadband(m/s) 0.02 # Allow depth rate between +/- this amount

sensor: u_thruster_ap_period(s) 30 # How often to run immediately after each command

sensor: x_thruster_ap_period(s) -1 # How often to check depth rate:

Immediately following a new command, we wait u_ap_thruster_depth_rate_period seconds

Any time after that, we run as fast as possible (-1)

sensor: u_depth_rate_thr_avg_num(enum) 3 # the number of data samples to collect for the m_depth_rate_thr_avg calculation.

sensor: m_depth_rate_thr_avg_final(m/s) 0.0 # Final value of the calculation

sensor: c_thruster_surface_secs(s) 0 # out, How long the thruster has been on in depth rate mode for this surfacing

sensor: c_thruster_depth_rate_secs(s) 0 # out, How long the thruster has been on in depth rate mode for the entire segment

sensor: c_thruster_surface_depth(m) 0 # out, What depth the thruster first turned on (in depth rate mode) for this surfacing

sensor: c_thruster_depth_rate_depth(m) 0 # out, What depth the thruster first turned on (in depth rate mode) for the entire segment

science.c/science_super.c

sensor: c_science_on(bool) 1 # In, nonzero turns on science uart

0 off

>=1 on + log errors

>=2 on + log successfully received variables

+ log errors

>=3 on + log all sent lines

+ log successfully received variables

+ log errors

>=4 on + log all received lines

+ log all sent lines

+ log successfully received variables

+ log errors

sensor: c_science_send_all(bool) 0 #! visible = True

T->send all sci_ vars from science but still log them on science.

F->just send standard subset but still log them all on science.

sensor: m_science_on(bool) 0 # Out, actual power state of science uart

sensor: sci_m_science_on(bool) 0 # In, set by science when powered on

clr by science when safe to power off

sensor: c_science_all_on(secs) 2 # in, if enabled this value is set into the

C_xxx_ON for all installed sensors on

the science computer as detected by

SCI_xxx_IS_INSTALLED on every cycle

sensor: c_science_all_on_enabled(bool) 1 # in, non-zero enables c_science_all_on

sensor: sci_software_ver(nodim) 0 # In, software version running on science

sensor: sci_reqd_heartbeat(secs) -1.0 # In. How often each side must communicate

over the clothesline

DISABLED, too many false alarms

sensor: m_science_sent_some_data(nodim) 0 # Out, incremented when the glider pulls a character

out of the clothesline buffer where chars received

from science processor are stored.

sensor: u_science_max_power_off_time(s) 120 # In, how long to wait for sci_m_science_on

to go low before giving up and yanking power

sensor: u_science_power_off_delay(s) 0.5 # In, how long to wait AFTER sci_m_science_on

has gone to 0 before yanking power. This

gives science a little time to clean up

sensor: u_max_clothesline_lag_for_consci(s) 20.0 # don't attempt to consci until

glider-science time lag is

below this.

sensor: m_science_unreadiness_for_consci(enum) 1 # 0 -> Ready

1 -> Not ready because sci_m_science_on = 0.

2 -> Not ready because m_science_clothesline_lag not updated.

3 -> Not ready because m_science_clothesline_lag

> u_max_clothesline_lag_for_consci.

4 -> Not ready because not checked yet.

sensor: m_science_ready_for_consci(bool) 0 # out, true -> clothesline ready for consci

determined in sensor_processing.c

sensor: x_clothesline_state(enum) 0 # out, state of the clothesline

0 = stopped

1 = running

2 = waiting for suspend acknowledgement

3 = suspended

4 = waiting for resume acknowledgement

sensor: x_sci_cmd_mode_state(enum) 0 # out, state of science console state machine

see science_cmd_execution.h, enum science_cmd_mode_t

sensor: u_sci_cmd_max_ack_wait_time(s) 60.0 # in, how long to wait for science to acknowdge request

to go to command mode

sensor: u_sci_cmd_max_consci_time(s) 3600. #! visible = True

in, maximum time in consci

sensor: u_science_send_time_limit_adjustment_factor(nodim) 0.5 # in, fudge factor to used with u_sci_cmd_max_consci_time

```
                                    #     to compute time limit on science send command


sensor: f_sci_max_input_process_time(msec) 200. # In, how long science driver can spend
                                    #     processing input lines from science
                                    #     on each call.  Set only to prevent
                                    #     science data from consuming all the
                                    #     glider cpu time.  Not really an issue
                                    #     with superscience, this replaces
                                    #     f_sci_max_sensors_per_call(nodim)


sensor: c_science_printout(nodim) 0  # How much science printout is seen
                                # on the glider:
                                #    0   none
                                #    1   proglet _begin()/_end()
                                #    2   proglet _start()/_stop()
                                #    3   proglet _run


sensor: c_science_stress_on(sensors/sec) 0 # causes proglet to send SCI_GENERIC_A-Z
                                    # this many times/sec for diagnostic purposes



sensor: sci_m_present_time(timestamp) 0 # In, written by science on every cycle
                                    #     their notion of time, secs since 1970
sensor: sci_m_present_secs_into_mission(sec) 0 # out, secs since mission started
sensor: m_science_clothesline_lag(s) 0  # out, How far behind science is
```

M_PRESENT_TIME - SCI_M_PRESENT_TIME

sensor: m_science_sync_time(timestamp) 0 # Out, Glider timestamp (secs since 1970) at the

request of Science for synchronizing clocks.

sensor: sci_wants_surface(enum) 0 # In, requests from science computer

0 science does not need to surface

1 science wants to surface at next reasonable opportunity

2 science wants to surface NOW!

sensor: sci_wants_comms(bool) 0 # In, t-> science computer wants direct comms

sensor: sci_wants_quiet(bool) 0 # In, t-> science computer wants glider in comatose behavior

PLACE HOLDER FOR OBSOLETED PROGLETS used in science_super.c and sample.c

sensor: c_obsolete_on(bool) -1

sensor: sci_obsolete_is_installed(bool) 0

sensor: sci_obsolete_var(nodim) 0

CTD data measured by Science. Updates m_water_cond, m_water_temp, & m_water_pressure

sensor: sci_ctd_is_installed(bool) 0 # in, t--> ctd installed on science

#

sensor: sci_ctd41_is_installed(bool) 0 # in, t--> ctd installed on science

sensor: sci_ctd41cp_is_installed(bool) 0 # in, t--> ctd installed on science

sensor: sci_nbctd_is_installed(bool) 0 # in, t--> ctd installed on science

sensor: sci_ctd41cp_sim_is_installed(bool) 0 # in, t--> ctd being simulated on science computer

sensor: c_ctd41cp_num_fields_to_send(nodim) 4 # in, number of columns to send on each

measurement, fields to send chosen

by order in the list below

A negative value signifies

to use this value as a bitmap.

The user may specify any

outputs regardless of order by

by using this sensor as a bitmap.

-1 * (2^(f1-1)+2^(f2-1)+...)

where fn is the field number.

For example, if the user wished

to just record temperature

they would use:

-1 * 2^(2-1) = -2

we use this one instead for a Neil Brown CTD

sensor: c_nbctd_num_fields_to_send(nodim) 3 # in, number of columns to send on each

measurement, fields to send chosen

by order in the list below

A negative value signifies

to use this value as a bitmap.

The user may specify any

outputs regardless of order by

```
                                # by using this sensor as a bitmap.
                                # -1 * (2^(f1-1)+2^(f2-1)+...)
                                # where fn is the field number.
                                # For example, if the user wished
                                # to just record temperature
                                # they would use:
                                # -1 * 2^(2-1) = -2

sensor: sci_water_cond(S/m) 3           # out, conductivity    f#=1
sensor: sci_water_temp(degC) 10          # out                f#=2
sensor: sci_water_pressure(bar) 0        # out                f#=3
sensor: sci_ctd41cp_timestamp(timestamp) 0 # out, secs since 1970 f#=4

# we use this one instead for a Neil Brown CTD
sensor: sci_nbctd_timestamp(timestamp) 0   # out, secs since 1970 f#=4

sensor: sci_generic_a(nodim)    0 # unspecified variables for science to use
sensor: sci_generic_b(nodim)    0
sensor: sci_generic_c(nodim)    0
sensor: sci_generic_d(nodim)    0
sensor: sci_generic_e(nodim)    0
sensor: sci_generic_f(nodim)    0
sensor: sci_generic_g(nodim)    0
sensor: sci_generic_h(nodim)    0
sensor: sci_generic_i(nodim)    0
```

sensor: sci_generic_j(nodim) 0

sensor: sci_generic_k(nodim) 0

sensor: sci_generic_l(nodim) 0

sensor: sci_generic_m(nodim) 0

sensor: sci_generic_n(nodim) 0

sensor: sci_generic_o(nodim) 0

sensor: sci_generic_p(nodim) 0

sensor: sci_generic_q(nodim) 0

sensor: sci_generic_r(nodim) 0

sensor: sci_generic_s(nodim) 0

sensor: sci_generic_t(nodim) 0

sensor: sci_generic_u(nodim) 0

sensor: sci_generic_v(nodim) 0

sensor: sci_generic_w(nodim) 0

sensor: sci_generic_x(nodim) 0

sensor: sci_generic_y(nodim) 0

sensor: sci_generic_z(nodim) 0

For testing connectivity

sensor: x_ping_glider_to_sci(nodim) 0 # Out, science driver increments this each cycle

and can be sent to science for testing

sensor: sci_ping_sci_to_glider(nodim) 0 # In, science can send this to us if its copy

does not match recvd version of x_ping_glider_to_sci

legacy sensor for Benthos Acoustic Modem amconnect.RUN

```
sensor: c_acoustic_modem_target_id(enum) 0 # Out, the address of the remote modem
                        #  (typically a deck unit) being called. Used by
                        #   the science program amconnct. min 0, max 31.

      # sensors for Benthos Acoustic Modem (bam) proglet

sensor: c_bam_on(sec)             0 # >0 secs between run cycles, <0 off,
                            # 0 = fast as possible
sensor: c_bam_mode(enum)          0 # 0: command mode
                            # 1: data collect mode
sensor: c_bam_target_id(enum) 1 # The address of the remote host modem being
                        # called (typically a deck unit, min 0, max 31).
sensor: c_bam_update_secs(sec)    120 # how often to transmit location and depth,
                            # <0 => don't transmit location and depth
                            # minimum value = c_bam_cmd_parse(sec) *
                            #     (c_bam_number_of_echos(nodim) + 1)
sensor: c_bam_inactivity_secs(sec)  60 # how long the modem must be quiet before
                            # location is broadcast
sensor: c_bam_cmd_parse_secs(sec)   5 # How often to check command input buffer
sensor: c_bam_number_of_echos(nodim) 3 # Number of times to echo commands
sensor: c_bam_chars_to_get_before_surfacing(nodim) 1000 # how many chars to collect
                                # in modmdata.dat before
                                # surfacing, <0 => don't
                                # collect any data
sensor: c_bam_datacol_report_secs(sec) 10 # How often to send bam_datacol
```

```
                        # output sensors to glider
                        # (xx_rcvd_chars_xx)

sensor: sci_bam_is_installed(bool)    # true -> proglet is installed
sensor: sci_bam_science_on(bool)      # false -> exit supersci app
                      # maps to c_science_on
sensor: sci_bam_rcvd_chars_since_last_report(nodim)   # num of chars heard in last 10 seconds
sensor: sci_bam_rcvd_chars_since_last_surfacing(nodim) # num of chars heard since last surfacing

   # HydroScat2 sensors
#sensor: c_hs2_on(sec)       0 # in, sets seconds between hs2 measurements
                  #  < 0 stops hs2 data collection
                  #  >=0 values are forced to be between 2 and 10 inclusive.
#sensor: sci_hs2_is_installed(bool) 0 # in, t--> installed on science
#sensor: sci_hs2_1bb(nodim)   0 # out, "bb"  backscatter for hs2 channel 1
#sensor: sci_hs2_1bbu(nodim)  0 #     "bbu" backscatter for hs2 channel 1
#sensor: sci_hs2_2bb(nodim)   0 #     "bb"  backscatter for hs2 channel 2
#sensor: sci_hs2_2bbu(nodim)  0 #     "bbu" backscatter for hs2 channel 2
#sensor: sci_hs2_3bb(nodim)   0 #     "bb"  backscatter for hs2 channel 3
#sensor: sci_hs2_3bbu(nodim)  0 #     "bbu" backscatter for hs2 channel 3

   # proglet bb2f: wet labs bb2f fluorometer / backscatter sensor
sensor: c_bb2f_on(sec)      0 # in, sets secs between measurements
                  # <0 stops, 0 fast as possible, >0 that many secs
sensor: sci_bb2f_is_installed(bool) 0 # in, t--> installed on science
```

```
sensor: c_bb2f_num_fields_to_send(nodim)   7   # in, number of columns to send on each
                                    #    measurement, fields to send chosen
                                    #    by order in the list below

                # output sensors, listed in PRIORITY order
                # e.g. if c_bb2f_num_fields_to_send is 3, cols 3,5,6 sent
sensor: sci_bb2f_b470(nodim)         0 # col 3, blue scatter
sensor: sci_bb2f_b700(nodim)         0 # col 5, red scatter
sensor: sci_bb2f_fluor(nodim)        0 # col 6, fluorescence
sensor: sci_bb2f_therm(nodim)        0 # col 7, thermistor
sensor: sci_bb2f_b470_ref(nodim)     0 # col 2, blue ref
sensor: sci_bb2f_b700_ref(nodim)     0 # col 4, red ref
sensor: sci_bb2f_counter(nodim)      0 # col 1, counter (resets to zero at each power-up)
sensor: sci_bb2f_timestamp(timestamp) 0 # secs since 1970


   # proglet bb2c: Wetlabs no clue what name is or data means
sensor: c_bb2c_on(sec)  2 # in, sets secs between measurements
                    # <0 stops, 0 fast as possible, >0 that many secs
sensor: sci_bb2c_is_installed(bool) 0 # in, t--> installed on science

sensor: u_bb2c_is_calibrated(bool) 0 # false, assume not calibrated

# for deriving bb2c engineering units, these should be tailered for each
```

```
# glider with this device in the science bay (these are defaults for RU04)
sensor: u_bb2c_beta532_factor(Mnodim) 7.494  # really 0.000007494 (see Mnodim doco above)
sensor: u_bb2c_beta660_factor(Mnodim) 1.8    # really 0.0000018     "    "     "     "
sensor: u_bb2c_beta532_offset(nodim)  55.37  # offset for eng unit conversion
sensor: u_bb2c_beta660_offset(nodim)  55.0   #  "    "          "       "

sensor: c_bb2c_num_fields_to_send(nodim)  9 # in, number of columns to send on each
                                #    measurement, fields to send chosen
                                #    by order in the list below
                # output sensors, listed in PRIORITY order
                # Note: date(col1) and time(col2) fields tossed
sensor: sci_bb2c_beta532_eng_units(nodim) 0 # derived from col 4
sensor: sci_bb2c_beta660_eng_units(nodim) 0 # derived from col 6
sensor: sci_bb2c_beta532(nodim)         0 # col 4
sensor: sci_bb2c_beta660(nodim)         0 # col 6
sensor: sci_bb2c_cdom(nodim)            0 # col 8
sensor: sci_bb2c_ref1(nodim)            0 # col 3
sensor: sci_bb2c_ref2(nodim)            0 # col 5
sensor: sci_bb2c_ref3(nodim)            0 # col 7
sensor: sci_bb2c_temp(nodim)            0 # col 9
sensor: sci_bb2c_timestamp(timestamp)     0 # secs since 1970


  # proglet bb2lss, wetlabs Light Scatter Sensor
sensor: c_bb2lss_on(sec)  2 # in, sets secs between measurements
```

<0 stops, 0 fast as possible, >0 that many secs

sensor: sci_bb2lss_is_installed(bool) 0 # in, t--> installed on science

sensor: u_bb2lss_is_calibrated(bool) 0 # false, assume not calibrated

for deriving bb2lss engineering units, these should be tailered for each

glider with this device in the science bay (these are defaults for RU04)

sensor: u_bb2lss_beta880_factor(Mnodim) 2.664 # really 0.000002664 (see Mnodim doco above)

sensor: u_bb2lss_beta880_offset(nodim) 52.97 # offset for eng unit conversion

sensor: c_bb2lss_num_fields_to_send(nodim) 6 # in, number of columns to send on each

measurement, fields to send chosen

by order in the list below

output sensors, listed in PRIORITY order

Note: date(col1) and time(col2) fields tossed

sensor: sci_bb2lss_beta880_eng_units(nodim) 0 # derived from col4

sensor: sci_bb2lss_beta880(nodim) 0 # col4

sensor: sci_bb2lss_lss(nodim) 0 # col6

sensor: sci_bb2lss_ref1(nodim) 0 # col3

sensor: sci_bb2lss_ref2(nodim) 0 # col5

sensor: sci_bb2lss_temp(nodim) 0 # col7

sensor: sci_bb2lss_timestamp(timestamp) 0 # secs since 1970

#proglet sam: Wetlabs: Scattering Attenuation Meter

sensor: c_sam_on(sec) 2 # in, sets secs between measurements

<0 stops, 0 fast as possible, >0 that many secs

sensor: sci_sam_is_installed(bool) 0 # in, t--> installed on science

sensor: u_sam_is_calibrated(bool) 0 # false, assume not calibrated

sensor specific calibration constants

sensor: u_sam_do1(nodim) 68.0 # for deriving engineering units

sensor: u_sam_do2(nodim) 85.0 # " " " "

sensor: u_sam_exp1coeff(nodim) 0.055 # " " " "

sensor: u_sam_exp2coeff(nodim) 4.448 # " " " "

sensor: u_sam_offset(nodim) 7.0 # " " " "

sensor: u_sam_eff_pathlength(nodim) 0.104 # " " " "

sensor: u_sam_a(nodim) 10.0 # " " " "

sensor: u_sam_transition_val(nodim) 1.8 # " " " "

sensor: u_sam_median_window(nodim) 10 # valid range 1-15 (for eng units)

sensor: c_sam_num_fields_to_send(nodim) 9 # in, number of columns to send on each

measurement, fields to send chosen

by order in the list below

output sensors, listed in PRIORITY order

sensor: sci_sam_c_mix(nodim) 0 # engineering unit1, derived from cols 2 and 3

sensor: sci_sam_vis(nodim) 0 # engineering unit2, derived from cols 2 and 3

sensor: sci_sam_filter_age(sec) 0 # age of oldest sample in median window

sensor: sci_sam_s1_filtered(nodim) 0 # median filtered version of sci_sam_s1

sensor: sci_sam_s2_filtered(nodim) 0 # median filtered version of sci_sam_s2

sensor: sci_sam_s1(nodim) 0 # col 2

sensor: sci_sam_s2(nodim) 0 # col 3

sensor: sci_sam_ref(nodim) 0 # col 1

sensor: sci_sam_temp(nodim) 0 # col 4

proglet whpar: WHOI Photosynthetic Active Radiation

#sensor: c_whpar_on(sec) 0 # in, sets secs between measurements

<0 stops, 0 fast as possible, >0 that many secs

#sensor: sci_whpar_is_installed(bool) 0 # in, t--> installed on science

#sensor: c_whpar_num_fields_to_send(nodim) 6 # in, number of columns to send on each

measurement, fields to send chosen

by order in the list below

output sensors, listed in PRIORITY order

e.g. if c_whpar_num_fields_to_send is 2, par and voltage sent

#sensor: sci_whpar_par(nodim) 0 # col 2, Primary PAR

#sensor: sci_whpar_ref(nodim) 0 # col 3, Second PAR or reference

#sensor: sci_whpar_therm(nodim) 0 # col 4, Temperature

#sensor: sci_whpar_volt(nodim) 0 # col 5, Voltage

#sensor: sci_whpar_counter(nodim) 0 # col 1, Frame counter

#sensor: sci_whpar_spare(nodim) 0 # col 6, Spare

#sensor: sci_whpar_timestamp(timestamp) 0 # secs since 1970

```
   # proglet whgpbm: WHOI Glider  Bathy-PhotoMeter
#sensor: c_whgpbm_on(sec)       0 # in, sets secs between measurements
                        # <0 stops, 0 fast as possible, >0 that many secs
#sensor: sci_whgpbm_is_installed(bool) 0 # in, t--> installed on science

#sensor: c_whgpbm_num_fields_to_send(nodim)  7   # in, number of columns to send on each
                                    #    measurement, fields to send chosen
                                    #    by order in the list below

                # output sensors, listed in PRIORITY order
                # e.g. if c_whgpbm_num_fields_to_send is 2, par and bio are sent
#sensor: sci_whgpbm_par(nodim)          0 # col 3, PPPPP
#sensor: sci_whgpbm_biolumin(nodim)      0 # col 2, BBBBB
#sensor: sci_whgpbm_interval(nodim)      0 # col 7, RR
#sensor: sci_whgpbm_volt_excite(nodim)   0 # col 4, LLLL
#sensor: sci_whgpbm_volt_left(nodim)     0 # col 5, QQQQ
#sensor: sci_whgpbm_volt_bat(nodim)      0 # col 6, VVVV
#sensor: sci_whgpbm_counter(nodim)       0 # col 1, CCCC
#sensor: sci_whgpbm_timestamp(timestamp) 0 # secs since 1970


   # proglet whfctd: WHoi Fast CTD
sensor: c_whfctd_on(sec)  10.0 # in, sets secs between measurements
                        # <0 stops, 0 fast as possible, >0 that many secs
```

sensor: c_whfctd_num_fields_to_send(nodim) 8

in, number of columns to send on each measurement,

fields to send chosen by order in the list above

sensor: sci_whfctd_is_installed(bool) 0 # in, t--> installed on science

sensor: sci_whfctd_ref_hi(nodim) 0 # col 1, AAAAAAAA

sensor: sci_whfctd_ref_mid(nodim) 0 # col 2, BBBBBBB

sensor: sci_whfctd_ref_lo(nodim) 0 # col 3, CCCCCC

sensor: sci_whfctd_raw_temp(nodim) 0 # col 4, DDDDDDD

sensor: sci_whfctd_raw_con1(nodim) 0 # col 5, EEEEEE

sensor: sci_whfctd_raw_con2(nodim) 0 # col 6, FFFFFFF

sensor: sci_whfctd_raw_pres(nodim) 0 # col 7, GGGGGG

sensor: sci_whfctd_elap_time(nodim) 0 # col 8, HHHHH

proglet MoteOPD

Mote Marine Laboratory Optical Phytoplankton Discriminator (OPD)

last modified: ahails@mote.org 29 August 2011

sensor: c_moteopd_on(sec) -1 # >=0 turns it on, <0 stops it

sensor: c_moteopd_debug(bool) 0 # 1 for verbose logging

sensor: c_moteopd_data_overtime(sec) 600 # Max. time to allow before OPD data flagged as overdue

sensor: sci_moteopd_is_installed(bool) 0 # installed on science

sensor: sci_moteopd_sn(nodim) 0 # OPD unit serial number

sensor: sci_moteopd_status(nodim) 0 # OPD's binary cumulative error code

sensor: sci_moteopd_volt(nodim) 0 # its measured supply voltage, VDC

sensor: sci_moteopd_press(nodim) 0 # filter backpressure, psi

sensor: sci_moteopd_cdomref(nodim) 0 # remaining cdom reference fluid supply, mL

sensor: sci_moteopd_int_time(nodim) 0 # spectrometer integration time, msec

sensor: sci_moteopd_start_time(timestamp) 0 # timestamp, unix

sensor: sci_moteopd_stop_time(timestamp) 0 # timestamp, unix

sensor: sci_moteopd_absorb_a(nodim) 0 # slope of best-fit line of CDOM absorbance

sensor: sci_moteopd_absorb_b(nodim) 0 # intercept of best-fit line of CDOM absorbance

sensor: sci_moteopd_corr0(nodim) 0 # similarity index for the 0th species file

sensor: sci_moteopd_corr1(nodim) 0 # similarity index for the 1st species file

sensor: sci_moteopd_corr2(nodim) 0 # similarity index for the 2nd species file

sensor: sci_moteopd_corr3(nodim) 0 # etc. these vary with OPD setup, may be all or just 0th

sensor: sci_moteopd_corr4(nodim) 0 #

sensor: sci_moteopd_corr5(nodim) 0 #

sensor: sci_moteopd_corr6(nodim) 0 #

sensor: sci_moteopd_corr7(nodim) 0 #

sensor: sci_moteopd_corr8(nodim) 0 #

sensor: sci_moteopd_corr9(nodim) 0 #

sensor: sci_moteopd_corr10(nodim) 0 #

sensor: sci_moteopd_corr11(nodim) 0 #

sensor: sci_moteopd_logout(nodim) 0 # 0=no logout, 1=successfully logged out before power down.

proglet hydrophone

sensor: c_hydrophone_on(sec) -1.0 # positive or zero turns it on and starts sampling sequence

sensor: c_hydrophone_pre_delay(sec) 15.0 # delay between proglet start and sample start

sensor: c_hydrophone_post_delay(sec) 30.0 # delay between sample done and starting over

sensor: c_hydrophone_duration(sec) 30.0 # how long a measurement

sensor: c_hydrophone_gain(nodim) 3.0 # 0-7

sensor: c_hydrophone_num_channels(nodim) 1.0 # 1-4

sensor: c_hydrophone_sample_rate(Hz) 5000.0 # 1000-5000, how fast to AD

sensor: c_hydrophone_drive_num(nodim) 3.0 # 2->C:, 3:->D: etc

sensor: c_hydrophone_pre_pings(nodim) 1.0 # number of pings before sample

sensor: c_hydrophone_post_pings(nodim) 2.0 # number of pings after sample

sensor: sci_hydrophone_is_installed(bool) 0.0 # T-> if proglet installed

sensor: sci_hydrophone_collecting(nodim) 0.0 # set during collection to sample#, DDHHMM

sample as filename, less two alpha chars

which encode year and month

proglet hard_disk

sensor: sci_hard_disk_is_installed(bool) 0.0 # true means installed

proglet bbfl2s: wet labs bbfl2slo fluorometer / backscatter sensor

sensor: c_bbfl2s_on(sec) 2 # in, sets secs between measurements

<0 stops, 0 fast as possible, >0 that many secs

sensor: sci_bbfl2s_is_installed(bool) 0 # in, t--> installed on science

sensor: c_bbfl2s_num_fields_to_send(nodim) 10 # in, number of columns to send on each

measurement, fields to send chosen

by order in the list below

sensor: u_bbfl2s_is_calibrated(bool) 0 # false, assume not calibrated

sensor specific input calibration constants (defaults for BBFL2SLO-234)

sensor: u_bbfl2s_bb_cwo(nodim) 55 # clean water offset, nodim == counts

sensor: u_bbfl2s_chlor_cwo(nodim) 56 # clean water offset, nodim == counts

sensor: u_bbfl2s_cdom_cwo(nodim) 54 # clean water offset, nodim == counts

sensor: u_bbfl2s_bb_sf(Mnodim) 2.47 # scale factor (0.00000247)

sensor: u_bbfl2s_chlor_sf(ug/l/nodim) 0.0125 # scale factor to get units

sensor: u_bbfl2s_cdom_sf(ppb/nodim) 0.0979 # scale factor to get units

output sensors, listed in PRIORITY order

e.g. if c_bbfl2s_num_fields_to_send is 3, cols derived

from 4,6,8 sent

sensor: sci_bbfl2s_bb_scaled(nodim) 0 # derived from col 4

sensor: sci_bbfl2s_chlor_scaled(ug/l) 0 # derived from col 6

sensor: sci_bbfl2s_cdom_scaled(ppb) 0 # derived from col 8

sensor: sci_bbfl2s_bb_sig(nodim) 0 # col 4

sensor: sci_bbfl2s_chlor_sig(nodim) 0 # col 6

sensor: sci_bbfl2s_cdom_sig(nodim) 0 # col 8

sensor: sci_bbfl2s_bb_ref(nodim) 0 # col 3

sensor: sci_bbfl2s_chlor_ref(nodim) 0 # col 5

sensor: sci_bbfl2s_cdom_ref(nodim) 0 # col 7

sensor: sci_bbfl2s_temp(nodim) 0 # col 9

sensor: sci_bbfl2s_timestamp(timestamp) 0 # secs since 1970

proglet bbfl2sV2: wet labs bbfl2slo fluorometer / backscatter sensor, 2nd conf

sensor: c_bbfl2sV2_on(sec) 2 # in, sets secs between measurements

<0 stops, 0 fast as possible, >0 that many secs

sensor: sci_bbfl2sV2_is_installed(bool) 0 # in, t--> installed on science

sensor: c_bbfl2sV2_num_fields_to_send(nodim) 10 # in, number of columns to send on each

measurement, fields to send chosen

by order in the list below

sensor: u_bbfl2sV2_is_calibrated(bool) 0 # false, assume not calibrated

sensor specific input calibration constants (defaults for BBFL2SLO-407#p)

sensor: u_bbfl2sV2_bb_cwo(nodim) 42 # 532 nm, clean water offset, nodim == counts

sensor: u_bbfl2sV2_fl1_cwo(nodim) 43 # Phycoerythrin, clean water offset, nodim == counts

sensor: u_bbfl2sV2_fl2_cwo(nodim) 52 # CDOM, clean water offset, nodim == counts

sensor: u_bbfl2sV2_bb_sf(Mnodim) 8.328 # 532 nm, scale factor (0.000008328)

sensor: u_bbfl2sV2_fl1_sf(nodim) 0.0434 # Phycoerythrin, scale factor to get units

sensor: u_bbfl2sV2_fl2_sf(nodim) 0.0930 # CDOM, scale factor to get units

output sensors, listed in PRIORITY order

e.g. if c_bbfl2s_num_fields_to_send is 3, cols derived

from 4,6,8 sent

sensor: sci_bbfl2sV2_bb_scaled(nodim) 0 # derived from col 4

sensor: sci_bbfl2sV2_fl1_scaled(nodim) 0 # derived from col 6

sensor: sci_bbfl2sV2_fl2_scaled(nodim) 0 # derived from col 8

sensor: sci_bbfl2sV2_bb_sig(nodim) 0 # col 4

sensor: sci_bbfl2sV2_fl1_sig(nodim) 0 # col 6

sensor: sci_bbfl2sV2_fl2_sig(nodim) 0 # col 8

sensor: sci_bbfl2sV2_bb_ref(nodim) 0 # col 3

sensor: sci_bbfl2sV2_fl1_ref(nodim) 0 # col 5

sensor: sci_bbfl2sV2_fl2_ref(nodim) 0 # col 7

sensor: sci_bbfl2sV2_therm(nodim) 0 # col 9

sensor: sci_bbfl2sV2_timestamp(timestamp) 0 # secs since 1970

proglet fl3slo: wet labs fl3slo fluorometer triplet sensor

sensor: c_fl3slo_on(sec) 2 # in, sets secs between measurements

<0 stops, 0 fast as possible, >0 that many secs

sensor: sci_fl3slo_is_installed(bool) 0 # in, t--> installed on science

sensor: c_fl3slo_num_fields_to_send(nodim) 10 # in, number of columns to send on each

measurement, fields to send chosen

by order in the list below

sensor: u_fl3slo_is_calibrated(bool) 0 # false, assume not calibrated

sensor specific input calibration constants (defaults for FL3-341)

sensor: u_fl3slo_chlor_cwo(nodim) 55 # clean water offset, nodim == counts

sensor: u_fl3slo_phyco_cwo(nodim) 55 # clean water offset, nodim == counts

sensor: u_fl3slo_cdom_cwo(nodim) 55 # clean water offset, nodim == counts

sensor: u_fl3slo_chlor_sf(ug/l/nodim) 0.0126 # scale factor to get units

sensor: u_fl3slo_phyco_sf(ppb/l/nodim) 0.0459 # scale factor to get units

sensor: u_fl3slo_cdom_sf(ppb/l/nodim) 0.0984 # scale factor to get units

output sensors, listed in PRIORITY order

e.g. if c_fl3slo_num_fields_to_send is 3, cols derived

from 4,6,8 sent

sensor: sci_fl3slo_chlor_units(ug/l) 0 # derived from col 4

sensor: sci_fl3slo_phyco_units(ppb) 0 # derived from col 6

sensor: sci_fl3slo_cdom_units(QSDE) 0 # derived from col 8

sensor: sci_fl3slo_chlor_sig(nodim) 0 # col 4

sensor: sci_fl3slo_phyco_sig(nodim) 0 # col 6

sensor: sci_fl3slo_cdom_sig(nodim) 0 # col 8

sensor: sci_fl3slo_chlor_ref(nodim) 0 # col 3

sensor: sci_fl3slo_phyco_ref(nodim) 0 # col 5

sensor: sci_fl3slo_cdom_ref(nodim) 0 # col 7

sensor: sci_fl3slo_temp(nodim) 0 # col 9

sensor: sci_fl3slo_timestamp(timestamp) 0 # secs since 1970

proglet bb3slo: wet labs bb3slo backscatter triplet sensor

sensor: c_bb3slo_on(sec) 2 # in, sets secs between measurements

<0 stops, 0 fast as possible, >0 that many secs

sensor: sci_bb3slo_is_installed(bool) 0 # in, t--> installed on science

sensor: c_bb3slo_num_fields_to_send(nodim) 10 # in, number of columns to send on each

measurement, fields to send chosen

by order in the list below

sensor: u_bb3slo_is_calibrated(bool) 0 # false, assume not calibrated

sensor specific input calibration constants (defaults for BB3SLO-207)

sensor: u_bb3slo_b470_do(nodim) 51 # dark offset, nodim == counts

sensor: u_bb3slo_b532_do(nodim) 51 # dark offset, nodim == counts

sensor: u_bb3slo_b660_do(nodim) 114 # dark offset, nodim == counts

sensor: u_bb3slo_b470_sf(Mnodim) 0.117 # scale factor (0.000000117)

sensor: u_bb3slo_b532_sf(Mnodim) 8.17 # scale factor (0.00000817)

sensor: u_bb3slo_b660_sf(Mnodim) 3.85 # scale factor (0.00000385)

output sensors, listed in PRIORITY order

e.g. if c_bb3slo_num_fields_to_send is 3, cols derived from 4,6,8 sent

sensor: sci_bb3slo_b470_scaled(nodim)   0 # from col 4, blue

sensor: sci_bb3slo_b532_scaled(nodim)   0 # from col 6, green

sensor: sci_bb3slo_b660_scaled(nodim)   0 # from col 8, red

sensor: sci_bb3slo_b470_sig(nodim)   0 # col 4, blue

sensor: sci_bb3slo_b532_sig(nodim)   0 # col 6, green

sensor: sci_bb3slo_b660_sig(nodim)   0 # col 8, red

sensor: sci_bb3slo_b470_ref(nodim)   0 # col 3, blue

sensor: sci_bb3slo_b532_ref(nodim)   0 # col 5, green

sensor: sci_bb3slo_b660_ref(nodim)   0 # col 7, red

sensor: sci_bb3slo_temp(nodim)   0 # col 9

sensor: sci_bb3slo_timestamp(timestamp) 0 # secs since 1970

proglet oxy3835: Aanderaa Oxygen Optode 3835

#sensor: c_oxy3835_on(sec)   2 # in, sets secs between measurements

                # <0 stops, 0 fast as possible, >0 that many secs

#sensor: sci_oxy3835_is_installed(bool) 0 # in, t--> installed on science

#sensor: c_oxy3835_num_fields_to_send(nodim)   3   # in, number of columns to send on each

                #   measurement, fields to send chosen

                #   by order in the list below

output sensors, listed in PRIORITY order

```
                # e.g. if c_oxy3835_num_fields_to_send is 3, cols 3,4,5 sent
#sensor: sci_oxy3835_oxygen(nodim)       0 # col 3, oxygen
#sensor: sci_oxy3835_saturation(nodim)    0 # col 4, saturation
#sensor: sci_oxy3835_temp(nodim)         0 # col 5, temperature
#sensor: sci_oxy3835_timestamp(timestamp) 0 # secs since 1970


# proglet oxy3835_wphase: Aanderaa Oxygen Optode 3835
sensor: c_oxy3835_wphase_on(sec)       2 # in, sets secs between measurements
                        # <0 stops, 0 fast as possible, >0 that many secs
sensor: sci_oxy3835_wphase_is_installed(bool) 0 # in, t--> installed on science

sensor: c_oxy3835_wphase_num_fields_to_send(nodim)   10   # in, number of columns to send on each
                                # measurement, fields to send chosen
                                # by order in the list below

            # output sensors, listed in PRIORITY order
            # e.g. if c_oxy3835_wphase_num_fields_to_send is 3, cols 3,4,5 sent
sensor: sci_oxy3835_wphase_oxygen(nodim)       0 # col 3, oxygen
sensor: sci_oxy3835_wphase_saturation(nodim)    0 # col 4, saturation
sensor: sci_oxy3835_wphase_temp(nodim)         0 # col 5, temperature
sensor: sci_oxy3835_wphase_dphase(nodim)       0 # col 6, d-phase
sensor: sci_oxy3835_wphase_bphase(nodim)       0 # col 7, b-phase
sensor: sci_oxy3835_wphase_rphase(nodim)       0 # col 8, r-phase
sensor: sci_oxy3835_wphase_bamp(nodim)         0 # col 9, b-amp
```

sensor: sci_oxy3835_wphase_bpot(nodim) 0 # col 10, b-pot

sensor: sci_oxy3835_wphase_ramp(nodim) 0 # col 11, r-amp

sensor: sci_oxy3835_wphase_rawtemp(nodim) 0 # col 12, RawTemp

sensor: sci_oxy3835_wphase_timestamp(timestamp) 0 # secs since 1970

proglet viper: DMA Viper Processor

sensor: c_viper_on(sec) -1.0 # positive or zero turns it on and starts sampling sequence

sensor: c_viper_turn_on_timeout(sec) 120.0 # max wait time for viper to power on

sensor: c_viper_collect_timeout(sec) 200.0 # max wait time for viper to collect/analyse acoustic data

sensor: c_viper_reset_timeout(sec) 60.0 # max wait time for viper to respond to reset gain command

sensor: c_viper_start_sampling_timeout(sec) 60.0 # max wait time for viper to respond to start sampling command

sensor: c_viper_detection_done_timeout(sec) 60.0 # max wait time for viper to respond to detection done command

sensor: c_viper_turn_off_timeout(sec) 120.0 # max wait time for viper to power off

sensor: c_viper_gain(nodim) 3.0 # 0-7 gain sent to viper

sensor: c_viper_max_sample_starts(nodim) 3.0 # max allowable attempts to obtain a definitive detection

sensor: c_viper_max_errors(nodim) 3.0 # max number of viper errors before mission abort

sensor: sci_viper_power_on(bool) 0 # power state of the Viper, true -> on

sensor: sci_viper_error(nodim) 0 # unique number for each error sequence

sensor: sci_viper_target(enum) 0 # target priority returned by Viper

sensor: sci_viper_collect_time(sec) 0 # data collection time returned by Viper

sensor: sci_viper_is_installed(bool) 0.0 # T-> if proglet installed

sensor: sci_viper_finished(bool) 0.0 # T-> viper is ready to be powered down

sensor: sci_viper_collecting(bool) 0.0 # T-> viper is doing it's thing, comatose time

proglet ocr504R: Satlantic OCR-504 Radiance configuration

#Inputs

sensor: c_ocr504R_on(sec) 0 # sets secs between how often data is sent

<0 stops, 0 fast as possible, 0> that many secs

sensor: u_ocr504R_is_calibrated(bool) 0 # needs to be set in autoexec.mi

sensor specific calibration constants (defaults for S/N 004)

sensor: u_ocr504R_dark_counts_c1(nodim) 2147326431.3 # dark offset for channel 1

sensor: u_ocr504R_cal_coeff_c1(Tnodim) 29310.139102 # calibration factor for channel 1

sensor: u_ocr504R_immersion_coeff_c1(nodim) 1.758 # immersion factor for channel 1

sensor: u_ocr504R_dark_counts_c2(nodim) 2147357165.1 # dark offset for channel 2

sensor: u_ocr504R_cal_coeff_c2(Tnodim) 33825.794480 # calibration factor for channel 2

sensor: u_ocr504R_immersion_coeff_c2(nodim) 1.752 # immersion factor for channel 2

sensor: u_ocr504R_dark_counts_c3(nodim) 2147621476.7 # dark offset for channel 3

sensor: u_ocr504R_cal_coeff_c3(Tnodim) 29314.178969 # calibration factor for channel 3

sensor: u_ocr504R_immersion_coeff_c3(nodim) 1.746 # immersion factor for channel 3

sensor: u_ocr504R_dark_counts_c4(nodim) 2147499550.4 # dark offset for channel 4

sensor: u_ocr504R_cal_coeff_c4(Tnodim) 18677.199017 # calibration factor for channel 4

sensor: u_ocr504R_immersion_coeff_c4(nodim) 1.739   # immersion factor for channel 4

sensor: u_ocr504R_Vin_a0(nodim) 0.0 # polynomial coefficient to scale Vin

sensor: u_ocr504R_Vin_a1(nodim) 0.03 # polynomial coefficient to scale Vin

sensor: u_ocr504R_itemp_a0(nodim) -50.0 # polynomial coefficient to scale itemp

sensor: u_ocr504R_itemp_a1(nodim)   0.5 # polynomial coefficient to scale itemp

sensor: c_ocr504R_num_fields_to_send(nodim) 16

number of columns to send on each

measurement, fields to send chosen

by order in the list below

sensor: sci_ocr504R_is_installed(bool) 0 # in, t--> installed on science

#Outputs, in order of priority:

sensor: sci_ocr504R_rad1(uW/cm^2/nm) # from channel1

sensor: sci_ocr504R_rad2(uW/cm^2/nm) # from channel2

sensor: sci_ocr504R_rad3(uW/cm^2/nm) # from channel3

sensor: sci_ocr504R_rad4(uW/cm^2/nm) # from channel4

sensor: sci_ocr504R_itemp(Celsius)   # internal temperature of instrument

sensor: sci_ocr504R_Vin(volts)   # regulated input voltage

```
sensor: sci_ocr504R_fcount(nodim)    # 0-255, count of frame transmitted
sensor: sci_ocr504R_channel1(nodim)  # raw counts from discrete optical waveband 1
sensor: sci_ocr504R_channel2(nodim)  # raw counts from discrete optical waveband 2
sensor: sci_ocr504R_channel3(nodim)  # raw counts from discrete optical waveband 3
sensor: sci_ocr504R_channel4(nodim)  # raw counts from discrete optical waveband 4
sensor: sci_ocr504R_itemp_raw(nodim) # raw pre-scaled temperature
sensor: sci_ocr504R_Vin_raw(nodim)   # raw pre-scaled regulated input voltage
sensor: sci_ocr504R_timer(sec)       # seconds since initialization (power-on)
sensor: sci_ocr504R_delay(msec)      # milliseconds offset to timer for
                                     # accurate indication of when frame's sensors
                                     # were sampled
sensor: sci_ocr504R_cksum(nodim)     # data integrity sensor, checksum on frame


# proglet ocr504I: Satlantic OCR-504 Irradiance configuration

#Inputs
sensor: c_ocr504I_on(sec)   0   # sets secs between how often data is sent
                                # <0 stops, 0 fast as possible, 0> that many secs

sensor: u_ocr504I_is_calibrated(bool)  0 # needs to be set in autoexec.mi

# sensor specific calibration constants (defaults for S/N 089)

sensor: u_ocr504I_dark_counts_c1(nodim) 2147679780.3 # dark offset for channel 1
```

sensor: u_ocr504l_cal_coeff_c1(Tnodim) 1636922.3650 # calibration factor for channel 1

sensor: u_ocr504l_immersion_coeff_c1(nodim) 1.368 # immersion factor for channel 1

sensor: u_ocr504l_dark_counts_c2(nodim) 2147446582.0 # dark offset for channel 2

sensor: u_ocr504l_cal_coeff_c2(Tnodim) 1940758.5765 # calibration factor for channel 2

sensor: u_ocr504l_immersion_coeff_c2(nodim) 1.410 # immersion factor for channel 2

sensor: u_ocr504l_dark_counts_c3(nodim) 2147390884.4 # dark offset for channel 3

sensor: u_ocr504l_cal_coeff_c3(Tnodim) 2286152.2061 # calibration factor for channel 3

sensor: u_ocr504l_immersion_coeff_c3(nodim) 1.365 # immersion factor for channel 3

sensor: u_ocr504l_dark_counts_c4(nodim) 2147443303.2 # dark offset for channel 4

sensor: u_ocr504l_cal_coeff_c4(Tnodim) 1804514.9462 # calibration factor for channel 4

sensor: u_ocr504l_immersion_coeff_c4(nodim) 1.372 # immersion factor for channel 4

sensor: u_ocr504l_Vin_a0(nodim) 0.0 # polynomial coefficient to scale Vin

sensor: u_ocr504l_Vin_a1(nodim) 0.03 # polynomial coefficient to scale Vin

sensor: u_ocr504l_itemp_a0(nodim) -50.0 # polynomial coefficient to scale itemp

sensor: u_ocr504l_itemp_a1(nodim) 0.5 # polynomial coefficient to scale itemp

sensor: c_ocr504l_num_fields_to_send(nodim) 16

number of columns to send on each

```
                # measurement, fields to send chosen
                # by order in the list below

sensor: sci_ocr504I_is_installed(bool) 0 # in, t--> installed on science

#Outputs, in order of priority:
sensor: sci_ocr504I_irrad1(uW/cm^2/nm) # from channel1
sensor: sci_ocr504I_irrad2(uW/cm^2/nm) # from channel2
sensor: sci_ocr504I_irrad3(uW/cm^2/nm) # from channel3
sensor: sci_ocr504I_irrad4(uW/cm^2/nm) # from channel4
sensor: sci_ocr504I_itemp(Celsius)     # internal temperature of instrument
sensor: sci_ocr504I_Vin(volts)         # regulated input voltage
sensor: sci_ocr504I_fcount(nodim)      # 0-255, count of frame transmitted
sensor: sci_ocr504I_channel1(nodim)    # raw counts from discrete optical waveband 1
sensor: sci_ocr504I_channel2(nodim)    # raw counts from discrete optical waveband 2
sensor: sci_ocr504I_channel3(nodim)    # raw counts from discrete optical waveband 3
sensor: sci_ocr504I_channel4(nodim)    # raw counts from discrete optical waveband 4
sensor: sci_ocr504I_itemp_raw(nodim)   # raw pre-scaled temperature
sensor: sci_ocr504I_Vin_raw(nodim)     # raw pre-scaled regulated input voltage
sensor: sci_ocr504I_timer(sec)         # seconds since initialization (power-on)
sensor: sci_ocr504I_delay(msec)        # milliseconds offset to timer for
                                       # accurate indication of when frame's sensors
                                       # were sampled
sensor: sci_ocr504I_cksum(nodim)       # data integrity sensor, checksum on frame
```

```
# proglet ocr507R: Satlantic OCR-507 Radiance configuration

#Inputs
sensor: c_ocr507R_on(sec)  0    # sets secs between how often data is sent
                        # <0 stops, 0 fast as possible, 0> that many secs

sensor: u_ocr507R_is_calibrated(bool)  0 # needs to be set in autoexec.mi

# sensor specific calibration constants (defaults for S/N 082)

sensor: u_ocr507R_dark_counts_c1(nodim) 2148739218.5 # dark offset for channel 1
sensor: u_ocr507R_cal_coeff_c1(Tnodim)  27096.112147 # calibration factor for channel 1
sensor: u_ocr507R_immersion_coeff_c1(nodim) 1.758    # immersion factor for channel 1

sensor: u_ocr507R_dark_counts_c2(nodim) 2147915422.1 # dark offset for channel 2
sensor: u_ocr507R_cal_coeff_c2(Tnodim)  27065.322575 # calibration factor for channel 2
sensor: u_ocr507R_immersion_coeff_c2(nodim) 1.754    # immersion factor for channel 2

sensor: u_ocr507R_dark_counts_c3(nodim) 2148704283.1 # dark offset for channel 3
sensor: u_ocr507R_cal_coeff_c3(Tnodim)  26930.360588 # calibration factor for channel 3
sensor: u_ocr507R_immersion_coeff_c3(nodim) 1.745    # immersion factor for channel 3

sensor: u_ocr507R_dark_counts_c4(nodim) 2148332704.3 # dark offset for channel 4
sensor: u_ocr507R_cal_coeff_c4(Tnodim)  17037.140659 # calibration factor for channel 4
```

sensor: u_ocr507R_immersion_coeff_c4(nodim) 1.741 # immersion factor for channel 4

sensor: u_ocr507R_dark_counts_c5(nodim) 2147608197.8 # dark offset for channel 5

sensor: u_ocr507R_cal_coeff_c5(Tnodim) 16287.406269 # calibration factor for channel 5

sensor: u_ocr507R_immersion_coeff_c5(nodim) 1.739 # immersion factor for channel 5

sensor: u_ocr507R_dark_counts_c6(nodim) 2146048148.6 # dark offset for channel 6

sensor: u_ocr507R_cal_coeff_c6(Tnodim) 11802.500350 # calibration factor for channel 6

sensor: u_ocr507R_immersion_coeff_c6(nodim) 1.730 # immersion factor for channel 6

sensor: u_ocr507R_dark_counts_c7(nodim) 2145662191.9 # dark offset for channel 7

sensor: u_ocr507R_cal_coeff_c7(Tnodim) 5511.536788 # calibration factor for channel 7

sensor: u_ocr507R_immersion_coeff_c7(nodim) 1.729 # immersion factor for channel 7

sensor: u_ocr507R_Vin_a0(nodim) 0.0 # polynomial coefficient to scale Vin

sensor: u_ocr507R_Vin_a1(nodim) 0.03 # polynomial coefficient to scale Vin

sensor: u_ocr507R_Va_a0(nodim) 0.0 # polynomial coefficient to scale Vin

sensor: u_ocr507R_Va_a1(nodim) 0.03 # polynomial coefficient to scale Vin

sensor: u_ocr507R_itemp_a0(nodim) -50.0 # polynomial coefficient to scale itemp

sensor: u_ocr507R_itemp_a1(nodim)   0.5 # polynomial coefficient to scale itemp

sensor: c_ocr507R_num_fields_to_send(nodim) 24

number of columns to send on each

measurement, fields to send chosen

by order in the list below

sensor: sci_ocr507R_is_installed(bool) 0 # in, t--> installed on science

#Outputs, in order of priority:

sensor: sci_ocr507R_rad1(uW/cm^2/nm) # from channel1

sensor: sci_ocr507R_rad2(uW/cm^2/nm) # from channel2

sensor: sci_ocr507R_rad3(uW/cm^2/nm) # from channel3

sensor: sci_ocr507R_rad4(uW/cm^2/nm) # from channel4

sensor: sci_ocr507R_rad5(uW/cm^2/nm) # from channel5

sensor: sci_ocr507R_rad6(uW/cm^2/nm) # from channel6

sensor: sci_ocr507R_rad7(uW/cm^2/nm) # from channel7

sensor: sci_ocr507R_itemp(Celsius) # internal temperature of instrument

sensor: sci_ocr507R_Vin(volts) # regulated input voltage

sensor: sci_ocr507R_Va(volts) # analog voltage

sensor: sci_ocr507R_fcount(nodim) # 0-255, count of frame transmitted

sensor: sci_ocr507R_channel1(nodim) # raw counts from discrete optical waveband 1

sensor: sci_ocr507R_channel2(nodim) # raw counts from discrete optical waveband 2

sensor: sci_ocr507R_channel3(nodim) # raw counts from discrete optical waveband 3

sensor: sci_ocr507R_channel4(nodim) # raw counts from discrete optical waveband 4

sensor: sci_ocr507R_channel5(nodim) # raw counts from discrete optical waveband 5

sensor: sci_ocr507R_channel6(nodim) # raw counts from discrete optical waveband 6

sensor: sci_ocr507R_channel7(nodim) # raw counts from discrete optical waveband 7

sensor: sci_ocr507R_itemp_raw(nodim) # raw pre-scaled temperature

```
sensor: sci_ocr507R_Vin_raw(nodim)   # raw pre-scaled regulated input voltage
sensor: sci_ocr507R_Va_raw(nodim)    # raw pre-scaled analog voltage
sensor: sci_ocr507R_timer(sec)       # seconds since initialization (power-on)
sensor: sci_ocr507R_delay(msec)      # milliseconds offset to timer for
                                     # accurate indication of when frame's sensors
                                     # were sampled
sensor: sci_ocr507R_cksum(nodim)     # data integrity sensor, checksum on frame


# proglet ocr507I: Satlantic OCR-507 Irradiance configuration

#Inputs
sensor: c_ocr507I_on(sec)   0   # sets secs between how often data is sent
                                # <0 stops, 0 fast as possible, 0> that many secs

sensor: u_ocr507I_is_calibrated(bool)  0 # needs to be set in autoexec.mi

# sensor specific calibration constants (defaults for S/N 152)

sensor: u_ocr507I_dark_counts_c1(nodim) 2149587489.7 # dark offset for channel 1
sensor: u_ocr507I_cal_coeff_c1(Tnodim)  2139416.2652 # calibration factor for channel 1
sensor: u_ocr507I_immersion_coeff_c1(nodim) 1.368    # immersion factor for channel 1

sensor: u_ocr507I_dark_counts_c2(nodim) 2147351752.0 # dark offset for channel 2
sensor: u_ocr507I_cal_coeff_c2(Tnodim)  1973191.3026 # calibration factor for channel 2
```

sensor: u_ocr507l_immersion_coeff_c2(nodim) 1.401 # immersion factor for channel 2

sensor: u_ocr507l_dark_counts_c3(nodim) 2148356170.6 # dark offset for channel 3

sensor: u_ocr507l_cal_coeff_c3(Tnodim) 2072416.6110 # calibration factor for channel 3

sensor: u_ocr507l_immersion_coeff_c3(nodim) 1.365 # immersion factor for channel 3

sensor: u_ocr507l_dark_counts_c4(nodim) 2147879094.8 # dark offset for channel 4

sensor: u_ocr507l_cal_coeff_c4(Tnodim) 2070368.1944 # calibration factor for channel 4

sensor: u_ocr507l_immersion_coeff_c4(nodim) 1.378 # immersion factor for channel 4

sensor: u_ocr507l_dark_counts_c5(nodim) 2147571956.1 # dark offset for channel 5

sensor: u_ocr507l_cal_coeff_c5(Tnodim) 2108980.9681 # calibration factor for channel 5

sensor: u_ocr507l_immersion_coeff_c5(nodim) 1.372 # immersion factor for channel 5

sensor: u_ocr507l_dark_counts_c6(nodim) 2147977849.9 # dark offset for channel 6

sensor: u_ocr507l_cal_coeff_c6(Tnodim) 2209709.2232 # calibration factor for channel 6

sensor: u_ocr507l_immersion_coeff_c6(nodim) 1.354 # immersion factor for channel 6

sensor: u_ocr507l_dark_counts_c7(nodim) 2147679441.1 # dark offset for channel 7

sensor: u_ocr507l_cal_coeff_c7(Tnodim) 2090347.2455 # calibration factor for channel 7

sensor: u_ocr507l_immersion_coeff_c7(nodim) 1.347 # immersion factor for channel 7

sensor: u_ocr507l_Vin_a0(nodim) 0.0 # polynomial coefficient to scale Vin

sensor: u_ocr507l_Vin_a1(nodim) 0.03 # polynomial coefficient to scale Vin

sensor: u_ocr507I_Va_a0(nodim) 0.0 # polynomial coefficient to scale Va

sensor: u_ocr507I_Va_a1(nodim) 0.03 # polynomial coefficient to scale Va

sensor: u_ocr507I_itemp_a0(nodim) -50.0 # polynomial coefficient to scale itemp

sensor: u_ocr507I_itemp_a1(nodim)   0.5 # polynomial coefficient to scale itemp

sensor: c_ocr507I_num_fields_to_send(nodim) 24

number of columns to send on each

measurement, fields to send chosen

by order in the list below

sensor: sci_ocr507I_is_installed(bool) 0 # in, t--> installed on science

#Outputs, in order of priority:

sensor: sci_ocr507I_irrad1(uW/cm^2/nm) # from channel1

sensor: sci_ocr507I_irrad2(uW/cm^2/nm) # from channel2

sensor: sci_ocr507I_irrad3(uW/cm^2/nm) # from channel3

sensor: sci_ocr507I_irrad4(uW/cm^2/nm) # from channel4

sensor: sci_ocr507I_irrad5(uW/cm^2/nm) # from channel5

sensor: sci_ocr507I_irrad6(uW/cm^2/nm) # from channel6

sensor: sci_ocr507I_irrad7(uW/cm^2/nm) # from channel7

sensor: sci_ocr507I_itemp(Celsius)   # internal temperature of instrument

sensor: sci_ocr507I_Vin(volts)   # regulated input voltage

sensor: sci_ocr507I_Va(volts)   # analog voltag

sensor: sci_ocr507I_fcount(nodim)   # 0-255, count of frame transmitted

```
sensor: sci_ocr507I_channel1(nodim)    # raw counts from discrete optical waveband 1

sensor: sci_ocr507I_channel2(nodim)    # raw counts from discrete optical waveband 2

sensor: sci_ocr507I_channel3(nodim)    # raw counts from discrete optical waveband 3

sensor: sci_ocr507I_channel4(nodim)    # raw counts from discrete optical waveband 4

sensor: sci_ocr507I_channel5(nodim)    # raw counts from discrete optical waveband 5

sensor: sci_ocr507I_channel6(nodim)    # raw counts from discrete optical waveband 6

sensor: sci_ocr507I_channel7(nodim)    # raw counts from discrete optical waveband 7

sensor: sci_ocr507I_itemp_raw(nodim)   # raw pre-scaled temperature

sensor: sci_ocr507I_Vin_raw(nodim)     # raw pre-scaled regulated input voltage

sensor: sci_ocr507I_Va_raw(nodim)      # raw pre-scaled analog voltage

sensor: sci_ocr507I_timer(sec)         # seconds since initialization (power-on)

sensor: sci_ocr507I_delay(msec)        # milliseconds offset to timer for

                                       # accurate indication of when frame's sensors

                                       # were sampled

sensor: sci_ocr507I_cksum(nodim)       # data integrity sensor, checksum on frame



      # sensors for Benthos Acoustic Data Delivery (badd) proglet


#Inputs:


sensor: c_badd_on(sec)                 -1 # secs between run cycles

sensor: c_badd_mode(enum)              0 # 0: search mode

                                       # 1: data collect mode

sensor: c_badd_target_id(enum)         -1 # address of remote host modem being called
```

```
sensor: c_badd_range_secs(sec)          60 # how often to request range to remote mode
                                        # <0 => don't request range,
                                        # min value = c_badd_input_parse_secs(sec) * 2
sensor: c_badd_input_parse_secs(sec)    30 # How long to check command response
                                        # input buffer
sensor: c_badd_datacol_status_secs(sec) 30 # How often to check download status
sensor: c_badd_clear_remote_data(bool)   0 # 0: do NOT clear remote data after successful
sensor: c_badd_autobaud(bool)            0 # 1: Perform autobaud, 0: Don't perform autobaud
sensor: c_baud_attempt_min(enum)         3 # minimum badd baud attempt
sensor: c_baud_attempt_max(enum)         8 # maximum badd baud attempt
                                        # Enum for min/max baud
                                        #   0: MODSPEC_NULL
                                        #   1: MODSPEC_80_MFSK
                                        #   2: MODSPEC_140_MFSK
                                        #   3: MODSPEC_300_MFSK
                                        #   4: MODSPEC_600_MFSK
                                        #   5: MODSPEC_800_MFSK
                                        #   6: MODSPEC_1066_MFSK
                                        #   7: MODSPEC_1200_MFSK
                                        #   8: MODSPEC_2400_MFSK
                                        #   9: MODSPEC_2560_PSK
                                        #  10: MODSPEC_5120_PSK
                                        #  11: MODSPEC_7680_PSK
                                        #  12: MODSPEC_10240_PSK
                                        #  13: MODSPEC_15360_PSK
```

64: MODSPEC_80_FH

sensor: c_autobaud_max_ber(nodim) 0 # max BER allowed

sensor: c_badd_transaction_num(nodim) 0 # ID of the transaction to perform at the mooring.(8 digit max)

sensor: c_badd_channel_probe_test(bool) 0 # Perform Channel Probe Test with Remote Node. 0:NO, 1:YES

#Outputs:

sensor: sci_badd_mmp_is_installed(bool) 0 # true -> MMP version of proglet is installed.

sensor: sci_badd_is_installed(bool) 0 # true -> proglet is installed

sensor: sci_badd_power_on(bool) 0 # power state of modem (true -> on)

sensor: sci_badd_error(nodim) 0 # unique number for each error type

sensor: sci_badd_remote_stored_bytes(nodim) 0 # number of stored bytes on remote modem

sensor: sci_badd_retrieved_bytes(nodim) 0 # number of bytes collected from remote modem

sensor: sci_badd_n_tries_to_connect(nodim) 0 # number of attempts to connect with target modem

sensor: sci_badd_target_range(m) 0 # response to range command

sensor: sci_badd_finished(bool) 0 # the proglet has finished

proglet flntu: wet labs flntu fluorometer and turbidity sensor

sensor: c_flntu_on(sec) 2 # in, sets secs between measurements

<0 stops, 0 fast as possible, >0 that many secs

sensor: sci_flntu_is_installed(bool) 0 # in, t--> installed on science

sensor: c_flntu_num_fields_to_send(nodim) 7 # in, number of columns to send on each

measurement, fields to send chosen

by order in the list below

sensor: u_flntu_is_calibrated(bool) 0 # false, assume not calibrated

sensor specific input calibration constants (defaults for FLNTUSLO-513)

sensor: u_flntu_chlor_do(nodim) 39 # dark water offset, nodim == counts

sensor: u_flntu_turb_do(nodim) 47 # dark water offset, nodim == counts

sensor: u_flntu_chlor_sf(ug/l/nodim) 0.0125 # scale factor to get units

sensor: u_flntu_turb_sf(NTU/nodim) 0.0062 # scale factor to get units

output sensors, listed in PRIORITY order

e.g. if c_flntu_num_fields_to_send is 2, cols derived

from 4,6 sent

sensor: sci_flntu_chlor_units(ug/l) 0 # derived from col 4

sensor: sci_flntu_turb_units(NTU) 0 # derived from col 6

sensor: sci_flntu_chlor_sig(nodim) 0 # col 4

sensor: sci_flntu_turb_sig(nodim) 0 # col 6

sensor: sci_flntu_chlor_ref(nodim) 0 # col 3

sensor: sci_flntu_turb_ref(nodim) 0 # col 5

sensor: sci_flntu_temp(nodim) 0 # col 7

sensor: sci_flntu_timestamp(timestamp) 0 # secs since 1970

```
# proglet fl3sloV2: wet labs fl3slo fluorometer triplet sensor, 2nd configuration
sensor: c_fl3sloV2_on(sec)       2 # in, sets secs between measurements
                                   # <0 stops, 0 fast as possible, >0 that many secs
sensor: sci_fl3sloV2_is_installed(bool) 0 # in, t--> installed on science

sensor: c_fl3sloV2_num_fields_to_send(nodim) 10 # in, number of columns to send on each
                                   # measurement, fields to send chosen
                                   # by order in the list below

sensor: u_fl3sloV2_is_calibrated(bool) 0 # false, assume not calibrated

# sensor specific input calibration constants (defaults for FL3SLO-496)
sensor: u_fl3sloV2_chlor_cwo(nodim)      46 # clean water offset, nodim == counts
sensor: u_fl3sloV2_rhod_cwo(nodim)       49 # clean water offset, nodim == counts
sensor: u_fl3sloV2_cdom_cwo(nodim)       48 # clean water offset, nodim == counts
sensor: u_fl3sloV2_chlor_sf(ug/l/nodim)  0.0127 # scale factor to get units
sensor: u_fl3sloV2_rhod_sf(ppb/nodim)    0.0481 # scale factor to get units
sensor: u_fl3sloV2_cdom_sf(ppb/nodim)    0.0961 # scale factor to get units

        # output sensors, listed in PRIORITY order
        # e.g. if c_fl3sloV2_num_fields_to_send is 3, cols derived
        # from 4,6,8 sent
sensor: sci_fl3sloV2_chlor_units(ug/l)    0 # derived from col 4
sensor: sci_fl3sloV2_rhod_units(ppb)      0 # derived from col 6
```

sensor: sci_fl3sloV2_cdom_units(ppb) 0 # derived from col 8

sensor: sci_fl3sloV2_chlor_sig(nodim) 0 # col 4

sensor: sci_fl3sloV2_rhod_sig(nodim) 0 # col 6

sensor: sci_fl3sloV2_cdom_sig(nodim) 0 # col 8

sensor: sci_fl3sloV2_chlor_ref(nodim) 0 # col 3

sensor: sci_fl3sloV2_rhod_ref(nodim) 0 # col 5

sensor: sci_fl3sloV2_cdom_ref(nodim) 0 # col 7

sensor: sci_fl3sloV2_temp(nodim) 0 # col 9

sensor: sci_fl3sloV2_timestamp(timestamp) 0 # secs since 1970

proglet bb3sloV2: wet labs bb3slo backscatter triplet sensor, 2nd configuration

sensor: c_bb3sloV2_on(sec) 2 # in, sets secs between measurements

<0 stops, 0 fast as possible, >0 that many secs

sensor: sci_bb3sloV2_is_installed(bool) 0 # in, t--> installed on science

sensor: c_bb3sloV2_num_fields_to_send(nodim) 10 # in, number of columns to send on each

measurement, fields to send chosen

by order in the list below

sensor: u_bb3sloV2_is_calibrated(bool) 0 # false, assume not calibrated

sensor specific input calibration constants (defaults for BB3SLO-286)

sensor: u_bb3sloV2_b532_do(nodim) 44 # dark offset, nodim == counts

sensor: u_bb3sloV2_b660_do(nodim) 49 # dark offset, nodim == counts

sensor: u_bb3sloV2_b880_do(nodim)   52   # dark offset, nodim == counts

sensor: u_bb3sloV2_b532_sf(Mnodim)   8.42 # scale factor (0.00000842)

sensor: u_bb3sloV2_b660_sf(Mnodim)   4.16 # scale factor (0.00000416)

sensor: u_bb3sloV2_b880_sf(Mnodim)   3.27 # scale factor (0.00000327)

output sensors, listed in PRIORITY order

e.g. if c_bb3sloV2_num_fields_to_send is 3, cols derived from 4,6,8 sent

sensor: sci_bb3sloV2_b532_scaled(nodim)   0 # from col 4

sensor: sci_bb3sloV2_b660_scaled(nodim)   0 # from col 6

sensor: sci_bb3sloV2_b880_scaled(nodim)   0 # from col 8

sensor: sci_bb3sloV2_b532_sig(nodim)   0 # col 4

sensor: sci_bb3sloV2_b660_sig(nodim)   0 # col 6

sensor: sci_bb3sloV2_b880_sig(nodim)   0 # col 8

sensor: sci_bb3sloV2_b532_ref(nodim)   0 # col 3

sensor: sci_bb3sloV2_b660_ref(nodim)   0 # col 5

sensor: sci_bb3sloV2_b880_ref(nodim)   0 # col 7

sensor: sci_bb3sloV2_temp(nodim)   0 # col 9

sensor: sci_bb3sloV2_timestamp(timestamp) 0 # secs since 1970

proglet bb3sloV3: wet labs bb3slo backscatter triplet sensor, 3nd configuration

sensor: c_bb3sloV3_on(sec)   2 # in, sets secs between measurements

                                  # <0 stops, 0 fast as possible, >0 that many secs

sensor: sci_bb3sloV3_is_installed(bool) 0 # in, t--> installed on science

sensor: c_bb3sloV3_num_fields_to_send(nodim) 10 # in, number of columns to send on each

measurement, fields to send chosen

by order in the list below

sensor: u_bb3sloV3_is_calibrated(bool) 0 # false, assume not calibrated

sensor specific input calibration constants (defaults for BB3SLO-300)

sensor: u_bb3sloV3_b532_do(nodim) 20 # dark offset, nodim == counts

sensor: u_bb3sloV3_b630_do(nodim) 11 # dark offset, nodim == counts

sensor: u_bb3sloV3_b880_do(nodim) 18 # dark offset, nodim == counts

sensor: u_bb3sloV3_b532_sf(Mnodim) 7.093 # scale factor (0.000007093)

sensor: u_bb3sloV3_b630_sf(Mnodim) 3.888 # scale factor (0.000003888)

sensor: u_bb3sloV3_b880_sf(Mnodim) 2.370 # scale factor (0.000002370)

output sensors, listed in PRIORITY order

e.g. if c_bb3sloV3_num_fields_to_send is 3, cols derived from 4,6,8 sent

sensor: sci_bb3sloV3_b532_scaled(nodim) 0 # from col 4

sensor: sci_bb3sloV3_b630_scaled(nodim) 0 # from col 6

sensor: sci_bb3sloV3_b880_scaled(nodim) 0 # from col 8

sensor: sci_bb3sloV3_b532_sig(nodim) 0 # col 4

sensor: sci_bb3sloV3_b630_sig(nodim) 0 # col 6

sensor: sci_bb3sloV3_b880_sig(nodim) 0 # col 8

sensor: sci_bb3sloV3_b532_ref(nodim) 0 # col 3

sensor: sci_bb3sloV3_b630_ref(nodim) 0 # col 5

sensor: sci_bb3sloV3_b880_ref(nodim) 0 # col 7

sensor: sci_bb3sloV3_temp(nodim) 0 # col 9

sensor: sci_bb3sloV3_timestamp(timestamp) 0 # secs since 1970

simulator proglet wetlabs_sim: generic wet labs sensor simulator

sensor: sci_wetlabs_sim_is_installed(bool) 0 # in, t--> wetlabs sensor is being simulated on science computer

sensor: u_wetlabs_sim_num_eng_units(nodim) 3 # currently, either 2 or 3

proglet bb2fls: wet labs bb2flslk scatter meter and fluorometer sensor

sensor: c_bb2fls_on(sec) 2 # in, sets secs between measurements

<0 stops, 0 fast as possible, >0 that many secs

sensor: sci_bb2fls_is_installed(bool) 0 # in, t--> installed on science

sensor: c_bb2fls_num_fields_to_send(nodim) 10 # in, number of columns to send on each

measurement, fields to send chosen

by order in the list below

sensor: u_bb2fls_is_calibrated(bool) 0 # false, assume not calibrated

sensor specific input calibration constants (defaults for BB2FLSLK-295)

sensor: u_bb2fls_b660_cwo(nodim) 38 # clean water offset, nodim == counts

sensor: u_bb2fls_b880_cwo(nodim) 48 # clean water offset, nodim == counts

sensor: u_bb2fls_cdom_cwo(nodim) 45 # clean water offset, nodim == counts

sensor: u_bb2fls_b660_sf(Mnodim) 3.298 # scale factor (0.000003298)

sensor: u_bb2fls_b880_sf(Mnodim) 3.079 # scale factor (0.000003079)

sensor: u_bb2fls_cdom_sf(ppb/nodim) 0.1695 # scale factor to get units

output sensors, listed in PRIORITY order

e.g. if c_bb2fls_num_fields_to_send is 3, cols derived

from 4,6,8 sent

sensor: sci_bb2fls_b660_scaled(nodim) 0 # derived from col 4

sensor: sci_bb2fls_b880_scaled(nodim) 0 # derived from col 6

sensor: sci_bb2fls_cdom_scaled(ppb) 0 # derived from col 8

sensor: sci_bb2fls_b660_sig(nodim) 0 # col 4

sensor: sci_bb2fls_b880_sig(nodim) 0 # col 6

sensor: sci_bb2fls_cdom_sig(nodim) 0 # col 8

sensor: sci_bb2fls_b660_ref(nodim) 0 # col 3

sensor: sci_bb2fls_b880_ref(nodim) 0 # col 5

sensor: sci_bb2fls_cdom_ref(nodim) 0 # col 7

sensor: sci_bb2fls_therm(nodim) 0 # col 9

sensor: sci_bb2fls_timestamp(timestamp) 0 # secs since 1970

proglet bb2flsV2: wet labs bb2flslk scatter meter and fluorometer sensor, 2nd configuration

sensor: c_bb2flsV2_on(sec) 2 # in, sets secs between measurements

<0 stops, 0 fast as possible, >0 that many secs

sensor: sci_bb2flsV2_is_installed(bool) 0 # in, t--> installed on science

sensor: c_bb2flsV2_num_fields_to_send(nodim) 10 # in, number of columns to send on each

measurement, fields to send chosen

by order in the list below

sensor: u_bb2flsV2_is_calibrated(bool) 0 # false, assume not calibrated

sensor specific input calibration constants (defaults for BB2FLSLK-296)

sensor: u_bb2flsV2_b470_cwo(nodim) 51 # clean water offset, nodim == counts

sensor: u_bb2flsV2_b532_cwo(nodim) 50 # clean water offset, nodim == counts

sensor: u_bb2flsV2_chl_cwo(nodim) 51 # clean water offset, nodim == counts

sensor: u_bb2flsV2_b470_sf(Mnodim) 11.67 # scale factor (0.00001167)

sensor: u_bb2flsV2_b532_sf(Mnodim) 3.079 # scale factor (0.000003079)

sensor: u_bb2flsV2_chl_sf(ug/l/nodim) 0.0133 # scale factor to get units

output sensors, listed in PRIORITY order

e.g. if c_bb2flsV2_num_fields_to_send is 3, cols derived

from 4,6,8 sent

sensor: sci_bb2flsV2_b470_scaled(nodim) 0 # derived from col 4

sensor: sci_bb2flsV2_b532_scaled(nodim) 0 # derived from col 6

sensor: sci_bb2flsV2_chl_scaled(ug/l) 0 # derived from col 8

sensor: sci_bb2flsV2_b470_sig(nodim) 0 # col 4

sensor: sci_bb2flsV2_b532_sig(nodim) 0 # col 6

sensor: sci_bb2flsV2_chl_sig(nodim) 0 # col 8

sensor: sci_bb2flsV2_b470_ref(nodim) 0 # col 3

sensor: sci_bb2flsV2_b532_ref(nodim) 0 # col 5

sensor: sci_bb2flsV2_chl_ref(nodim) 0 # col 7

sensor: sci_bb2flsV2_therm(nodim) 0 # col 9

sensor: sci_bb2flsV2_timestamp(timestamp) 0 # secs since 1970

simulator proglet auvb_sim: Wet Labs AUV-B Fluorometer simulator

sensor: sci_auvb_sim_is_installed(bool) 0 # in, t--> auvb is being simulated on science computer

proglet auvb: wet labs auv-b ECO Fluorometer

sensor: c_auvb_on(sec) 2 # in, sets secs between measurements

<0 stops, 0 fast as possible, >0 that many secs

sensor: sci_auvb_is_installed(bool) 0 # in, t--> installed on science

sensor: c_auvb_num_fields_to_send(nodim) 3 # in, number of columns to send on

each measurement, fields to send

chosen by order in the list below

output sensors, listed in PRIORITY order

sensor: sci_auvb_ref(nodim) 0 # col 3, refernece counts

sensor: sci_auvb_sig(nodim) 0 # col 4, signal counts

sensor: sci_auvb_therm(nodim) 0 # col 5, internal thermistor

sensor: sci_auvb_timestamp(timestamp) 0 # secs since 1970

proglet bb2fV2: wet labs bb2fslo scatter meter and fluorometer sensor

sensor: c_bb2fV2_on(sec) 2 # in, sets secs between measurements

<0 stops, 0 fast as possible, >0 that many secs

sensor: sci_bb2fV2_is_installed(bool) 0 # in, t--> installed on science

sensor: c_bb2fV2_num_fields_to_send(nodim) 9 # in, number of columns to send on each

measurement, fields to send chosen

by order in the list below

A negative value signifies

to use this value as a bitmap.

The user may specify any

outputs regardless of order by

by using this sensor as a bitmap.

-1 * (2^(f1-1)+2^(f2-1)+...)

where fn is the field number.

For example, if the user wished

to just record sci_bb2fV2_b700_scaled

they would use:

-1 * 2^(2-1) = -2

sensor: u_bb2fV2_is_calibrated(bool) 0 # false, assume not calibrated

sensor specific input calibration constants (defaults for BB2FSLO-341)

sensor: u_bb2fV2_b470_cwo(nodim) 54 # clean water offset, nodim == counts

sensor: u_bb2fV2_b700_cwo(nodim) 58 # clean water offset, nodim == counts

sensor: u_bb2fV2_chlor_cwo(nodim) 53 # clean water offset, nodim == counts

sensor: u_bb2fV2_b470_sf(Mnodim) 22.09 # scale factor (0.00002209)

sensor: u_bb2fV2_b700_sf(Mnodim) 1.45 # scale factor (0.00000145)

sensor: u_bb2fV2_chlor_sf(ug/l/nodim) 0.0152 # scale factor to get units

output sensors, listed in PRIORITY order

e.g. if c_bb2fV2_num_fields_to_send is 3, cols derived

from 4,6,7 sent

sensor: sci_bb2fV2_b470_scaled(nodim) 0 # derived from col 4

sensor: sci_bb2fV2_b700_scaled(nodim) 0 # derived from col 6

sensor: sci_bb2fV2_chlor_scaled(ug/l) 0 # derived from col 7

sensor: sci_bb2fV2_b470_sig(nodim) 0 # col 4

sensor: sci_bb2fV2_b700_sig(nodim) 0 # col 6

sensor: sci_bb2fV2_chlor(nodim) 0 # col 7

sensor: sci_bb2fV2_b470_ref(nodim) 0 # col 3

sensor: sci_bb2fV2_b700_ref(nodim) 0 # col 5

sensor: sci_bb2fV2_therm(nodim) 0 # col 9

sensor: sci_bb2fV2_timestamp(timestamp) 0 # secs since 1970

proglet tarr: OASIS Towed Array Receiver / DSP

#inputs:

sensor: c_tarr_on(sec) 0 # in, sets secs between measurements

<0 stops, 0 fast as possible, >0 that many secs

sensor: u_tarr_num_errors_before_restart(nodim) 5 # number of errors before cycling

power, <0 = never cycle power

0 = restart on any error

sensor: u_tarr_dsp_power_on_delay(sec) 75.0 # wait time between tarr and dsp power on

#outputs:

sensor: sci_tarr_is_installed(bool) 0 # in, t--> installed on science

sensor: sci_tarr_track_count(nodim) 0 # number of data tracks produced since power on

sensor: sci_tarr_error(nodim) 0 # unique number to indicate error type

proglet glbps: ASL GLBPS SONAR Device

#sensor: c_glbps_on(sec) 1 # in, sets secs between measurements

<0 stops, 0 fast as possible, >0 that many secs

#sensor: sci_glbps_is_installed(bool) 0 # in, t--> installed on science

#sensor: c_glbps_num_fields_to_send(nodim) 3 # in, number of columns to send on each

measurement, fields to send chosen

by order in the list below

output sensors, listed in PRIORITY order

e.g. if c_glbps_num_fields_to_send is 12, cols 10,11,timestamp,1,2,3,4,5,6,7,8,9 sent

#sensor: sci_glbps_round_trip_time(nodim) 0 # col 10, round trip time

#sensor: sci_glbps_persistance(nodim) 0 # col 11, persistance

#sensor: sci_glbps_timestamp(timestamp) 0 # secs since 1970

#sensor: sci_glbps_ping_number(nodim) 0 # col 1, ping number

#sensor: sci_glbps_year(nodim) 0 # col 2, year

#sensor: sci_glbps_month(nodim) 0 # col 3, month

#sensor: sci_glbps_day(nodim) 0 # col 4, day

#sensor: sci_glbps_hour(nodim) 0 # col 5, hour

#sensor: sci_glbps_minute(nodim) 0 # col 6, minute

#sensor: sci_glbps_second(nodim) 0 # col 7, second

#sensor: sci_glbps_hundreds_of_second(nodim) 0 # col 8, hundreds of second

#sensor: sci_glbps_target_count(nodim) 0 # col 9, target count

#SPAWAR Acoustic Array Proglet

sensor: c_sscsd_on(sec) 2.0 #

sensor: sci_sscsd_is_installed(bool) 0 # in, t--> installed on science

sensor: sci_sscsd_test(nodim) 0 # this is only a test

#output sensors:

sensor: sci_wants_turn(enum) 0 # 0 no request yet

1 request

sensor: sci_wants_wpt(enum) 0

sensor: sci_heading(rad) 0 # heading sci wants to turn to

sensor: sci_wpt_x(m) -7032.0610 # The waypoint (east or lon)

sensor: sci_wpt_y(m) 4137.9980 # (north or lat)

sensor: sci_wpt_units(enum) 2 # 0 LMC, 1 UTM, 2 LAT/LONG

sensor: sci_wants_depth(enum) 0 # science request to change depth profile

sensor: sci_depth(m) 0 # depth to change to

sensor: sci_array_heading1(deg) 0

sensor: sci_array_pitch1(deg) 0

sensor: sci_array_roll1(deg) 0

sensor: sci_array_heading2(deg) 0

sensor: sci_array_pitch2(deg) 0

sensor: sci_array_roll2(deg) 0

sensor: sci_array_heading3(deg) 0

sensor: sci_array_pitch3(deg) 0

sensor: sci_array_roll3(deg) 0

proglet bb2flsV3: wet labs bb2flslk scatter meter and fluorometer sensor, 3rd configuration

sensor: c_bb2flsV3_on(sec) 2 # in, sets secs between measurements

<0 stops, 0 fast as possible, >0 that many secs

sensor: sci_bb2flsV3_is_installed(bool) 0 # in, t--> installed on science

sensor: c_bb2flsV3_num_fields_to_send(nodim) 10 # in, number of columns to send on each

measurement, fields to send chosen

by order in the list below

sensor: u_bb2flsV3_is_calibrated(bool) 0 # false, assume not calibrated

sensor specific input calibration constants (defaults for BB2FLSLK-296)

sensor: u_bb2flsV3_b715_cwo(nodim) 55 # clean water offset, nodim == counts

sensor: u_bb2flsV3_b880_cwo(nodim) 51 # clean water offset, nodim == counts

sensor: u_bb2flsV3_pe_cwo(nodim) 51 # clean water offset, nodim == counts

sensor: u_bb2flsV3_b715_sf(Mnodim) 3.62 # scale factor x 1e-6

sensor: u_bb2flsV3_b880_sf(Mnodim) 2.97 # scale factor x 1e-6

sensor: u_bb2flsV3_pe_sf(ppb/nodim) 0.0432 # scale factor to get units

output sensors, listed in PRIORITY order

e.g. if c_bb2flsV3_num_fields_to_send is 3, cols derived

from 4,6,8 sent

sensor: sci_bb2flsV3_b715_scaled(nodim) 0 # derived from col 4

sensor: sci_bb2flsV3_b880_scaled(nodim) 0 # derived from col 6

sensor: sci_bb2flsV3_pe_scaled(ppb) 0 # derived from col 8

sensor: sci_bb2flsV3_b715_sig(nodim) 0 # col 4

sensor: sci_bb2flsV3_b880_sig(nodim) 0 # col 6

sensor: sci_bb2flsV3_pe_sig(nodim) 0 # col 8

sensor: sci_bb2flsV3_b715_ref(nodim) 0 # col 3

sensor: sci_bb2flsV3_b880_ref(nodim) 0 # col 5

sensor: sci_bb2flsV3_pe_ref(nodim) 0 # col 7

sensor: sci_bb2flsV3_therm(nodim) 0 # col 9

sensor: sci_bb2flsV3_timestamp(timestamp) 0 # secs since 1970

proglet bb2flsV4: wet labs bb2flslk scatter meter and fluorometer sensor, 4th configuration

sensor: c_bb2flsV4_on(sec) 2 # in, sets secs between measurements

<0 stops, 0 fast as possible, >0 that many secs

sensor: sci_bb2flsV4_is_installed(bool) 0 # in, t--> installed on science

sensor: c_bb2flsV4_num_fields_to_send(nodim) 10 # in, number of columns to send on each

measurement, fields to send chosen

by order in the list below

sensor: u_bb2flsV4_is_calibrated(bool) 0 # false, assume not calibrated

sensor specific input calibration constants (defaults for BB2FLSLK-507)

sensor: u_bb2flsV4_b412_cwo(nodim) 51 # clean water offset, nodim == counts

sensor: u_bb2flsV4_b470_cwo(nodim) 48 # clean water offset, nodim == counts

sensor: u_bb2flsV4_chl_cwo(nodim) 54 # clean water offset, nodim == counts

sensor: u_bb2flsV4_b412_sf(Mnodim) 13.27 # scale factor x 1e-6

sensor: u_bb2flsV4_b470_sf(Mnodim) 12.08 # scale factor x 1e-6

sensor: u_bb2flsV4_chl_sf(ug/l/nodim) 0.0118 # scale factor to get units

output sensors, listed in PRIORITY order

e.g. if c_bb2flsV4_num_fields_to_send is 3, cols derived

from 4,6,8 sent

sensor: sci_bb2flsV4_b412_scaled(nodim) 0 # derived from col 4

sensor: sci_bb2flsV4_b470_scaled(nodim) 0 # derived from col 6

sensor: sci_bb2flsV4_chl_scaled(ug/l) 0 # derived from col 8

sensor: sci_bb2flsV4_b412_sig(nodim) 0 # col 4

sensor: sci_bb2flsV4_b470_sig(nodim) 0 # col 6

sensor: sci_bb2flsV4_chl_sig(nodim) 0 # col 8

sensor: sci_bb2flsV4_b412_ref(nodim) 0 # col 3

sensor: sci_bb2flsV4_b470_ref(nodim) 0 # col 5

sensor: sci_bb2flsV4_chl_ref(nodim) 0 # col 7

sensor: sci_bb2flsV4_therm(nodim) 0 # col 9

sensor: sci_bb2flsV4_timestamp(timestamp) 0 # secs since 1970

proglet bb2flsV5: wet labs bb2flslk scatter meter and fluorometer sensor, 5th configuration

sensor: c_bb2flsV5_on(sec) 2 # in, sets secs between measurements

<0 stops, 0 fast as possible, >0 that many secs

sensor: sci_bb2flsV5_is_installed(bool) 0 # in, t--> installed on science

sensor: c_bb2flsV5_num_fields_to_send(nodim) 10 # in, number of columns to send on each

measurement, fields to send chosen

by order in the list below

sensor: u_bb2flsV5_is_calibrated(bool) 0 # false, assume not calibrated

sensor specific input calibration constants (defaults for BB2FLSLK-506)

sensor: u_bb2flsV5_b532_cwo(nodim) 52 # clean water offset, nodim == counts

sensor: u_bb2flsV5_b660_cwo(nodim) 59 # clean water offset, nodim == counts

sensor: u_bb2flsV5_cdom_cwo(nodim) 63 # clean water offset, nodim == counts

sensor: u_bb2flsV5_b532_sf(Mnodim) 7.678 # scale factor x 1e-6

sensor: u_bb2flsV5_b660_sf(Mnodim) 3.829 # scale factor x 1e-6

sensor: u_bb2flsV5_cdom_sf(ppb/nodim) 0.0959 # scale factor to get units

output sensors, listed in PRIORITY order

e.g. if c_bb2flsV5_num_fields_to_send is 3, cols derived

from 4,6,8 sent

sensor: sci_bb2flsV5_b532_scaled(nodim) 0 # derived from col 4

sensor: sci_bb2flsV5_b660_scaled(nodim) 0 # derived from col 6

sensor: sci_bb2flsV5_cdom_scaled(ppb) 0 # derived from col 8

sensor: sci_bb2flsV5_b532_sig(nodim) 0 # col 4

sensor: sci_bb2flsV5_b660_sig(nodim) 0 # col 6

sensor: sci_bb2flsV5_cdom_sig(nodim) 0 # col 8

sensor: sci_bb2flsV5_b532_ref(nodim) 0 # col 3

sensor: sci_bb2flsV5_b660_ref(nodim) 0 # col 5

sensor: sci_bb2flsV5_cdom_ref(nodim) 0 # col 7

sensor: sci_bb2flsV5_therm(nodim) 0 # col 9

sensor: sci_bb2flsV5_timestamp(timestamp) 0 # secs since 1970

proglet bb2flsV6: wet labs bb2flslk scatter meter and fluorometer sensor, 3rd configuration

sensor: c_bb2flsV6_on(sec) 2 # in, sets secs between measurements

<0 stops, 0 fast as possible, >0 that many secs

sensor: sci_bb2flsV6_is_installed(bool) 0 # in, t--> installed on science

sensor: c_bb2flsV6_num_fields_to_send(nodim) 10 # in, number of columns to send on each

measurement, fields to send chosen

by order in the list below

sensor: u_bb2flsV6_is_calibrated(bool) 0 # false, assume not calibrated

sensor specific input calibration constants (defaults for BB2FLSLK-687)

sensor: u_bb2flsV6_b532_cwo(nodim) 53 # clean water offset, nodim == counts

sensor: u_bb2flsV6_b880_cwo(nodim) 51 # clean water offset, nodim == counts

sensor: u_bb2flsV6_cdom_cwo(nodim) 42 # clean water offset, nodim == counts

sensor: u_bb2flsV6_b532_sf(Mnodim) 7.689 # scale factor (0.00001167)

sensor: u_bb2flsV6_b880_sf(Mnodim) 2.471 # scale factor (0.000003079)

sensor: u_bb2flsV6_cdom_sf(ppb/nodim) 0.0905 # scale factor to get units

output sensors, listed in PRIORITY order

e.g. if c_bb2flsV6_num_fields_to_send is 3, cols derived

from 4,6,8 sent

sensor: sci_bb2flsV6_b532_scaled(nodim) 0 # derived from col 4

sensor: sci_bb2flsV6_b880_scaled(nodim) 0 # derived from col 6

sensor: sci_bb2flsV6_cdom_scaled(ppb) 0 # derived from col 8

sensor: sci_bb2flsV6_b532_sig(nodim) 0 # col 4

sensor: sci_bb2flsV6_b880_sig(nodim) 0 # col 6

sensor: sci_bb2flsV6_cdom_sig(nodim) 0 # col 8

sensor: sci_bb2flsV6_b532_ref(nodim) 0 # col 3

sensor: sci_bb2flsV6_b880_ref(nodim) 0 # col 5

sensor: sci_bb2flsV6_cdom_ref(nodim) 0 # col 7

sensor: sci_bb2flsV6_therm(nodim) 0 # col 9

sensor: sci_bb2flsV6_timestamp(timestamp) 0 # secs since 1970

```
# proglet FIRe: Satlantic Fluorescence Induction and Relaxation electronics


# input sensors
sensor: c_FIRe_on(sec)                  0  #in, >=0 turns it on, <0 stops it
sensor: c_FIRe_num_fields_to_send(nodim)  10 #in, number of columns to send
sensor: u_FIRe_num_errors_before_restart(nodim) 5 # number of errors before
                                        # cycling power,
                                        # <0 = never cycle power


sensor: sci_FIRe_is_installed(bool)      0  # in, t--> installed on science


# output sensors
sensor: sci_FIRe_timestamp(timestamp) 0 # measurement timestamp
sensor: sci_FIRe_Fo(nodim)           0 # Calculated initial fluorescence
sensor: sci_FIRe_Fm(nodim)           0 # Calculated maximum fluorescence
sensor: sci_FIRe_FvFm(nodim)         0 # Calculated maximum quantum yield of
                                  # photochemistry in PSII
sensor: sci_FIRe_s(nodim)            0 # Calculated Sigma-PSII
sensor: sci_FIRe_p(nodim)            0 # Calculated connectivity factor
sensor: sci_FIRe_par(nodim)          0 # Calculated PAR
sensor: sci_FIRe_battery(volts)      0 # Battery volts measured by FIRe
sensor: sci_FIRe_temp(degC)          0 # FIRe PCB temp
sensor: sci_FIRe_frame_count(nodim)   0 # what it says
```

sensor: sci_FIRe_error(nodim) 0 # unique number to indicate error type

proglet ohf: Oasis High Frequency hydrophone

#sensor: c_ohf_on(sec) 0 # in, 0 = on, -1 = off

#sensor: sci_ohf_is_installed(bool) 0 # in, t--> installed on science

#sensor: sci_ohf_status(enum) 0 # out

proglet logger: generic data logger on/off control

sensor: c_logger_on(sec) 0 # in, 0 = on, -1 = off

sensor: sci_logger_is_installed(bool) 0 # in, t--> installed on science

sensor: sci_logger_status(enum) 0 # out

sensor: c_logger_ctrl_timeout(sec) -1 # in, -1 --> disable, i.e. logger ctrl bit is never lowered.

>=0 --> num of seconds to pull logger ctrl bit low

before shutting power to instrument.

simulator proglet bbam_sim: Wet Labs BAM beam attenuation meter simulator

sensor: sci_bbam_sim_is_installed(bool) 0 # in, t--> bbam is being simulated on science computer

proglet bbam: Wetlabs BAM beam attenuation meter

sensor: c_bbam_on(sec) 0 # in, 0 = on, -1 = off

sensor: c_bbam_num_fields_to_send(nodim) 6 # in, number of columns to send on each

sensor: sci_bbam_is_installed(bool) 0 # in, t--> installed on science

sensor: sci_bbam_beam_c(1/m) 0 # out, beam C

```
sensor: sci_bbam_corr_sig(nodim)        0 # out, corrected signal value

sensor: sci_bbam_raw_sig(nodim)         0 # out, raw signal value

sensor: sci_bbam_raw_ref(nodim)         0 # out, raw reference value

sensor: sci_bbam_therm(nodim)           0 # out, thermistor

sensor: sci_bbam_timestamp(timestamp)    0 # secs since 1970


# proglet uModem: W.H.O.I acoustic micro-modem


#inputs:


sensor: c_uModem_on(sec)        0 # in, sets secs between measurements

                        # <0 stops, 0 fast as possible, >0 that many secs

                        #


sensor: u_uModem_hes_secs(sec) 300.0  # how often to transmit HES messages

                           # <0 => don't transmit HES messages


sensor: u_uModem_num_errors_before_restart(nodim)  5 # number of errors before cycling

                                    # power, <0 = never cycle power

                                    # 0 = restart on any error


sensor: u_uModem_SRC(nodim)  1   # SRC: [0-15] address of uModem on glider


sensor: u_uModem_BND(enum)   1   # BND: Frequency Bank (1, 2, or 3 for band A, B,

                        # or C, 0 for user-defined PSK only band)
```

DON'T CHANGE THIS TO ANYTHING OTHER THAN THE BAND

THAT THE HARDWARE IS CONFIGURED FOR AS THIS MAY DAMAGE

THE POWER AMPLIFIER (BND should be 1 according to

Lee Freitag email May 11, 2009).

sensor: u_uModem_FML(nodim) 200 # PSK FM probe length, symbols

sensor: u_uModem_CST(bool) 1 # Cycle statistics message 0 = off, 1 = on

sensor: u_uModem_DTO(sec) 8 # Data request timeout in seconds

#outputs:

sensor: sci_uModem_is_installed(bool) 0 # in, t--> installed on science

sensor: sci_uModem_error(nodim) 0 # unique number to indicate error type

proglet rinkoII: JFE ALEC RINKO-II disolved oxygen and temperature sensor

sensor: sci_rinkoII_is_installed(bool) 0 # t--> installed on science

#inputs:

sensor: c_rinkoII_on(sec) 2 # in, sets secs between measurements

```
                        # <0 stops, 0 fast as possible, >0 that many secs

sensor: c_rinkoII_num_fields_to_send(nodim) 3
                        # in, number of columns to send on each
                        #    measurement, fields to send chosen
                        #    by order in the list above

#outputs:

sensor: sci_rinkoII_temp(degC)          0 # col 3, temperature

sensor: sci_rinkoII_DO(%)               0 # col 4, disolved oxygen

sensor: sci_rinkoII_voltage(volts)      0 # col 5, voltage output of oxygen sensor

sensor: sci_rinkoII_timestamp(timestamp) 0 # secs since 1970 of data arrival



# proglet dvl for the TRDI ExplorerDVL

#inputs:
sensor: c_dvl_on(sec)               0 # how often start ensembles in seconds
                                # <0 stops, 0 fast as possible, 0> that many secs
sensor: u_dvl_pd_data_stream_select(enum)  0 # (0 or 6) Supports formats PD0 and PD6
sensor: u_dvl_es_expected_salinity(ppt)       35 # (0 - 40) Expected salinity
```

```
                                # of water in parts per thousand.
sensor: u_dvl_bk_water_mass_layer_mode(enum)    2 # 0 = Disables the water-mass layer ping
                                # 1 = Sends a water-mass layer ping after
                                #     every bottom-track ping
                                # 2 = Sends a water-mass layer ping after
                                #     every bottom-track ping that is
                                #     unable to find the bottom.
                                # 3 = Disables the bottom-track ping and
                                #     enables the water-mass ping.
                                #
                                # The far boundary must be greater than the near plus the min
                                # layer size. The minimum layer and the difference between the
                                # near and the far layers cannot be larger than the maximum
                                # profile bin size (800cm for 600 kHz).
sensor: u_dvl_num_errors_before_restart(nodim)  1 # number of errors before cycling power
                                # <0 = never cycle power
sensor: u_dvl_ensemble_timeout(sec)          20 # generate error and retry if no data
                                # after this many seconds
sensor: u_dvl_single_pd0_file(bool)          0 # Select multiple (0) or single (1) pd0 file per segment


#PD6 outputs:
sensor: sci_dvl_is_installed(bool)  0   # in, t--> installed on science
sensor: sci_dvl_error(nodim)        0   # unique number to indicate error type
      # system attitude data
sensor: sci_dvl_sa_pitch(deg)       0   # pitch in degrees
```

sensor: sci_dvl_sa_roll(deg) 0 # roll in degrees

sensor: sci_dvl_sa_heading(deg) 0 # heading in degrees

  # timing and scaling data

sensor: sci_dvl_ts_timestamp(timestamp) 0 # secs since 1970

sensor: sci_dvl_ts_sal(ppt) 0 # salinity in parts per thousand

sensor: sci_dvl_ts_temp(degC) 0 # temp in degC

sensor: sci_dvl_ts_depth(m) 0 # depth of transducer face in meters

sensor: sci_dvl_ts_sound_speed(m/s) 0 # speed of sound in m/s

sensor: sci_dvl_ts_bit(nodim) 0 # Built-in Test (BIT) result code

  # water-mass, instrument-referenced velocity data

sensor: sci_dvl_wi_x_vel(mm/s) 0 # X-axis vel. data in mm/s

sensor: sci_dvl_wi_y_vel(mm/s) 0 # Y-axis vel. data in mm/s

sensor: sci_dvl_wi_z_vel(mm/s) 0 # Z-axis vel. data in mm/s

sensor: sci_dvl_wi_err_vel(mm/s) 0 # Error velocity data in mm/s

sensor: sci_dvl_wi_vel_good(bool) 0 # Velocity data status 0=bad, 1=good

  # bottom-track, instrument-referenced velocity data

sensor: sci_dvl_bi_x_vel(mm/s) 0 # X-axis vel. data in mm/s

sensor: sci_dvl_bi_y_vel(mm/s) 0 # Y-axis vel. data in mm/s

sensor: sci_dvl_bi_z_vel(mm/s) 0 # Z-axis vel. data in mm/s

sensor: sci_dvl_bi_err_vel(mm/s) 0 # Error velocity data in mm/s

sensor: sci_dvl_bi_vel_good(bool) 0 # Velocity data status 0=bad, 1=good

  # water-mass, ship-referenced velocity data

sensor: sci_dvl_ws_transverse_vel(mm/s) 0 # Transverse vel. data in mm/s

sensor: sci_dvl_ws_longitudinal_vel(mm/s) 0 # Longitudinal vel. data in mm/s

sensor: sci_dvl_ws_normal_vel(mm/s) 0 # Normal vel. data in mm/s

sensor: sci_dvl_ws_vel_good(bool) 0 # Vel. data status 0=bad, 1=good

bottom-track, ship-referenced velocity data

sensor: sci_dvl_bs_transverse_vel(mm/s) 0 # Transverse vel. data in mm/s

sensor: sci_dvl_bs_longitudinal_vel(mm/s) 0 # Longitudinal vel. data in mm/s

sensor: sci_dvl_bs_normal_vel(mm/s) 0 # Normal vel. data in mm/s

sensor: sci_dvl_bs_vel_good(bool) 0 # Vel. data status 0=bad, 1=good

water-mass, earth-referenced velocity data

sensor: sci_dvl_we_u_vel(mm/s) 0 # East (u-axis) vel. data in mm/s

sensor: sci_dvl_we_v_vel(mm/s) 0 # North (v-axis) vel. data in mm/s

sensor: sci_dvl_we_w_vel(mm/s) 0 # Upward(w-axis) vel. data in mm/s

sensor: sci_dvl_we_vel_good(bool) 0 # Vel. data status 0=bad, 1=good

bottom-track, earth-referenced velocity data

sensor: sci_dvl_be_u_vel(mm/s) 0 # East (u-axis) vel. data in mm/s

sensor: sci_dvl_be_v_vel(mm/s) 0 # North (v-axis) vel. data in mm/s

sensor: sci_dvl_be_w_vel(mm/s) 0 # Upward(w-axis) vel. data in mm/s

sensor: sci_dvl_be_vel_good(bool) 0 # Vel. data status 0=bad, 1=good

water-mass, earth-referenced distance data

sensor: sci_dvl_wd_u_dist(m) 0 # East (u-axis) distance data in meters

sensor: sci_dvl_wd_v_dist(m) 0 # North (v-axis) distance data in meters

sensor: sci_dvl_wd_w_dist(m) 0 # Upward (w-axis) distance data in meters

sensor: sci_dvl_wd_range_to_water_mass_center(m) 0 # Range to water-mass center in meters

sensor: sci_dvl_wd_time_since_last_good_vel(sec) 0 # Time since last good-velocity estimate in seconds

bottom-track, earth-referenced distance data

sensor: sci_dvl_bd_u_dist(m) 0 # East (u-axis) distance data in meters

sensor: sci_dvl_bd_v_dist(m) 0 # North (v-axis) distance data in meters

sensor: sci_dvl_bd_w_dist(m) 0 # Upward (w-axis) distance data in meters

sensor: sci_dvl_bd_range_to_bottom(m) 0 # Range to bottom in meters

sensor: sci_dvl_bd_time_since_last_good_vel(sec) 0 # Time since last good-velocity estimate in seconds

#PD0 outputs:

sensor: sci_dvl_ensemble_offset(nodim) 0 # Byte offset for each ensemble in the

PD0 binary data file.

proglet flbbrh: Wet Labs flbbrh fluorometer, scattering meter, and rhodamine sensor

sensor: c_flbbrh_on(sec) 0 # in, sets secs between measurements

<0 stops, 0 fast as possible, >0 that many secs

sensor: sci_flbbrh_is_installed(bool) 0 # in, t--> installed on science

sensor: c_flbbrh_num_fields_to_send(nodim) 10 # in, number of columns to send on each

measurement, fields to send chosen

by order in the list below

sensor: u_flbbrh_is_calibrated(bool) 0 # false, assume not calibrated

sensor specific input calibration constants (defaults for FLBBRHSLK-1766)

sensor: u_flbbrh_chlor_cwo(nodim) 48 # clean water offset, nodim == counts

sensor: u_flbbrh_bb_cwo(nodim) 48 # clean water offset, nodim == counts

sensor: u_flbbrh_rhod_cwo(nodim) 58 # clean water offset, nodim == counts

sensor: u_flbbrh_chlor_sf(ug/l/nodim) 0.0123 # scale factor to get units

sensor: u_flbbrh_bb_sf(Mnodim) 3.653 # (0.000003653) scale factor to get units

sensor: u_flbbrh_rhod_sf(ppb/nodim) 0.0430 # scale factor to get units

output sensors, listed in PRIORITY order

e.g. if c_flbbrh_num_fields_to_send is 3, cols derived

from 4,6,8 sent

sensor: sci_flbbrh_chlor_units(ug/l) 0 # derived from col 4

sensor: sci_flbbrh_bb_units(nodim) 0 # derived from col 6

sensor: sci_flbbrh_rhod_units(ppb) 0 # derived from col 8

sensor: sci_flbbrh_chlor_sig(nodim) 0 # col 4

sensor: sci_flbbrh_bb_sig(nodim) 0 # col 6

sensor: sci_flbbrh_rhod_sig(nodim) 0 # col 8

sensor: sci_flbbrh_chlor_ref(nodim) 0 # col 3

sensor: sci_flbbrh_bb_ref(nodim) 0 # col 5

sensor: sci_flbbrh_rhod_ref(nodim) 0 # col 7

sensor: sci_flbbrh_temp(nodim) 0 # col 9

sensor: sci_flbbrh_timestamp(timestamp) 0 # secs since 1970

proglet flur: Wet Labs flur uranine sensor

sensor: c_flur_on(sec) 0 # in, sets secs between measurements

<0 stops, 0 fast as possible, >0 that many secs

sensor: sci_flur_is_installed(bool) 0 # in, t--> installed on science

sensor: c_flur_num_fields_to_send(nodim) 4 # in, number of columns to send on each

measurement, fields to send chosen

by order in the list below

sensor: u_flur_is_calibrated(bool) 0 # false, assume not calibrated

sensor specific input calibration constants (defaults for FLURSLK-1733)

sensor: u_flur_cwo(nodim) 50 # clean water offset, nodim == counts

sensor: u_flur_sf(ppb/nodim) 0.0281 # scale factor to get units

output sensors, listed in PRIORITY order

e.g. if c_flur_num_fields_to_send is 3, cols derived

from 4,3,5 sent

sensor: sci_flur_units(ppb) 0 # derived from col 4

sensor: sci_flur_sig(nodim) 0 # col 4

sensor: sci_flur_ref(nodim) 0 # col 3

sensor: sci_flur_temp(nodim) 0 # col 5

sensor: sci_flur_timestamp(timestamp) 0 # secs since 1970

proglet bb2flsV7: wet labs bb2flslk scatter meter and fluorometer sensor, 7th configuration

sensor: c_bb2flsV7_on(sec) 2 # in, sets secs between measurements

<0 stops, 0 fast as possible, >0 that many secs

sensor: sci_bb2flsV7_is_installed(bool) 0 # in, t--> installed on science

sensor: c_bb2flsV7_num_fields_to_send(nodim) 10 # in, number of columns to send on each

measurement, fields to send chosen

by order in the list below

sensor: u_bb2flsV7_is_calibrated(bool) 0 # false, assume not calibrated

sensor specific input calibration constants (defaults for BB2FLSLK-760)

sensor: u_bb2flsV7_b532_cwo(nodim) 46 # clean water offset, nodim == counts

sensor: u_bb2flsV7_b650_cwo(nodim) 46 # clean water offset, nodim == counts

sensor: u_bb2flsV7_chl_cwo(nodim) 40 # clean water offset, nodim == counts

sensor: u_bb2flsV7_b532_sf(Mnodim) 7.683 # scale factor (0.000007683)

sensor: u_bb2flsV7_b650_sf(Mnodim) 3.893 # scale factor (0.000003893)

sensor: u_bb2flsV7_chl_sf(ug/l/nodim) 0.0121 # scale factor to get units

output sensors, listed in PRIORITY order

e.g. if c_bb2flsV7_num_fields_to_send is 3, cols derived

from 4,6,8 sent

sensor: sci_bb2flsV7_b532_scaled(nodim) 0 # derived from col 4

sensor: sci_bb2flsV7_b650_scaled(nodim) 0 # derived from col 6

sensor: sci_bb2flsV7_chl_scaled(ug/l) 0 # derived from col 8

sensor: sci_bb2flsV7_b532_sig(nodim) 0 # col 4

sensor: sci_bb2flsV7_b650_sig(nodim) 0 # col 6

sensor: sci_bb2flsV7_chl_sig(nodim) 0 # col 8

sensor: sci_bb2flsV7_b532_ref(nodim) 0 # col 3

sensor: sci_bb2flsV7_b650_ref(nodim) 0 # col 5

sensor: sci_bb2flsV7_chl_ref(nodim) 0 # col 7

sensor: sci_bb2flsV7_therm(nodim) 0 # col 9

sensor: sci_bb2flsV7_timestamp(timestamp) 0 # secs since 1970

proglet flbbcd: Wet Labs flbbcd fluorometer, scattering meter, and cdom sensor

sensor: c_flbbcd_on(sec) 0 # in, sets secs between measurements

<0 stops, 0 fast as possible, >0 that many secs

sensor: sci_flbbcd_is_installed(bool) 0 # in, t--> installed on science

sensor: c_flbbcd_num_fields_to_send(nodim) 10 # in, number of columns to send on each

measurement, fields to send chosen

by order in the list below

sensor: u_flbbcd_is_calibrated(bool) 0 # false, assume not calibrated

sensor specific input calibration constants (defaults for FLBBCDSLK-1845)

sensor: u_flbbcd_chlor_cwo(nodim) 35 # clean water offset, nodim == counts

sensor: u_flbbcd_bb_cwo(nodim) 49 # clean water offset, nodim == counts

sensor: u_flbbcd_cdom_cwo(nodim) 47 # clean water offset, nodim == counts

sensor: u_flbbcd_chlor_sf(ug/l/nodim) 0.0119 # scale factor to get units

sensor: u_flbbcd_bb_sf(Mnodim) 3.522 # (0.000003522) scale factor to get units

sensor: u_flbbcd_cdom_sf(ppb/nodim) 0.0919 # scale factor to get units

```
                # output sensors, listed in PRIORITY order
                # e.g. if c_flbbcd_num_fields_to_send is 3, cols derived
                # from 4,6,8 sent
sensor: sci_flbbcd_chlor_units(ug/l)    0 # derived from col 4
sensor: sci_flbbcd_bb_units(nodim)      0 # derived from col 6
sensor: sci_flbbcd_cdom_units(ppb)      0 # derived from col 8
sensor: sci_flbbcd_chlor_sig(nodim)     0 # col 4
sensor: sci_flbbcd_bb_sig(nodim)        0 # col 6
sensor: sci_flbbcd_cdom_sig(nodim)      0 # col 8
sensor: sci_flbbcd_chlor_ref(nodim)     0 # col 3
sensor: sci_flbbcd_bb_ref(nodim)        0 # col 5
sensor: sci_flbbcd_cdom_ref(nodim)      0 # col 7
sensor: sci_flbbcd_therm(nodim)         0 # col 9
sensor: sci_flbbcd_timestamp(timestamp) 0 # secs since 1970


# proglet dmon: W.H.O.I DMON, Digital Monitor, a passive acoustic monitor


sensor: sci_dmon_is_installed(bool) 0 # t--> installed on science


#inputs:
sensor: c_dmon_on(sec)   0          # >= 0 enables the device
sensor: u_dmon_ctd_msg_period(sec) 0 # How often to send ctd data to the DMON, this
                                # assumes that the user has configured the CTD
                                # to be sampling whenever the DMON is enabled.
```

#outputs:

sensor: sci_dmon_msg_byte_count(nodim) 0 # message byte count for open disk file

proglet suna: Submersible Ultraviolet Nitrate Analyzer from Satlantic

sensor: sci_suna_is_installed(bool) 0 # t--> installed on science

#inputs:

sensor: c_suna_on(sec) 0 # >= 0 enables the device

(must take into account u_suna_bootup_time)

sensor: u_suna_num_errors_before_restart(nodim) 5 # number of errors before cycling

power, <0 = never cycle power

0 = restart on any error

sensor: u_suna_bootup_time(sec) 10 # bootup time of the SUNA sensor

sensor: f_suna_firmware_cfg(enum) 0 # SUNA Firmware configuration.

0 - SUNA Firmware version 1 (Unit 110)

1 - SUNA Firmware version 2 (Unit 219)

#outputs

sensor: sci_suna_record_offset(bytes) 0 # message byte count for open disk file

sensor: sci_suna_nitrate_um(uMol/L) 0 # Nitrate concentration in uMol/L

sensor: sci_suna_nitrate_mg(mg/L) 0 # Nitrate concentration in mg/L

sensor: sci_suna_timestamp(sec) 0 # Timestamp of sample

proglet c3sfl: Turner Designs C3 Submersible Fluorometer

```
#inputs:
sensor: c_c3sfl_on(sec)  0          # in, sets secs between measurements
                            # <0 stops, 0 fast as possible, >0 that many secs

sensor: c_c3sfl_num_fields_to_send(nodim)  4
                            # in, number of columns to send on each
                            # measurement, fields to send chosen
                            # by order in the list below
                            # A negative value signifies
                            # to use this value as a bitmap.
                            # The user may specify any
                            # outputs regardless of order by
                            # by using this sensor as a bitmap.
                            # -1 * (2^(f1-1)+2^(f2-1)+...)
                            # where fn is the field number.
                            # For example, if the user wished
                            # to just record temperature
                            # they would use:
                            # -1 * 2^(6-1) = -32

sensor: sci_c3sfl_is_installed(bool) 0 # in, t--> installed on science

#outputs:
  sensor: sci_c3sfl_chlorophyll(rfu)     0 # col 3 Relative Fluorescence Units
  sensor: sci_c3sfl_phycoerythrin(rfu)   0 # col 4
```

```
 sensor: sci_c3sfl_turbidity(rfu)      0 # col 5
 sensor: sci_c3sfl_cdom(rfu)           0 # col 6
 sensor: sci_c3sfl_depth(m)            0 # col 7
 sensor: sci_c3sfl_temp(degC)          0 # col 8
 sensor: sci_c3sfl_timestamp(timestamp) 0 # secs since 1970

 # proglet satpar: Satlantic PAR sensor

sensor: sci_satpar_is_installed(bool) 0 # in, t--> installed on science

#inputs:
sensor: c_satpar_on(sec)  0          # in, sets secs between measurements
                          # <0 stops, 0 fast as possible,
                          # >0 that many secs

sensor: c_satpar_num_fields_to_send(nodim)  3
                          # in, number of columns to send on each
                          # measurement, fields to send chosen
                          # by order in the list above
                          # A negative value signifies
                          # to use this value as a bitmap.
                          # The user may specify any
                          # outputs regardless of order by
                          # by using this sensor as a bitmap.
                          # -1 * (2^(f1-1)+2^(f2-1)+...)
```

```
                        # where fn is the field number.
                        # For example, if the user wished
                        # to just record raw counts
                        # they would use:
                        # -1 * 2^(3-1) = -4

sensor: u_satpar_is_calibrated(bool)    0 # needs to be set in autoexec.mi

# sensor specific input calibration constants (defaults for S/N 0171)
sensor: u_satpar_immersion_coeff(nodim) 1.3589     # calibration coeffients
sensor: u_satpar_dark_offset(nodim)     2156930000 #     "          "
sensor: u_satpar_slope(Mnodim)          2.44356    # (0.00000244356)

#outputs:
sensor: sci_satpar_par(umol-photons/m^2/s) 0 # derived from col 3
sensor: sci_satpar_raw_counts(nodim)       0 # col 3
sensor: sci_satpar_timer(sec)              0 # col 2
sensor: sci_satpar_timestamp(timestamp)    0 # secs since 1970


# proglet vsf: Wet Labs Volume Scattering Function meter
sensor: c_vsf_on(sec)       0 # in, sets secs between measurements
                        # <0 stops, 0 fast as possible, >0 that many secs
sensor: sci_vsf_is_installed(bool) 0 # in, t--> installed on science
```

sensor: c_vsf_num_fields_to_send(nodim) 7 # in, number of columns to send on each

measurement, fields to send chosen

by order in the list below

sensor: u_vsf_is_calibrated(bool) 0 # false, assume not calibrated

sensor specific input calibration constants (defaults for VSFSLK-147)

sensor: u_vsf_100_dc(nodim) 49 # dark counts, nodim == counts

sensor: u_vsf_125_dc(nodim) 50 # dark counts, nodim == counts

sensor: u_vsf_150_dc(nodim) 40 # dark counts, nodim == counts

sensor: u_vsf_100_sf(Mnodim) 4.820 # (0.000004820) scale factor to get units

sensor: u_vsf_125_sf(Mnodim) 3.527 # (0.000003527) scale factor to get units

sensor: u_vsf_150_sf(Mnodim) 3.006 # (0.000003006) scale factor to get units

output sensors, listed in PRIORITY order

e.g. if c_vsf_num_fields_to_send is 3, cols derived

from 4,5,6 sent

sensor: sci_vsf_100_scaled(nodim) 0 # derived from col 4

sensor: sci_vsf_125_scaled(nodim) 0 # derived from col 5

sensor: sci_vsf_150_scaled(nodim) 0 # derived from col 6

sensor: sci_vsf_100_sig(nodim) 0 # col 4

sensor: sci_vsf_125_sig(nodim) 0 # col 5

sensor: sci_vsf_150_sig(nodim) 0 # col 6

sensor: sci_vsf_therm(nodim) 0 # col 7

sensor: sci_vsf_timestamp(timestamp) 0 # secs since 1970

proglet : Aanderaa Oxygen Optode 4330F or 4831

sensor: c_oxy4_on(sec) 0 # in, sets secs between measurements

<0 stops, 0 fast as possible, >0 that many secs

sensor: sci_oxy4_is_installed(bool) 0 # in, t--> installed on science

sensor: c_oxy4_num_fields_to_send(nodim) 10 # in, number of columns to send on each

measurement, fields to send chosen

by order in the list below

sensor: u_oxy4_slow_surface_mode(bool) 1 # default for 4330f, make false for 4831 model.

output sensors, listed in PRIORITY order

e.g. if c_oxy4_num_fields_to_send is 3, cols 3,4,5 sent

sensor: sci_oxy4_oxygen(uM) 0 # col 3, O2 Concentration

sensor: sci_oxy4_saturation(%) 0 # col 4, air saturation

sensor: sci_oxy4_temp(degC) 0 # col 5, temperature

sensor: sci_oxy4_calphase(deg) 0 # col 6, CalPhase

sensor: sci_oxy4_tcphase(deg) 0 # col 7, TCPhase

sensor: sci_oxy4_c1rph(deg) 0 # col 8, C1RPh

sensor: sci_oxy4_c2rph(deg) 0 # col 9, C2RPh

sensor: sci_oxy4_c1amp(mV) 0 # col 10, C1Amp

sensor: sci_oxy4_c2amp(mV) 0 # col 11, C2Amp

sensor: sci_oxy4_rawtemp(mV) 0 # col 12, RawTemp

sensor: sci_oxy4_timestamp(timestamp) 0 # secs since 1970

proglet gamma_rad5: Amptek Gamma-Ray Detection System (GAMMA_RAD5).

Inputs:

sensor: c_gamma_rad5_on(sec) 0 # >= 0 enables the device

sensor: u_gamma_rad5_status_period(sec) 60 # how often to request status packets

sensor: u_gamma_rad5_files_before_surfacing(nodim) 5 # number of files to collect before
requesting to surface. <= 0 means never
request to surface.

Outputs:

sensor: sci_gamma_rad5_is_installed(bool) 0 # in, t--> installed on science

sensor: sci_gamma_rad5_file_count(nodim) # number of spectrum packets acquired

sensor: sci_gamma_rad5_error(enum) # unique number to indicate error type

proglet bsipar: BioSpherical Instruments PAR sensor

Inputs:

sensor: c_bsipar_on(sec) 0 # in, sets secs between measurements
<0 stops, 0 fast as possible,
>0 that many secs

sensor: c_bsipar_num_fields_to_send(nodim) 4

in, number of columns to send on each

measurement, fields to send chosen

by order in the list above

A negative value signifies

to use this value as a bitmap.

The user may specify any

outputs regardless of order by

by using this sensor as a bitmap.

-1 * (2^(f1-1)+2^(f2-1)+...)

where fn is the field number.

For example, if the user wished

to just record raw counts

they would use:

-1 * 2^(3-1) = -4

sensor: u_bsipar_is_calibrated(bool) 0 # needs to be set in autoexec.mi

sensor specific calibration coefficients (defaults for Model:QSP2155 S/N:50136)

sensor: u_bsipar_dark_offset(volts)   0.0101 # 10.1 mV

sensor: u_bsipar_scale_factor(Mnodim) 589.7 # (0.0005897 volts/uE/m^2sec)

Outputs:

sensor: sci_bsipar_is_installed(bool) 0   # in, t--> installed on science

sensor: sci_bsipar_par(uE/m^2sec)   # derived from col 1

```
sensor: sci_bsipar_sensor_volts(volts)     # col 1

sensor: sci_bsipar_temp(degC)              # col 2

sensor: sci_bsipar_supply_volts(volts)     # col 3

sensor: sci_bsipar_timestamp(timestamp)    # secs since 1970



# proglet flbb: wet labs flbb fluorometer and scattering meter

sensor: c_flbb_on(sec)                  0 # in, sets secs between measurements

                                        # <0 stops, 0 fast as possible, >0 that many secs

sensor: sci_flbb_is_installed(bool)       0 # in, t--> installed on science


sensor: c_flbb_num_fields_to_send(nodim)  4 # in, number of columns to send on each

                                        # measurement, fields to send chosen

                                        # by order in the list below

                                        # in, number of columns to send on each

                                        # A negative value signifies

                                        # to use this value as a bitmap.

                                        # The user may specify any

                                        # outputs regardless of order by

                                        # by using this sensor as a bitmap.

                                        # -1 * (2^(f1-1)+2^(f2-1)+...)

                                        # where fn is the field number.

                                        # For example, if the user wished

                                        # to just record derived units and

                                        # the timestamp they would use:
```

```
                                # -1 * (2^(1-1)+2^(2-1)+2^(8-1)) = -131

sensor: u_flbb_is_calibrated(bool)       0 # false, assume not calibrated

# sensor specific input calibration constants (defaults for FLBBSLK-2414)
sensor: u_flbb_chlor_do(nodim)     51    # dark water offset, nodim == counts
sensor: u_flbb_bb_do(nodim)        52    # dark water offset, nodim == counts
sensor: u_flbb_chlor_sf(ug/l/nodim) 0.0072 # scale factor to get units
sensor: u_flbb_bb_sf(Mnodim)       1.921  # really 0.000001921 (see Mnodim doco above)

                # output sensors, listed in PRIORITY order
                # e.g. if c_flbb_num_fields_to_send is 2, cols derived
                # from 4,6 sent
sensor: sci_flbb_chlor_units(ug/l)   0 # derived from col 4
sensor: sci_flbb_bb_units(nodim)     0 # derived from col 6
sensor: sci_flbb_chlor_sig(nodim)    0 # col 4
sensor: sci_flbb_bb_sig(nodim)       0 # col 6
sensor: sci_flbb_chlor_ref(nodim)    0 # col 3
sensor: sci_flbb_bb_ref(nodim)       0 # col 5
sensor: sci_flbb_therm(nodim)        0 # col 7
sensor: sci_flbb_timestamp(timestamp) 0 # secs since 1970

# Vemco VR2C Proglet specific sensors.
sensor: c_vr2c_on(sec)                -1 # in, sets secs between measurements
                                # <0 stops, 0 real-time, >0 polling interval
```

sensor: sci_vr2c_is_installed(bool) 0 # in, 1-->installed on science

sensor: c_vr2c_num_fields_to_send(nodim) 2 # number of fields to send to science.

#outputs

sensor: sci_vr2c_state(nodim) -1 # current state of the VR2C proglet

Imagenex

#input sensors

sensor: c_echosndr853_on(sec) -1

sensor: sci_echosndr853_is_installed(bool) 0

#output sensors

sensor: sci_echosndr853_ping_count(nodim) 0

Used if there are two CTD's installed. ctd41cp2 will output to these

sensors.

Inputs:

sensor: c_ctd41cp2_on(sec) 0 # in, sets secs between measurements

<0 stops, 0 fast as possible,

>0 that many secs

sensor: sci_ctd41cp2_is_installed(bool) 0 # in, t--> ctd installed on science

sensor: c_ctd41cp2_num_fields_to_send(nodim) 3 # in, number of columns to send on each

measurement, fields to send chosen

by order in the list below

A negative value signifies

to use this value as a bitmap.

The user may specify any

outputs regardless of order by

by using this sensor as a bitmap.

-1 * (2^(f1-1)+2^(f2-1)+...)

where fn is the field number.

For example, if the user wished

to just record temperature

they would use:

-1 * 2^(2-1) = -2

sensor: sci_water_cond2(S/m) 3 # out, conductivity f#=1

sensor: sci_water_temp2(degC) 10 # out f#=2

sensor: sci_water_pressure2(bar) 0 # out f#=3

sensor: sci_ctd41cp2_timestamp(timestamp) 0 # out, secs since 1970 f#=4

#tritech.c

sensor: c_tritech_on(sec) 0

sensor: c_tritech_flag(nodim) 0 #1 -use tritech data for heading control

do not use

sensor: sci_tritech_is_installed(bool) 0

sensor: c_tritech_srange(nodim) 70

sensor: c_tritech_leftlim(nodim) 210

sensor: c_tritech_rightlim(nodim) 330

sensor: c_tritech_frequency(nodim) 2

sensor: c_tritech_num_fields_to_send(nodim) 4

sensor: sci_tritech_range(nodim) 0

sensor: sci_tritech_angle(nodim) 0

sensor: sci_tritech_timestamp(timestamp) 0

sensor: sci_tritech_heading(rad) 0.5

sensor: c_tritech_depth(nodim) 5

sensor: c_tritech_sectorl(nodim) 260

sensor: c_tritech_sectorr(nodim) 280

sensor: c_tritech_inithead(rad) 1.2

sensor: c_tritech_pgain(nodim) 20

<PROTO> Add additional science proglets here

console.c

sensor: c_console_on(bool) 2.0 # in 0 power it off

1 power on automatically at surface

power off automatically when underwater AND

no carrier for U_CONSOLE_REQD_CD_OFF_TIME secs

2 power on regardless

sensor: u_console_reqd_cd_off_time(sec) 15.0 # in, how long without CD before powering off

modem if C_CONSOLE_ON == 1

sensor: m_console_on(bool) 1.0 # out, power state of RF modem

```
sensor: m_console_cd(bool)            1.0  # out, state of RF modem carrier detect

sensor: u_console_off_if_mission_iridium(bool) 1.0 #! visible = True
                                      # in, if non-zero causes the freewave
                                      # to be powered off during a mission if a
                                      # carrier isn't detected.

sensor: f_ignore_console_cd_time(sec)   5.0  # in, how long to "filter", i.e. ignore
                                      #     carrier detect after the freewave is
                                      #     just powered on.


sensor: m_chars_tossed_with_power_off(nodim) 0 # out, chars eaten with power off
sensor: m_chars_tossed_with_cd_off(nodim)    0 # out, chars eaten with CD off
sensor: m_chars_tossed_by_abend(nodim)       0 # out, chars eaten by abend
                                       #          this one maintained by behavior abend,
                                       #          listed here for completeness

sensor: u_console_announce_time(sec)  60    # controls how often glidername
                                     # is announced when M_CONSOLE_CD
                                     # <0 disables announcement
sensor: x_console_announcement_made(nodim) 0 # incremented whenever an announcement is made
```

```
#=======================================================
# Gliderbus (gbus) devices
#   gb_devdrvr paradigm devices use:
#     where <X> is the device driver name (as listed in "use")
#
#       C_<X>_ON      ; How often to measure
#                     ; <0 => never(off)
#                     ;  0 => fast as possible
#                     ; >0 => every this many seconds
#
#       U_GLIDERBUS_DEBUG  ; controls whole bus
#       U_<X>_DEBUG       ; controls device <X>
#                         ;   or'ed together for each device
#                         ; bit-mapped debug fields
#                         ; add desired on bit values (2^N) together
#                         ;   the composite value
#                         ; SEE top of gb_devdrvr.c for meaning
#
#   each device is free to define it's on M_<X>_whatever
#
# Bus wide
sensor: u_gliderbus_debug(nodim)   0.0



# coulomb counter
```

sensor: c_coulomb_on(sec) 0 # required by gb_devdrvr paradigm

sensor: u_coulomb_debug(nodim) 0

sensor: f_coulomb_calibration_factor(%) .05 # calibration factor for the

onboard coulomb counter

sensor: m_coulomb_amphr(amp-hrs) 0.0 # integrated current, i.e. energy

sensor: m_coulomb_current(amp) 0.0 # instantaneous current

sensor: m_coulomb_amphr_raw(nodim) 0.0

sensor: m_coulomb_current_raw(nodim) 0.0

sensor: m_coulomb_amphr_total(amp-hrs) 0.0 # persistant amp-hours total

sensor: f_coulomb_battery_capacity(amp-hrs) 720.0 # nominal battery capacity

sensor: s_coulomb_relative_charge(%) 50.0 # Simulated relative charge (wall_power provided in simul.sim)

digifin_v2 (a gbus version of digifin)

sensor: f_fin_safety_max(rad) 0.47 # in, damage to glider

sensor: u_digifin_v2_debug(nodim) 0

sensor: c_fin(rad) 0 # in, >0 vehicle turns right

sensor: x_last_commanded_fin_pos(rad) 0 # stores the last commanded fin position.

sensor: m_digifin_rawposition(nodim) 0 # raw position in A/D counts

sensor: m_fin(rad) # out

#===

Clock Source

```
sensor: f_clock_source(enum)   1    # in, defines the real time clock source.
                        #  0 - Use the RTC provided by the persistor
                        #  1 - Use DS3234 RTC (new hardware only)
                        #  # - Any other value will default to RTC provided
                        #      by the persistor.

#=======================================================


# A variety of simulated variables.  These are all maintained by
# simdrvr.c.
   # Keep track of if simulating
sensor: x_are_simulating(enum)   0   # out
                        # 0   not simulating
                        # 3   on bench
                        # 2   just electronics
                        # 1   no electronics

sensor: s_hardware_ver(nodim)   128  # what no_electronics reports for X_HARDWARE_VER
```

```
                        # RevE board.
                        #  This is only read at startup, to change it,
                        # you probably have to change it here
                        # and recompile or store it as longterm sensor.
sensor: s_hardware_cop_jumper(bool) 0 # simulated jumper setting for no_electronics only
                         # 0 2hr, 1 16hr

   # Configuration(environmental) controls
sensor: xs_water_depth(m) 30.0  # How deep the water is (COMPUTED! do not set directly)
                        # xs_water_depth = s_water_depth_avg -
                        #              s_water_depth_delta *
                        #              sin( 2PI * r / s_water_depth_wavelength)
                        # where r = current distance from (0,0) LMC
sensor: s_water_depth_avg(m)        30.0
sensor: s_water_depth_delta(m)       0.0
sensor: s_water_depth_wavelength(m) 100.0



sensor: s_water_cond(S/m)   4.0  # conductivity, How salty it is

sensor: xs_water_temp(degC) 00  # How warm water is, (COMPUTED! do not set directly)
  sensor: s_water_temp_surface(degc)   20.0   # temp above
  sensor: s_water_temp_depth_inft(m)    5.0   #      this depth (inflection top)
```

```
 sensor: s_water_temp_bottom(degc)     4.0   # temp below
 sensor: s_water_temp_depth_infb(m)  500.0   #      this depth (inflection bottom)
 # mnenonic: ....INF(T/B) stands for inflection top and inflection bottom.


      #XS_VEHICLE_TEMP = S_VEHICLE_TIME_TC * ( XS_WATER_TEMP - XS_VEHICLE_TEMP) * delta_t
sensor: xs_vehicle_temp(degC)  25.0   # How warm vehicle is
sensor: s_vehicle_temp_tc(1/sec) 0.01 #    tc ==> time constant
                              #    See simdrvr.c do_xs_vehicle_temp() for derivation



sensor: s_wind_speed(m/s)      9.0  # how fast the wind is blowing, 3.0 ==> 5.4 knots
sensor: s_wind_direction(rad) 0.0  # Direction wind is blowing FROM

sensor: s_water_speed(m/s)      0.05     # Current speed,   0.5 ==> 1knot
sensor: s_water_direction(rad)  4.712    # direction current is going TO,
                                 # toward the west

sensor: s_mag_var(rad) 0.2810     # mag_heading = true_heading + mag_var
                             #    mag_var>0 ==>  variation is West (like on cape cod)
                             # This is cape cod number

sensor: xs_wax_temp(degC)         20    # temperature of working fluid
sensor: xs_wax_frac_frozen(nodim)  0    # what fraction of the fluid is frozen
sensor:  s_wax_freeze_temp(degC)  10    # where it freezes
```

do_thermal_oil

sensor: xs_thermal_aft_oil_vol(cc)   # simulated oil volume in the aft bladder

sensor: xs_thermal_int_oil_vol(cc)   # simulated oil volume in the interior reservoir

sensor: xs_thermal_tube_oil_vol(cc) # simulated oil volume in the external tube

sensor: xs_thermal_acc_oil_vol(cc)   # simulated oil volume in the accumulator

combination config/working

Glider real world location

DO NOT CHANGE THESE SETTINGS

Users should PUT s_ini_lat or s_ini_lon to change

simulated glider location

sensor: xs_lat(deg)   4138.051 # Ashumet

sensor: xs_lon(deg)   -7032.124

Users should change these to move the simulated glider

position

sensor: s_ini_lat(deg)   69696969   # these are purposely set to

sensor: s_ini_lon(deg)   69696969   # unreasonable values

sensor: x_simulated_position_moved(bool) 0 # non-zero means user moved simulated position

flag between gps.c and simdrvr.c

to tell gps to skip moved too far check

set in simdrvr.c, cleared in gps.c

deep electric observed oil pot voltage rate of change

sensor: s_de_oil_pot_volt_flux(volts/sec) 0.0031

working

sensor: x_simdrvr_ran(out) 0 # out, set to 1 on every simdrvr_ctrl() call

sensor: xs_battpos(in) 0 # simdrvr.c, do_glider_internals()

sensor: xs_battroll(rad) 0

sensor: xs_ballast_pumped(cc) 0

sensor: xs_fluid_pumped(cc) 0

sensor: xs_fin(rad) 0

sensor: xs_roll(rad) 0 # simdrvr.c, do_glider_attitude()

sensor: xs_pitch(rad) 0

sensor: xs_depth(m) 0 # simdrvr.c, do_glider_depth()

sensor: xs_altitude(m) 0 # how far above bottm

sensor: xs_vert_speed(m/s) 0 # veh vert speed thru water

sensor: s_ocean_pressure_min(volts) 0.20 # used to generate voltage for 0 pressure

sensor: xs_pressure_drift(volts) 0 # integrated pressure drift

sensor: xs_pressure_noise(bar) 0 # simulated random noise to be added to simulated pressure reading

sensor: xs_hdg_rate(rad/sec) 0

sensor: xs_heading(rad) 0

sensor: xs_speed(m/s) 0 # veh horz speed thru water

sensor: xs_vx_lmc(m/s) 0 # vehicle horizontal velocity OVER GROUND

sensor: xs_vy_lmc(m/s) 0

sensor: xs_x_lmc(m) 0 # vehicle position in Local Mission Coordinates

sensor: xs_y_lmc(m) 0 # (0,0) at mission start Y axis is magnetic north

These are set to 1 if bad data is generated for a device

sensor: s_corrupted_altitude(bool) 0 # altimeter

sensor: s_corrupted_gps(bool) 0 # The gps, valid or invalid

sensor: s_corrupted_gps_error(bool) 0 # The gps, error added to fix

sensor: s_corrupted_watchdog_oddity(bool) 0 # watchdog generated oddity

sensor: s_corrupted_bpump_stalled(bool) 0 # buoyancy pump "jammed"

sensor: s_corrupted_bpump_overheated(bool) 0 # buoyancy pump overheat bit went high

sensor: s_corrupted_pitch_stalled(bool) 0 # pitch motor "jammed"

sensor: s_corrupted_memory_leak(bool) 0 # We leaked some heap memory

sensor: s_corrupted_pressure_drift(bool) 0 # we generated a pressure drift

sensor: s_corrupted_pressure_spike(bool) 0 # we generated an ocean pressure spike

sensor: s_corrupted_pressure_noise(bool) 0 # we generated ocean pressure noise

sensor: s_corrupted_oil_volume(bool) 0 # we generated an oil volume out-of-deadband

error metrics

sensor: xs_x_lmc_error(m)   0 # m_x/y_lmc - s_x/y_lmc

sensor: xs_y_lmc_error(m)   0

sensor: xs_speed_error(m/s) 0 #xs_speed_error = m_speed - xs_speed

test_driver

sensor: u_test_driver_errors_per_min(nodim)   0.0 # Only for testing error handling

sensor: u_test_driver_warnings_per_min(nodim) 0.0 # Only for testing error handling

sensor: u_test_driver_oddities_per_min(nodim) 0.0 # Only for testing error handling

DBD/SBD header control for header

sensor: u_dbd_sensor_list_xmit_control(enum) 1 # -1 = always transmit header, compatibility mode

use for legacy shore side programs

0 = always transmit header

1 = transmit header on initial mission segment only

2 = transmit header if THIS glider hasn't sent it before

3 = never transmit header

The same for science side data logging; default is more conservative

because headers are not that large on science side

sensor: u_sci_dbd_sensor_list_xmit_control(enum) 0 # -1 = always transmit header, compatibility mode

use for legacy shore side programs

0 = always transmit header

1 = transmit header on initial mission segment only

2 = transmit header if THIS glider hasn't sent it before

3 = never transmit header

Science data logging state as known by glider

KEEP THIS IN SYNC WITH THE enum IN science_super.c !!!!!

sensor: x_science_logging_state(enum) 99 # 0 = pending turn on

1 = turning on

2 = turned on

3 = pending turn off

4 = turning off

5 = turned off

99 = in limbo

File system make-space pruning

sensor: u_reqd_disk_space(Mbytes) 10.0 # How much disk space do we want to keep free

as a minimum. ~ 1 Mbyte/hour is generated

sensor: m_disk_usage(Mbytes) 0.0 # How much disk space is currently used on glider

sensor: sci_m_disk_usage(Mbytes) 0.0 # How much disk space is currently used on science

sensor: m_disk_free(Mbytes) 0.0 # How much disk space is currently free on glider

sensor: sci_m_disk_free(Mbytes) 0.0 # How much disk space is currently free on science

sensor: x_disk_files_removed(nodim) 0 # Count of how many files pruned last time on glider

sensor: sci_x_disk_files_removed(nodim) 0 # Count of how many files pruned last time on science

Send log files time requirement calculation

sensor: u_freewave_data_rate(KBps) 3.0 # Nominal data throughput on Freewave kilobytes per second

sensor: u_iridium_data_rate(KBps) 0.1 # Nominal data throughput on Iridium kilobytes per second

```
# Some documentation on b_args common to all behaviors
# NOTE: When you add these common b_args, put them at END of b_arg
#      list for behaviors.  They do not "naturally" belong there, but
#      it means you do not have to edit behaviors which typically have
#      hardwired b_arg positions in them

# NOTE: These are symbolically defined beh_args.h
# b_arg: START_WHEN     When the behavior should start, i.e. go from UNITIALIZED to ACTIVE
#    BAW_IMMEDIATELY    0   // immediately
#    BAW_STK_IDLE       1   // When stack is idle (nothing is being commanded)
#    BAW_PITCH_IDLE     2   // When pitch is idle(nothing is being commanded)
#    BAW_HEADING_IDLE   3   // When heading is idle(nothing is being commanded)
#    BAW_UPDWN_IDLE     4   // When bpump/threng is idle(nothing is being commanded)
#    BAW_NEVER          5   // Never stop
#    BAW_WHEN_SECS      6   // After behavior arg "when_secs", from prior END if cycling
#    BAW_WHEN_WPT_DIST  7   // When sensor(m_dist_to_wpt) < behavior arg "when_wpt_dist"
#    BAW_WHEN_HIT_WAYPOINT 8 // When X_HIT_A_WAYPOINT is set by goto_wpt behavior
#    BAW_EVERY_SECS     9   // After behavior arg "when_secs", from prior START if cycling
#    BAW_EVERY_SECS_UPDWN_IDLE 10  // After behavior arg "when_secs", from prior START AND
#                           //      updown is idle, no one commanding vertical motion
#    BAW_SCI_SURFACE    11  // SCI_WANTS_SURFACE is non-zero
#    BAW_NOCOMM_SECS    12  // when have not had comms for WHEN_SECS secs
#    BAW_WHEN_UTC_TIME  13  // At a specific UTC time or UTC minute into the hour
#    BAW_HOVER_ACTIVE   14  // Drifting at depth (hovering)
#
```

```
# b_arg: STOP_WHEN
#   0   complete
#   1-N same as "start_when"




# ----- This is the start of a typical mission


behavior: abend
                                # MS_ABORT_OVERDEPTH
   b_arg: overdepth(m)              10000.0 # <0 disables,
                                # clipped to F_MAX_WORKING_DEPTH
   b_arg: overdepth_sample_time(sec)    15.0     #! simple=False
                                   # how often to check


                                # MS_ABORT_OVERTIME
   b_arg: overtime(sec)               -1.0 # < 0 disables


                                # MS_ABORT_UNDERVOLTS
   b_arg: undervolts(volts)            10.0 # < 0 disables
   b_arg: undervolts_sample_time(sec)   60.0     #! simple=False
                                   # how often to check


                                # MS_ABORT_SAMEDEPTH
```

```
b_arg: samedepth_for(sec)           1800.0   #! simple=False
                                    # <0 disables
b_arg: samedepth_for_sample_time(sec)   30.0   #! simple=False
                                    # how often to check


                                  # MS_ABORT_STALLED
b_arg: stalled_for(sec)           1800.0     #! simple=False
                                    # <0 disables
b_arg: stalled_for_sample_time(sec) 1800.0     #! simple=False
                                    # how often to check


                                  # MS_ABORT_NO_TICKLE
b_arg: no_cop_tickle_for(sec)      48600.0     #! simple=False
                                    # secs, abort mission if watchdog
                                  # not tickled this often, <0 disables
b_arg: no_cop_tickle_percent(%)      -1.0     #! simple=False
                                    # 0-100, <0 disables
                                  # Abort this % of time before
                                  # hardware, i.e. for 12.5%
                                  #  hardware 2hr   15min before
                                  #         16hr    2hr  before
      # Note: no_cop_tickle_percent only used on RevE boards or later
      # If non-zero and hardware supports COP timeout readback...
      #           causes no_cop_tickle_for(sec) to be IGNORED
      # On old boards, no_cop_tickle_percent(%) is IGNORED and
```

```
      #       control reverts to no_cop_tickle_for(sec)


                              # MS_ABORT_ENG_PRESSURE, thermal only
b_arg: eng_pressure_mul(nodim)     0.90      #! simple=False
                                  # abort if M_THERMAL_ACC_PRES <
                              #   (eng_pressure_mul * F_THERMAL_REQD_ACC_PRES)


b_arg: eng_pressure_sample_time(sec)  15.0     #! simple=False
                                  # how often to measure, <0 disables


b_arg: max_wpt_distance(m)         -1.0 # MS_ABORT_WPT_TOOFAR
                              # Maximum allowable distance to a waypoint
                              # < 0 disables


b_arg: chk_sensor_reasonableness(bool) 1      #! simple=False
                                  # MS_ABORT_UNREASONABLE_SETTINGS
                              # 0 disables check


b_arg: reqd_spare_heap(bytes)     50000      #! simple=False
                                  # MS_ABORT_NO_HEAP if M_SPARE_HEAP is less than this
                              # <0 disables check
                              ######################################################
                              # NOTE - VALUE OF REQD_SPARE_HEAP IN LASTGASP.MI
                              # SHOULD BE MAINTAINED LOWER THAN THIS NUMBER SO
                              # IF A MISSION ABORTS WITH MS_ABORT_NO_HEAP AND WE
```

```
                                # SEQUENCE TO LASTGASP.MI, THAT IN TURN WILL NOT

                                # ITSELF LIKEWISE DO A HEAP ABORT

                                ##################################################


   b_arg: leakdetect_sample_time(sec)  60.0      #! simple=False

                                    # MS_ABORT_LEAK, M_LEAK is non-zero

                                # <0 disables check


   b_arg: vacuum_min(inHg)           4.0  # MS_ABORT_VACUUM, M_VACUUM out of limits

   b_arg: vacuum_max(inHg)           12.0

   b_arg: vacuum_sample_time(sec)    120.0  # <0 disables check

   b_arg: oil_volume_sample_time(sec) 180.0      #! simple=False

                                    # how often to measure, <0 disables check

   b_arg: max_allowable_busy_cpu_cycles(cycles) 75 #! simple=False

                                    # aborts if M_DEVICE_DRIVERS_CALLED_ABNORMALLY

                                     # is true for this many cycles in a row

                                     # <= 0 disables the abort, 75 = ~5min assuming

                                     # a 4 second cycle time.


   b_arg: remaining_charge_min(%)          10.0  # MS_ABORT_CHARGE_MIN out of limits

   b_arg: remaining_charge_sample_time(sec) 60.0  #! simple=False

   b_arg: use_thruster_for_ascent(bool) 0.0            #! simple=False

                                    # If True, then use the thruster to maintain an emergency ascent depth
rate u_min_thruster_abort_ascent_rate


behavior: surface
```

```
b_arg: args_from_file(enum) -1              #! ignore=True

                                # >= 0 enables reading from mafiles/surfac<N>.ma

b_arg: start_when(enum)     12              #! choices=start_when([0, 1, 2, 3, 6, 7, 8, 9, 10, 11, 12, 13])

b_arg: when_secs(sec)     1200              #! min = 120.0

                                # How long between surfacing, only if start_when==6,9, or 12

b_arg: when_wpt_dist(m)  10                 #! min = 5.0

                                # how close to waypoint before surface, only if start_when==7

b_arg: end_action(enum) 1                   #! choices=end_action([0, 1, 2, 3, 4, 5])

      # 0-quit, 1-wait for ^C quit/resume, 2-resume, 3-drift til "end_wpt_dist"

                        # 4-wait for ^C once  5-wait for ^C quit on timeout


b_arg: report_all(bool) 0                   #! simple=False

                                # T->report all sensors once, F->just gps

b_arg: gps_wait_time(sec) 300               #! min = 120.0

                                # how long to wait for gps

b_arg: keystroke_wait_time(sec) 300         #! min = 0.0; max = 900.0

                                # how long to wait for control-C

b_arg: end_wpt_dist(m) 0                    #! min = 0.0

                                # end_action == 3   ==> stop when m_dist_to_wpt > this arg


                        # Arguments for climb_to when going to surface

b_arg: c_use_bpump(enum)      2             #! choices=use_bpump

b_arg: c_bpump_value(X)  1000.0             #! min = 0.0; max = 1000.0

b_arg: c_use_pitch(enum)      3             #! choices=use_pitch

b_arg: c_pitch_value(X)  0.4363             #! min = minPitch; max = maxPitch
```

b_arg: c_stop_when_air_pump(bool) 0 # Terminate climb once air pump has been inflated. For use with thruster only.

b_arg: c_use_thruster(enum) 0 # 0 Not in use

1 Command input voltage (as % of glider voltage)

2 Command input voltage (as % of max thruster input voltage)

3 Command depth rate. See sensors for use_thruster = depthrate

4 Command input power. See sensors for use_thruster = power

b_arg: c_thruster_value(X) 0.0 # use_thruster == 0 None

use_thruster == 1 %, desired % of glider voltage, between [0, 100] (will be clipped to F_THRUSTER_MAX_V)

use_thruster == 2 %, desired % of F_THRUSTER_MAX_V, between [0, 100]

use_thruster == 3 m/s, desired depth rate. <0 for climb

use_thruster == 4 watt, desired input power, between [f_thruster_power_min,max]

b_arg: printout_cycle_time(sec) 60.0 #! simple=False

How often to print dialog

iridium related stuff

b_arg: gps_postfix_wait_time(sec) 60.0 #! simple=False

How long to wait after initial

gps fix before turning the iridium

```
                                # on (which disables the gps).  It will
                                # wait the shorter of this time or until
                                # all the water velocity calculations are
                                # complete.

b_arg: force_iridium_use(nodim)  0.0          #! simple=False
                                #  Only for test.  non-zero values are set
                               # into C_IRIDIUM_ON.  Used to force the
                               # use of the iridium even if freewave is
                               # present.

b_arg: min_time_between_gps_fixes(sec)  300.0  #! simple=False
                                # The irdium will be hung up this often
                             # to get gps fixes.  It will call back however.
                             # Primarily for use in hold missions to get
                             # periodic gps fixes to tell how far the glider
                             # has drifted.

b_arg: sensor_input_wait_time(sec)  10.0      #! simple=False
                                # Time limit to wait for input sensors at surface.


                              # For when_utc
b_arg: when_utc_min(min)       -1           #! simple=False; min = -1; max = 59
                                # 0-59, otherwise any minute
b_arg: when_utc_hour(hour)     -1           #! simple=False; min = -1; max = 23
```

```
                                # 0-23, otherwise any hour
  b_arg: when_utc_day(day)      -1          #! simple=False; min = -1; max = 31
                                # 1-31, otherwise any day
  b_arg: when_utc_month(month)  -1          #! simple=False; min = -1; max = 12
                                # 1-12, otherwise any month
  b_arg: when_utc_on_surface(bool) 0        #! simple=False
                                # If true (>0), adjust when_utc_month/day/hour/min
                        # to get glider on surface by this time.

  b_arg: strobe_on(bool)      0           # Behavior argument to control the strobe light
  b_arg: thruster_burst(bool)     1       # Behavior argument to turn thruster on
                        #If thruster/thruster_g1 is in use, turn on thruster to remove buildup before
getting final GPS fix before diving.
                        #See 'u_thruster_burst_' sensors
behavior: goto_wpt                      #! visible = False
  b_arg: start_when(enum) 0             #! choices=start_when([0, 1, 2, 4])
  b_arg: stop_when(enum)  2             #! choices=stop_when([1, 2, 5, 7])

  b_arg: when_wpt_dist(m) 0             #! min = 5.0
                                # stop_when == 7   ==> stop when m_dist_to_wpt < this arg

  b_arg: wpt_units(enum)  0             #! choices=wpt_units
  b_arg: wpt_x(X)       0   # The waypoint (east or lon)
  b_arg: wpt_y(X)       0   #           (north or lat)
                      # These only used for UTM waypoints
  b_arg: utm_zd(byte)   19.0  #   UTM Zone as digit (see coord_sys.h)
```

```
   b_arg: utm_zc(byte)   19.0   # (T) UTM Zone as char (see coord_sys.h)


   b_arg: end_action(enum) 0                  #! choices=end_action([0, 1, 2, 3, 4, 5])


behavior: goto_list
   b_arg: args_from_file(enum) -1             #! ignore=True
                                 # >= 0 enables reading from mafiles/goto_l<N>.ma
   b_arg: start_when(enum)     0   # See doco above


   b_arg: num_waypoints(nodim)  0             #! min = 1; max = 8
                                 # Number of valid waypoints in list
                               # maximum of 8 (this can be increased at compile-time)
   b_arg: num_legs_to_run(nodim) -1           #! min = -2
                                 # Number of waypoints to sequence thru:
                               #  1-N    exactly this many waypoints
                               #  0      illegal
                               # -1      loop forever
                               # -2      traverse list once (stop at last in list)
                               # <-2     illegal


   b_arg: initial_wpt(enum)     -2            #! min = -2; max = 7
                                 # Which waypoint to head for first
                               #  0 to N-1 the waypoint in the list
                               # -1 ==> one after last one achieved
                               # -2 ==> closest
```

```
# Stopping condition applied to all of waypoints in the list
b_arg: list_stop_when(enum)  7              #! choices = stop_when([1, 2, 5, 7])
                                # See doco above
b_arg: list_when_wpt_dist(m) 10.            #! min = 10.0
                                # used if list_stop_when == 7

# When behavior is complete, either quit or stay active waiting for new mafile
b_arg: end_action(enum)  0                  #! choices = end_action([0, 6])
                                # 0-quit, 6-wait for ^F (re-read mafiles)

# The waypoints: Control center does not edit coordinates in UTM or LMC
b_arg: wpt_units_0(enum) 0                  #! ignore=True
b_arg: wpt_x_0(X)       0       # The waypoint (east or lon)
b_arg: wpt_y_0(X)       0       #            (north or lat)

b_arg: wpt_units_1(enum) 0                  #! ignore=True
b_arg: wpt_x_1(X)       0
b_arg: wpt_y_1(X)       0

b_arg: wpt_units_2(enum) 0                  #! ignore=True
b_arg: wpt_x_2(X)       0
b_arg: wpt_y_2(X)       0
```

```
    b_arg: wpt_units_3(enum) 0              #! ignore=True
    b_arg: wpt_x_3(X)       0
    b_arg: wpt_y_3(X)       0

    b_arg: wpt_units_4(enum) 0              #! ignore=True
    b_arg: wpt_x_4(X)       0
    b_arg: wpt_y_4(X)       0

    b_arg: wpt_units_5(enum) 0              #! ignore=True
    b_arg: wpt_x_5(X)       0
    b_arg: wpt_y_5(X)       0

    b_arg: wpt_units_6(enum) 0              #! ignore=True
    b_arg: wpt_x_6(X)       0
    b_arg: wpt_y_6(X)       0

    b_arg: wpt_units_7(enum) 0              #! ignore=True
    b_arg: wpt_x_7(X)       0
    b_arg: wpt_y_7(X)       0


behavior: yo
    b_arg: args_from_file(enum) -1           #! ignore=True
                                  # >= 0 enables reading from mafiles/yo<N>.ma
    b_arg: start_when(enum)      2           #! choices=start_when([0, 1, 2, 4])
```

```
b_arg: start_diving(bool)    1   # T-> dive first, F->climb first

b_arg: num_half_cycles_to_do(nodim) -1        #! min = -1

                                    # Number of dive/climbs to perform

                            # <0 is infinite, i.e. never finishes


# arguments for dive_to

b_arg: d_target_depth(m)     12               #! min = 3.0; max = 1000.0

b_arg: d_target_altitude(m)   5              #! min = -1; max = 100.0

b_arg: d_use_bpump(enum)      2              #! choices=use_bpump

b_arg: d_bpump_value(X) -1000.0              #! min = -1000; max = 1000

b_arg: d_use_pitch(enum)      3              #! choices=use_pitch

b_arg: d_pitch_value(X) -0.4538              #! min = -maxPitch; max = -minPitch

b_arg: d_stop_when_hover_for(sec) 180.0       #! simple=False

b_arg: d_stop_when_stalled_for(sec) 240.0     #! simple=False

b_arg: d_speed_min(m/s)       -100.0               #! simple = False; min = -100.0; max = 0.3

b_arg: d_speed_max(m/s)       100.0               #! simple = False; min = 0.05; max = 100.0

b_arg: d_use_thruster(enum)   0             #! choices = use_thruster

b_arg: d_thruster_value(X)   0.0            #! min = minThruster; max = maxThruster

b_arg: d_depth_rate_method(enum) 3           #! simple = False; choices = depth_rate_method

b_arg: d_wait_for_pitch(bool) 1              #! simple = False

b_arg: d_wait_for_ballast(sec) 100.0         #! simple = False

b_arg: d_delta_bpump_speed(X) 50.0           #! simple = False; min = 10.0; max = 100.0

b_arg: d_delta_bpump_ballast(X) 25.0         #! simple = False; min = 10.0; max = 100.0

b_arg: d_time_ratio(X) 1.1                          #! simple = False; min = 0.0; max = 2.0

b_arg: d_use_sc_model(bool) 0                #! simple = False
```

```
    b_arg: d_max_thermal_charge_time(sec) 1200.0   #! simple = False
    b_arg: d_max_pumping_charge_time(sec) 300.0    #! simple = False
    b_arg: d_thr_reqd_pres_mul(nodim) 1.50        #! simple = False

    # arguments for climb_to
    b_arg: c_target_depth(m)      3              #! min = 3.0; max = 1000.0
    b_arg: c_target_altitude(m)  -1              #! simple = False
    b_arg: c_use_bpump(enum)      2              #! choices = use_bpump
    b_arg: c_bpump_value(X)  1000.0              #! min = -1000.0; max = 1000.0
    b_arg: c_use_pitch(enum)      3              #! choices = use_pitch
    b_arg: c_pitch_value(X)  0.4538              #! min = minPitch; max = maxPitch
    b_arg: c_stop_when_hover_for(sec) 180.0       #! simple=False
    b_arg: c_stop_when_stalled_for(sec) 240.0     #! simple=False
    b_arg: c_speed_min(m/s)       100.0           #! min = -0.3; max = 100.0
    b_arg: c_speed_max(m/s)       -100.0          #! min = -100.0; max = 0.05
    b_arg: c_use_thruster(enum)   0              #! choices = use_thruster
    b_arg: c_thruster_value(X)   0.0           #! min = minPitch; max = maxPitch

    b_arg: end_action(enum)       2              #! choices = end_action([0, 2])


behavior: prepare_to_dive
    b_arg: args_from_file(enum) -1               #! ignore = True
                                    # >= 0 enables reading from mafiles/prepar<N>.ma
```

```
    b_arg: start_when(enum) 0                   #! choices = start_when([0, 1, 2])
    b_arg: wait_time(sec) 720     # 12minutes, how long to wait for gps
    b_arg: max_thermal_charge_time(sec) 120       #! simple = False
                                      # The maximum length of time to wait for
                                    # charge from thermal tubes.  After this time the
                                    # electric charge pump is used.
    b_arg: max_pumping_charge_time(sec) 1000      #! simple = False
                                      # The maximum length of time to wait for a charge
                                  # after using electric c charge pump.
                                  # max time to wait = max_thermal_charge_time +
                                  #                 max_pumping_charge_time


behavior: sensors_in                          #! visible = False
    # <0 off, 0 as fast as possible, N, sample every N secs


# Glider sensors
    b_arg: c_att_time(sec)         -1.0
    b_arg: c_pressure_time(sec)     -1.0
    b_arg: c_alt_time(sec)         -1.0
    b_arg: u_battery_time(sec)      -1.0
    b_arg: u_vacuum_time(sec)       -1.0
    b_arg: c_leakdetect_time(sec)   -1.0
    b_arg: c_gps_on(bool)           0.0  # Special, 1 is on, 0 is off
```

```
# Science sensors start here
   b_arg: c_science_all_on(sec)    -1.0
   b_arg: c_profile_on(sec)        -1.0
#    b_arg: c_hs2_on(sec)           -1.0 removed
   b_arg: c_bb2f_on(sec)           -1.0
   b_arg: c_bb2c_on(sec)           -1.0
   b_arg: c_bb2lss_on(sec)         -1.0
   b_arg: c_sam_on(sec)            -1.0
#    b_arg: c_whpar_on(sec)          -1.0 removed
#    b_arg: c_whgpbm_on(sec)         -1.0 removed
   b_arg: c_moteopd_on(sec)        -1.0
   b_arg: c_bbfl2s_on(sec)         -1.0
   b_arg: c_fl3slo_on(sec)         -1.0
   b_arg: c_bb3slo_on(sec)         -1.0
#    b_arg: c_oxy3835_on(sec)        -1.0 removed
   b_arg: c_whfctd_on(sec)         -1.0
   b_arg: c_bam_on(sec)            -1.0
   b_arg: c_ocr504R_on(sec)        -1.0
   b_arg: c_ocr504I_on(sec)        -1.0
#           b_arg: c_badd_on(sec)                   -1.0 removed
   b_arg: c_flntu_on(sec)          -1.0
   b_arg: c_fl3slov2_on(sec)       -1.0
   b_arg: c_bb3slov2_on(sec)       -1.0
   b_arg: c_ocr507R_on(sec)        -1.0
   b_arg: c_ocr507I_on(sec)        -1.0
```

```
   b_arg: c_bb3slov3_on(sec)      -1.0
   b_arg: c_bb2fls_on(sec)        -1.0
   b_arg: c_bb2flsV2_on(sec)      -1.0
   b_arg: c_oxy3835_wphase_on(sec) -1.0
   b_arg: c_auvb_on(sec)          -1.0
   b_arg: c_bb2fV2_on(sec)        -1.0
   b_arg: c_tarr_on(sec)          -1.0
   b_arg: c_bbfl2sV2_on(sec)      -1.0
#   b_arg: c_glbps_on(sec)         -1.0 removed
   b_arg: c_sscsd_on(sec)         -1.0
   b_arg: c_bb2flsV3_on(sec)      -1.0
   b_arg: c_fire_on(sec)          -1.0
#   b_arg: c_ohf_on(sec)           -1.0  removed
   b_arg: c_bb2flsV4_on(sec)      -1.0
   b_arg: c_bb2flsV5_on(sec)      -1.0
   b_arg: c_logger_on(sec)        -1.0
   b_arg: c_bbam_on(sec)          -1.0
   b_arg: c_uModem_on(sec)        -1.0
   b_arg: c_rinkoII_on(sec)       -1.0
   b_arg: c_dvl_on(sec)           -1.0
   b_arg: c_bb2flsV6_on(sec)      -1.0
   b_arg: c_flbbrh_on(sec)        -1.0
   b_arg: c_flur_on(sec)          -1.0
   b_arg: c_bb2flsV7_on(sec)      -1.0
   b_arg: c_flbbcd_on(sec)        -1.0
```

```
    b_arg: c_dmon_on(sec)          -1.0
    b_arg: c_c3sfl_on(sec)         -1.0
    b_arg: c_suna_on(sec)          -1.0
    b_arg: c_satpar_on(sec)        -1.0
    b_arg: c_vsf_on(sec)           -1.0
    b_arg: c_oxy4_on(sec)          -1.0
    b_arg: c_gamma_rad5_on(sec)    -1.0
    b_arg: c_bsipar_on(sec)        -1.0
    b_arg: c_flbb_on(sec)          -1.0
    b_arg: c_vr2c_on(sec)          -1.0
    b_arg: c_ctd41cp2_on(sec)      -1.0
    b_arg: c_echosndr853_on(sec)   -1.0
            b_arg: c_tritech_on(sec)                -1.0
# <PROTO> Add additional science proglets here



# ----- This is end of a typical mission



# These usually do not get called directly
behavior: set_heading
    b_arg: use_heading(bool) 2              #! choices=set_heading
                                    # in, 1 HM_HEADING
                        # in, 2 HM_ROLL
                        # in, 3 HM_BATTROLL
```

```
                    # in, 4 HM_FIN


   b_arg: heading_value(X) 1000.0

                    # use_heading == 1 C_HEADING(rad) desired heading

                    # use_heading == 2 C_ROLL(rad),    >0 bank right

                    # use_heading == 3 C_BATTROLL(rad) >0 puts stbd wing down

                    # use_heading == 4 C_FIN(rad),     >0 turns to stbd


   b_arg: start_when(enum) 0     # See doco above

   b_arg: stop_when(enum)  2     # See doco above

   b_arg: when_secs(sec)  1200   # only if start_when==6,9, or 12




behavior: dive_to                       #! visible = False

   b_arg: target_depth(m) 10    # how deep to dive

   b_arg: target_altitude(m) -1 # stop this far from bottom, <0 disables


                    # bpump_mode_t values - ballast control

                    # Electric only, ignored in thermal

   b_arg: use_bpump(enum) 2     # 0  Autoballast/Speed control.  See doco/how-it-works/autoballast.txt

                    # 1  Reserved - do not use (relative to neutral)

                    # 2  Buoyancy Pump absolute

   b_arg: bpump_value(X) -1000.0 # use_bpump == 0   Total amt of ballast.  Stored as
C_AUTOBALLAST_VOLUME
```

```
                    # use_bpump == 1   cc, clips to max legal, >0 goes up
                    # use_bpump == 2   cc, clips to max legal  >0 goes up

                    # pitch_mode_t values - battery or fluid fore/aft control
  b_arg: use_pitch(enum) 1     # 4  Fluid Pumped absolute
                    # 3  Servo on Pitch
                    # 2  Pitch, set once from curve
                    # 1  BattPos
  b_arg: pitch_value(X)  0.0   # use_pitch == 4    cc, clips to max legal, >0 to nose down
                    # use_pitch == 2,3  rad, desired pitch angle, <0 to dive
                    # use_pitch == 1    in,  desired battpos, >0 to nose down
                    #                   clips to max legal

  b_arg: start_when(enum) 0     # See doco above
  b_arg: stop_when_hover_for(sec) 180.0 # terminate dive when depth does not change for
                          # this many secs, <0 to disable
  b_arg: stop_when_stalled_for(sec) 240.0 # terminate dive when glider not moving thru water
                           # this many secs, i.e. M_SPEED is 0
                           # <0 to disable
  b_arg: stop_when_air_pump(bool) 0 # Ignored for dives, here as a placeholder

  b_arg: initial_inflection(bool) 1.0  # T->Want to start with an inflection
                                                                  #  Autoballast only (use_bpump == 0).
Ignored for other bpump modes.
  b_arg: speed_min(m/s) -100.0          #vertical minimum dive speed for speed control SM_SPEED
```

#User should set this to a positive value (dive rate > 0), otherwise, it gets set to default f_speed_min

b_arg: speed_max(m/s) 100.0 #vertical maximum dive speed for speed control SM_SPEED

b_arg: use_thruster(enum) 0 # 0 Not in use

1 Command input voltage (as % of glider voltage)

2 Command input voltage (as % of max thruster input voltage)

3 Command depth rate. See sensors for use_thruster = depthrate

4 Command input power. See sensors for use_thruster = power

b_arg: thruster_value(X) 0.0 # use_thruster == 0 None

use_thruster == 1 %, desired % of glider voltage, between [0, 100] (will be clipped to F_THRUSTER_MAX_V)

use_thruster == 2 %, desired % of F_THRUSTER_MAX_V, between [0, 100]

use_thruster == 3 m/s, desired depth rate. >0 for dive

use_thruster == 4 watt, desired input power, between [f_thruster_power_min,max]

b_arg: depth_rate_method(enum) 3 #method of filtered depth rate to use for speed control

0= raw m_depth_rate

1= m_depth_rate_subsample

2 = tbd.

3 = running average. (see description of m_depth_rate_avg_final)

b_arg: wait_for_pitch(bool) 1 #if true, wait for pitch/batt pos dynamics to settle before enabling speed control.

b_arg value stored as c_wait_for_pitch

b_arg: wait_for_ballast(sec) 100.0 # wait this many seconds after ballast pump has stopped moving (inflection only) before

#enabling speed control. b_arg value stored as c_wait_for_ballast

b_arg: delta_bpump_speed(X) 50.0 #amount of ballast to add to bpump in order to reach desired speed. Should always be positive.

b_arg value stored as c_delta_bpump_ballast.

b_arg: delta_bpump_ballast(X) 25.0 #amount of ballast to add to bpump in order to converge on ballast.

If < deadband, diveclimb.c will set it to deadband. b_arg value stored as c_delta_bpump_speed.

b_arg: time_ratio(X) 1.1 #ratio of climb/dive times that must be maintained for speed control.

b_arg: use_sc_model(bool) 0 #if using model of veh for SM_SPEED. Always set to 0 for now until the model is designed

Thermal only, ignored in electric

b_arg: max_thermal_charge_time(sec) 1200.0 # How long to wait for thermal

charge before using the thermal pump

b_arg: max_pumping_charge_time(sec) 300.0 # how long to wait after starting charge pump

before an error

b_arg: thr_reqd_pres_mul(nodim) 1.50 # engine pressure must be this many

times the ocean pressure at target_depth

before the dive is started.

```
behavior: climb_to                      #! visible = False
   b_arg: target_depth(m) 10    # how deep to dive
   b_arg: target_altitude(m) -1 # stop this far from bottom, <0 disables


                        # bpump_mode_t values - ballast control
   b_arg: use_bpump(enum) 2     # 0  Autoballast/Speed control.  See doco/how-it-works/autoballast.txt
                        # 1  Buoyancy Pump relative to neutral
                        # 2  Buoyancy Pump absolute
   b_arg: bpump_value(X) 1000.0 # use_bpump == 0: c_bpump_value is ignored.  Dive value
d_bpump_value stored
                        # as C_AUTOBALLAST_VOLUME
                        # use_bpump == 1   cc, clips to max legal, >0 goes up
                        # use_bpump == 2   cc, clips to max legal  >0 goes up


                        # pitch_mode_t values - battery for fluid fore/aft control
   b_arg: use_pitch(enum) 1     # 4  Fluid Pumped absolute
                        # 3  Servo on Pitch
                        # 2  Pitch, set once from curve
                        # 1  BattPos
   b_arg: pitch_value(X)  0.0   # use_pitch == 4    cc, clips to max legal, >0 to nose down
                        # use_pitch == 2,3  rad, desired pitch angle, <0 to dive
                        # use_pitch == 1    in,  desired battpos, >0 to nose down
                        #                   clips to max legal
```

b_arg: start_when(enum) 0 # See doco above

b_arg: stop_when_hover_for(sec) -1.0 # terminate dive when depth does not change for

this many secs, <0 to disable

b_arg: stop_when_stalled_for(sec) 240.0 # terminate climb when glider not moving thru water

this many secs, i.e. M_SPEED is 0

<0 to disable

b_arg: stop_when_air_pump(bool) 0 # terminate climb once air pump has been inflated. See M_VACUUM_CHANGE_SINCE_AIR_PUMP_ON

Only used in surface behaviors. If set to True for any other behaviors,

argument will be ignored.

b_arg: initial_inflection(bool) 1.0 # T->Want to start with an inflection

Autoballast only (use_bpump == 0). Ignored for other bpump modes.

b_arg: speed_min(m/s) 100.0 #vertical minimum dive speed for speed control SM_SPEED.

#User should set this to a negative value (climb rate < 0), otherwise, it gets set to -f_speed_min

b_arg: speed_max(m/s) -100.0 #vertical maximum dive speed for speed control SM_SPEED

b_arg: use_thruster(enum) 0 # 0 Not in use

1 Command input voltage (as % of glider voltage)

2 Command input voltage (as % of max thruster input voltage)

3 Command depth rate. See sensors for use_thruster = depthrate

```
                                                                    # 4  Command
input power. See sensors for use_thruster = power

   b_arg: thruster_value(X)   0.0            # use_thruster == 0  None

                                                                    # use_thruster
== 1  %, desired % of glider voltage, between [0, 100] (will be clipped to F_THRUSTER_MAX_V)

                                                                    # use_thruster
== 2  %, desired % of F_THRUSTER_MAX_V, between [0, 100]

                                                                    # use_thruster
== 3  m/s, desired depth rate. < 0 for climb

                                                                    # use_thruster
== 4  watt, desired input power, between [f_thruster_power_min,max]


behavior: drift_at_depth

   b_arg: args_from_file(enum) -1              #! ignore = True

   b_arg: start_when(enum) 4                  #! choices=start_when([0, 1, 2, 3, 4, 6, 7, 8, 9, 10, 13])

   b_arg: when_secs(sec)   180    # For start_when = 6, 9, or 10

   b_arg: when_wpt_dist(m) 10                 # ! min = 5.0


   b_arg: when_utc_min(min)      -1  # 0-59, -1 any minute

   b_arg: when_utc_hour(hour)    -1  # 0-23, -1 any hour

   b_arg: when_utc_day(day)      -1  # 1-31, -1 any day

   b_arg: when_utc_month(month)  -1  # 1-12, -1 any month


   b_arg: end_action(enum) 0      # 0-quit, 2-resume

   b_arg: stop_when_hover_for(sec) 3600.0 # terminate hover when depth does not change for

                                  # this many secs, <0 to disable

   b_arg: est_time_to_settle(s) 360.0 # Used to force invalid cc_time_til_inflect for this
```

This many seconds at the beginning of the behavior.

b_arg: target_depth(m) 100.0 # depth to drift at

b_arg: target_altitude(m) 20.0 # altitude to drift at, <=0 disables

b_arg: alt_time(s) 120.0 # time spacing for altimeter pings

<0 is off, =0 as fast as possible

>0 that many seconds betweens measurements

b_arg: target_deadband(m) 5.0 # +/- around target depth or altitude

b_arg: start_dist_from_target(m) -1.0 # start the drift this distance from the target depth/altitude. -1 means use the target_deadband

b_arg: depth_ctrl(enum) 0 # 0: Default mode for buoyancy drive only drift_at_depth. increment by bpump_delta_value

1: servo bpump on depth: see 'bpump servo mode' sensors

2: Recommended mode for thruster control. pitch-based depth control: see 'pitching depth control mode' sensors

b_arg: bpump_delta_value(cc) 1.0 # Increments to adjust x_hover_ballast(cc) to obtain

neutral buoyancy (>= 1 cc)

b_arg: bpump_delay(s) 0.0 # Minimum time between making buoyancy adjustments

b_arg: bpump_deadz_width(cc) 30.0 # For temporarily adjusting the buoyancy pump

b_arg: bpump_db_frac_dz(nodim) 0.1 # deadband during the drift_at_depth behavior

pitch_mode_t values - battery or fluid fore/aft control

b_arg: use_pitch(enum) 1 # 4 Fluid Pumped absolute

3 Servo on Pitch

2 Pitch, set once from curve

1 BattPos

b_arg: pitch_value(X) 0.0 # use_pitch == 4 cc, clips to max legal, >0 to nose down

use_pitch == 2,3 rad, desired pitch angle, <0 to dive

use_pitch == 1 in, desired battpos, >0 to nose down

clips to max legal

b_arg: wait_for_pitch(bool) 0 # if true, we only adjust x_hover_ballast if pitch is within deadband (for use_pitch = 2 or 3)

b_arg: use_thruster(enum) 0 # 0 Not in use

1 Command input voltage (as % of glider voltage)

2 Command input voltage (as % of max thruster input voltage)

3 N/A for this behavior

4 Command input power. See sensors for use_thruster = power

b_arg: thruster_value(X) 0.0 # use_thruster == 0 None

use_thruster == 1 %, desired % of glider voltage, between [0, 100] (will be clipped to F_THRUSTER_MAX_V)

use_thruster == 2 %, desired % of F_THRUSTER_MAX_V, between [0, 100]

3 N/A for this behavior

use_thruster == 4 watt, desired input power, between [f_thruster_power_min,max]

b_arg: enable_steering(bool) 0 # Enable or disable steering while hovering. If True, heading is

controlled as normal (set_heading, goto_list etc) during hovering. If False,

commanded fin position = 0 during hovering.

Arguments for dive_to when diving to hover zone

b_arg: d_use_bpump(enum) 2

```
   b_arg: d_bpump_value(X)  -1000.0
   b_arg: d_use_pitch(enum)      3  # servo on pitch
   b_arg: d_pitch_value(X)  -0.4538  # ~-26 degrees
   b_arg: d_use_thruster(enum)   0            # 0  Not in use
                                                                    # 1  Command
input voltage (as % of glider voltage)
                                                                    # 2  Command
input voltage (as % of max thruster input voltage)
                                                                    # 3  Command
depth rate.  See sensors for use_thruster = depthrate
                                                                    # 4  Command
input power. See sensors for use_thruster = power

   b_arg: d_thruster_value(X)   0.0           # use_thruster == 0  None
                                                                    # use_thruster
== 1  %, desired % of glider voltage, between [0, 100] (will be clipped to F_THRUSTER_MAX_V)
                                                                    # use_thruster
== 2  %, desired % of F_THRUSTER_MAX_V, between [0, 100]
                                                                    # use_thruster
== 3  m/s, desired depth rate. >0 for dive
                                                                    # use_thruster
== 4  watt, desired input power, between [f_thruster_power_min,max]


          # Arguments for climb_to when climbing to hover zone
   b_arg: c_use_bpump(enum)      2
   b_arg: c_bpump_value(X)   1000.0
   b_arg: c_use_pitch(enum)      3  # servo on pitch
   b_arg: c_pitch_value(X)   0.4538  # ~26 degrees
```

```
   b_arg: c_use_thruster(enum)   0            # 0  Not in use
                                                                                  # 1  Command
input voltage (as % of glider voltage)
                                                                                  # 2  Command
input voltage (as % of max thruster input voltage)
                                                                                  # 3  Command
depth rate.  See sensors for use_thruster = depthrate
                                                                                  # 4  Command
input power. See sensors for use_thruster = power

   b_arg: c_thruster_value(X)   0.0           # use_thruster == 0  None
                                                                                  # use_thruster
== 1  %, desired % of glider voltage, between [0, 100] (will be clipped to F_THRUSTER_MAX_V)
                                                                                  # use_thruster
== 2  %, desired % of F_THRUSTER_MAX_V, between [0, 100]
                                                                                  # use_thruster
== 3  m/s, desired depth rate. <0 for climb
                                                                                  # use_thruster
== 4  watt, desired input power, between [f_thruster_power_min,max]


          # Arguments for pitch depth controller (depth_ctrl = 2 )
   b_arg: depth_pitch_limit(rad) 0.174      #limit pitch response to 25 deg
   b_arg: depth_gain_scale(bool) 1 # whether or not to use X_HOVER_DEPTH_P_GAIN = m * speed + b

   b_arg: depth_p_gain(X) -0.15             #proportional gain: should always be < 0.  See
X_HOVER_DEPTH_P_GAIN
   b_arg: depth_i_gain(X) -0.0001                                  #integral gain: should be < 0
   b_arg: depth_d_gain(X) 0.1                                 #derivative gain: should be > 0
   b_arg: depth_pitch_deadband(m/s) 0.0349     #don't adjust bouyancy until depth rate is less than this
```

b_arg: depth_pitch_max_time(s) 60 # Max time at maximum u_hover_depth_pitch_limit before we start to adjust ballast

behaviors which control/communicate with the science computer

bconsci

A terminal session with the glider.

Stops by loss of carrier. Package with abend

to stop by time/depth

behavior: bconsci #! visible = False

b_arg: terminate_mission_when_done(bool) 1 # end mission when this behavior is done

Controls the sampling of the hydrophones

Obsolete, should be removed

replaced by behavior: bhydrophone

behavior: hydrosmp #! visible = False

b_arg: args_from_file(enum) -1 # >= 0 enables reading from mafiles/hydros<N>.ma

#

b_arg: num_samples(nodim) 1 # How many collections, -1 runs forever

b_arg: time_between_samples(min) 10 # wait time between samples

controls initial start minute sych to hour

b_arg: duration(sec) 30 # How long each sample is

b_arg: gain(dB) 0 # 0, 5, 10, .., 35

b_arg: channel(nodim) 0 # 0-3, which channel

```
   b_arg: xmit_files(bool)          0  # t-> have science xmit files

   b_arg: silence_lvl(nodim)        0  # 0-1, higher the number, the quieter the glider


   b_arg: idle_stack_when_done(bool) 1  # T-> idle the stack to terminate

                                #     mission when sampling is done


   # bhydrophone

   # A behavior to control the superscience (quest-2003) version of

   # drea hydrophone sampling

behavior: bhydrophone                        #! visible = False

   b_arg: args_from_file(enum) -1     # >= 0 enables reading from mafiles/bhydro<N>.ma


   b_arg: start_when(enum) 0         # See doco above: 0, 9

   b_arg: when_secs(sec)   0         #


                                # Behavior ends when either of these conditions met

   b_arg: max_collection_time(sec) -1 # Collect for this long maximum, <0 ==> forever

   b_arg: num_collections(nodim)   -1 # Number of collections to make, <0 ==> infinite


   # Timing of collection


   b_arg: c_hydrophone_duration(sec)   60.0 # How long to collect

   b_arg: c_hydrophone_pre_delay(sec)  15.0 # Delay between proglet start and collection

   b_arg: no_sample_time(sec)          15.0 # Time between collection

                                # c_hydrophone_pre_delay+no_sample_time is total
```

```
                                # time when NOT sampling


    b_arg: c_hydrophone_post_delay(sec) 30.0 # How long before proglet recycles
                                # This is not part of duty cycle
                                # only how long before proglet recycles
                                # It is normally stopped after every measurement

    # Ping control
    b_arg: c_hydrophone_pre_pings(nodim)   1  # number of pings before sample
    b_arg: c_hydrophone_post_pings(nodim)  2  # number of pings after sample



    # Collection parameters
    b_arg: c_hydrophone_gain(nodim)        3.0 # 0-7
    b_arg: c_hydrophone_num_channels(nodim) 1.0 # 1-4
    b_arg: c_hydrophone_sample_rate(Hz)  5000.0 # 1000-5000, how fast to AD
    b_arg: c_hydrophone_drive_num(nodim)  3.0   # 2->C:, 3:->D: etc


    # bviper
    # A behavior to control the DMA Viper processor
behavior: bviper                        #! visible = False
    b_arg: args_from_file(enum) -1     # >= 0 enables reading from mafiles/bhydro<N>.ma


    b_arg: start_when(enum) 0         # See doco above: 0, 9, 13
    b_arg: when_secs(sec)   0         #
```

b_arg: when_utc_min(min) -1 # 0-59, -1 any minute

b_arg: when_utc_hour(hr) -1 # 0-23, -1 any hour

b_arg: when_utc_day(day) -1 # 1-31, -1 any day

b_arg: when_utc_month(mon) -1 # 1-12, -1 any month

Behavior ends when either of these conditions met

b_arg: max_collection_time(sec) -1 # Collect for this long maximum, <0 ==> forever

b_arg: num_collections(nodim) -1 # Number of collections to make, <0 ==> infinite

Timing of collection

b_arg: no_sample_time(sec) 300.0 # Time between collection

time when NOT sampling

Collection parameters

b_arg: c_viper_turn_on_timeout(sec) 120.0 # max wait time for viper to power on

b_arg: c_viper_collect_timeout(sec) 200.0 # max wait time for viper to collect/analyse acoustic data

b_arg: c_viper_reset_timeout(sec) 60.0 # max wait time for viper to respond to reset gain command

b_arg: c_viper_start_sampling_timeout(sec) 60.0 # max wait time for viper to respond to start sampling command

b_arg: c_viper_detection_done_timeout(sec) 60.0 # max wait time for viper to respond to detection done command

b_arg: c_viper_turn_off_timeout(sec) 120.0 # max wait time for viper to power off

b_arg: c_viper_gain(nodim) 3.0 # 0-7 gain sent to viper

b_arg: c_viper_max_sample_starts(nodim) 3.0 # max allowable attempts to obtain a definitive detection

b_arg: c_viper_max_errors(nodim) 3.0 # max number of viper errors before mission abort

Added at sea (that's why out of order)

b_arg: min_sample_depth(m) 20 # min depth to start, <0 disables

b_arg: max_sample_depth(m) 60 # max depth to start, <0 disables

b_arg: min_reqd_quiet_time(s) 480 # must be < cc_final_time_to_inflect before start. Set <0 to disable

b_arg: post_inflection_holdoff(s) 60 # must be this long since inflection, < 0 disables

b_arg: allow_sample_at_surface(bool) 0 # non-zero allows sample at surface

Controls the sampling of specified sensor type (b_arg: sensor_type)

behavior: sample

b_arg: args_from_file(enum) -1 #! ignore=True

>= 0 enables reading from mafiles/sample<N>.ma

b_arg: sensor_type(enum) 0 #! choices = sensor_type()

ALL 0 C_SCIENCE_ALL_ON

PROFILE 1 C_PROFILE_ON

BB2F 2 C_BB2F_ON

BB2C 3 C_BB2C_ON

BB2LSS 4 C_BB2LSS_ON

SAM 5 C_SAM_ON

MOTEOPD 6 C_MOTEOPD_ON

BBFL2S 7 C_BBFL2S_ON

FL3SLO 8 C_FL3SLO_ON

BB3SLO 9 C_BB3SLO_ON

WHFCTD 10 C_WHFCTD_ON

BAM 11 C_BAM_ON

OCR504R 12 C_OCR504R_ON

OCR504I 13 C_OCR504I_ON

BADD 14 C_BADD_ON

FLNTU 15 C_FLNTU_ON

FL3SLOV2 16 C_FL3SLOV2_ON

BB3SLOV2 17 C_BB3SLOV2_ON

OCR507R 18 C_OCR507R_ON

OCR507I 19 C_OCR507I_ON

BB3SLOV3 20 C_BB3SLOV3_ON

BB2FLS 21 C_BB2FLS_ON

BB2FLSV2 22 C_BB2FLSV2_ON

OXY3835_WPHASE 23 C_OXY3835_WPHASE_ON

AUVB 24 C_AUVB_ON

BB2FV2 25 C_BB2FV2_ON

TARR 26 C_TARR_ON

BBFL2SV2 27 C_BBFL2SV2_ON

SSCSD 28 C_SSCSD_ON

BB2FLSV3 29 C_BB2FLSV3_ON

FIRE 30 C_FIRE_ON

BB2FLSV4 31 C_BB2FLSV4_ON

BB2FLSV5 32 C_BB2FLSV5_ON

LOGGER 33 C_LOGGER_ON

BBAM 34 C_BBAM_ON

```
# UMODEM     35  C_UMODEM_ON

# RINKOII    36  C_RINKOII_ON

# DVL        37  C_DVL_ON

# BB2FLSV6   38  C_BB2FLSV6_ON

# FLBBRH     39  C_FLBBRH_ON

# FLUR       40  C_FLUR_ON

# BB2FLSV7   41  C_BB2FLSV7_ON

# FLBBCD     42  C_FLBBCD_ON

# DMON       43  C_DMON_ON

# C3SFL      44  C_C3SFL_ON

# SUNA       45  C_SUNA_ON

# SATPAR     46  C_SATPAR_ON

# VSF        47  C_VSF_ON

# OXY4       48  C_OXY4_ON

# GAMMA_RAD5 49  C_GAMMA_RAD5_ON

# BSIPAR     50  C_BSIPAR_ON

# FLBB       51  C_FLBB_ON

# VR2C       52  C_VR2C_ON

# CTD41CP2   53  C_CTD41CP2_ON

# ECHOSNDR853 54 C_ECHOSNDR853_ON

# TRITECH    55  C_TRITECH_ON


# <PROTO> pick next number here for new proglet

#  REQUIRED: also add it to: science_super.c: __ss_indexes[],

#  add it to output_sensors[] in snsr_in.c,
```

```
                                # and update header doco in sample.c.




                                # This is a bit-field, combine:

                                # 8 on_surface, 4 climbing, 2 hovering, 1 diving
b_arg: state_to_sample(enum)          1     #! choices = state_to_sample

                                  # 0  none

                                # 1  diving

                                # 2  hovering

                                # 3  diving|hovering

                                # 4  climbing

                                # 5  diving|climbing

                                # 6  hovering|climbing

                                # 7  diving|hovering|climbing

                                # 8  on_surface

                                # 9  diving|on_surface

                                # 10 hovering|on_surface

                                # 11 diving|hovering|on_surface

                                # 12 climbing|on_surface

                                # 13 diving|climbing|on_surface

                                # 14 hovering|climbing|on_surface

                                # 15 diving|hovering|climbing|on_surface


b_arg: sample_time_after_state_change(s) 15    #! simple = False; min = 0.0
```

time after a positional stat

change to continue sampling

b_arg: intersample_time(s) 0 # if < 0 then off, if = 0 then # as fast as possible, and if

as fast as possible, and if

> 0 then that many seconds

between measurements

b_arg: nth_yo_to_sample(nodim) 1 #! min = 1.0

After the first yo, sample only

on every nth yo. If argument is

negative then exclude first yo.

b_arg: intersample_depth(m) -1 #! min = -1.0

supersedes intersample_time

by dynamically estimating

and setting intersample_time

to sample at the specified

depth interval. If <=0 then

then sample uses

intersample_time, if > 0 then

that many meters between

measurements

b_arg: min_depth(m) -5 #! min = -5; max = 1000.0

```
                                    # minimum depth to collect data, default
                                    # is negative to leave on at surface in
                                    # spite of noise in depth reading
   b_arg: max_depth(m)              2000    #! min = -5.0; max = 2000
                                    # maximum depth to collect data


behavior: badd_b                    #! visible = False
   b_arg: args_from_file(enum)      -1      #! ignore = True
                                    # >= 0 enables reading from mafiles/bhydro<N>.ma
   b_arg: start_when(enum)          1 # See doco above: 0, 1, 2
   b_arg: when_wpt_dist(m)          0 #! min = 5.0
                                # start_when == 7   ==> start when m_dist_to_wpt < this arg


   b_arg: stop_when(enum)           0 # Valid [0, 5] 0 stop when complete, 5 never stop
   b_arg: max_collection_time(sec)  1800 # timeout for data collect mode
   b_arg: max_search_time(sec)      1800 # timeout for search mode
   b_arg: min_download_range(m)     2000 # minimum range to start collecting data
   b_arg: max_tries_to_connect(nodim)  15 # max number of connection attempts
   b_arg: max_badd_errors(nodim)    30 # abort after this many errors
   b_arg: run_on_surface(bool)      0 # 1 -> allow running on surface
                                # 0 -> don't allow to run on surface
   b_arg: collect_data_after_range(bool)   1 # 1 -> collect data after range mode
                                # 0 -> don't collect data after range mode
```

```
# Collection parameters
b_arg: c_badd_mode(enum)            -1 # -1: Off, 0: search, 1: collect data
b_arg: c_badd_target_id(enum)        0 # address of remote host modem being called
b_arg: c_badd_range_secs(sec)       60 # how often to request range to remote modem
                                       # <0 => don't request range,
                                       # min value = c_badd_input_parse_secs(sec) * 2
b_arg: c_badd_input_parse_secs(sec)   30 # How long to check command response input buffer
b_arg: c_badd_datacol_status_secs(sec) 30 # How long to check command response input buffer
b_arg: c_badd_clear_remote_data(bool)  0 # 0: do NOT clear remote data after successful
                                       # download, 1: clear remote data after download
# Autobaud Parameters (if supported)
b_arg: c_badd_autobaud(bool)         0 # 0: No Autobaud, 1: Autobaud when in range
b_arg: c_baud_attempt_min(enum)      3 # 1: 80 bps MFSK
b_arg: c_baud_attempt_max(enum)      8 # 2: 140 bps MFSK
                                       # 3: 300 bps MFSK
                                       # 4: 600 bps MFSK
                                       # 5: 800 bps MFSK
                                       # 6: 1066 bps MFSK
                                       # 7: 1200 bps MFSK
                                       # 8: 2400 bps MFSK
                                       # 9: 2560 bps PSK
                                       # 10: 5120 bps PSK
                                       # 11: 7680 bps PSK
                                       # 12: 10240 bps PSK
                                       # 13: 15360 bps PSK
```

64: 80 bps FH

b_arg: c_autobaud_max_BER(nodim) 0 # Maximum Bit Error Rate to allow during autobaud

b_arg: c_badd_transaction_num(nodim) 0 # Transaction ID to execute for the glider. (8 digit max)

b_arg: c_badd_channel_probe_test(bool) 0 # Execute Channel Probe Test: 0-No 1-Yes

This behavior provides a mechanism to end a mission other than using the

typical surface behavior method. This was instigated by the need to end

a mission at depth.

behavior: mission_ender #! visible = False

b_arg: start_when(enum) 1 # See doco above: 1,2,3, or 4

behavior: comatose #! visible = False

b_arg: start_sci_hydrophone_collecting(bool) 1.0 # in, t-> start when this sensor true

b_arg: start_sci_viper_collecting(bool) 1.0 # in, t-> start when this sensor true

b_arg: start_sci_wants_quiet(bool) 1.0 # in, t-> start when this sensor true

b_arg: post_inflection_holdoff(s) 30.0 # in, how many secs post inflection to

hold off before going comatose

These do not get used to much. Generally only for testing

behavior: nop_cmds #! visible = False

b_arg: nop_pitch(bool) 0 # t-> cmd pitch to _IGNORE to keep stack busy

b_arg: nop_bpump(bool) 0 # t-> cmd bpump to _IGNORE to keep stack busy

b_arg: nop_heading(bool) 0 # t-> cmd heading to _IGNORE to keep stack busy

b_arg: nop_threng(bool) 0 # t-> cmd threng to _IGNORE to keep stack busy

```
   b_arg: secs_to_run(sec)   -1   # how long this behavior runs, <0 to run forever


behavior: oob_abort                    #! visible = False
   b_arg: start_when(enum) 6     # see doco above
   b_arg: when_secs(sec)   120.0 # How long to wait for issuing out of band abort


   # For testing iridium, sends file irdatst.dat
behavior: iridium_ascii_test                   #! visible = False
   b_arg: time_between_xmit(secs)   900.0  # 15 minutes
   b_arg: tries_per_xmit(nodim)       5    # How many attempts to send file
   b_arg: link_ok_timeout(secs)      30.0  # How long to wait for link ok
                                  # < 0 means do not expect "link ok"
   b_arg: modem_drain_time(secs)     30.0  # How long to delay phone power off


   # Turn the pinger on during a test mission
behavior: pinger_on                    #! visible = False
   b_arg: c_pinger_on(bool)          1
   b_arg: u_ping_n_enabled(bool)     1
   b_arg: u_pinger_rep_rate(sec)     8
   b_arg: u_pinger_max_depth(m)    1000
END
#endif
```