SLOCUM GLIDER TRAINING

RUTGERS
THE STATE UNIVERSITY
OF NEW JERSEY

14
MTS & NAVO

marine technology
SOCIETY
Opportunity runs deep™

TABLE OF CONTENTS

Glider Mission Preparation Overview

1. Post Deployment

Issues from last deployment:

- Consider issues that came up last deployment through note tracking or record keeping system. Timestamps of events typically help. Monitor items such as altimeter failures, bottom collisions, pitch angle and steering response, and battery curve/duration.
- Note/recall any settings you had to set, permanently (longterm.dat) or temporarily and properly associate with vehicle, software version, and scenario so that you may properly anticipate or correct for in next deployment.
- Your knowledge of last deployment will help prepare you and the glider for the next deployment.
- Take note of the number of inflections your vehicle has done.

Software change consideration

- Most of the time a software upgrade is a safe bet. However, always weigh the changes in the change log vs. how reliable a mission needs to be run. If the mission completed successfully and mission difficulty is high combined with no needed software changes, perhaps delaying an upgrade is prudent. During multiple short deployments, likely not recommended to open vehicle and upgrade unless a software change is necessary for mission completion.
- Before beginning your hardware preparations, now is a good time to upgrade software. You will be preparing the vehicle for ballasting/preparation and it will be open.

Proper procedure for updating software:

1. Obtain both flash cards and backup if not already backed up.
2. Format the card FAT and name appropriately for your inventory system
3. Transfer entire contents of 'to-glider' and 'to-science' for your vehicle type to flash card
4. Re-copy state\ and config\ folders back over to the cards for both glider & science, overwrite.
    - Science: Copy over old: proglets, tbdlist, nbdlist
    - Glider: copy over old autoexec, state folder, missions if you want them, config-surf, sbdlist, longterm.dat
5. Replace cards and proceed with vehicle open or closed depending on your status.
6. Turn on glider → goes into pico (app gives an error)
7. Do Science first:
    - Consci
    - Boot pico
    - Burnapp
    - App → test it, hit 'n' don't run independently, hit 'p'
    - Boot app
8. Do Glider second
    - Boot pico
    - Burnapp
    - App
    - Boot app

Post Inspection

- Measure all battery voltage levels and log. Replace batteries and prepare old batteries for shipment. If any cells are drastically different than another, contact TWR or investigate poor connector or conductor quality.
- Remove nose cone and altimeter and inspect pump for sediment and dirt. If pump inhibits access, leave nose pieces off until you seal and begin ballasting
- Briefly inspect for any metal fatigue, shavings, loose screws, and connectors
- Clean the inside and outside of the vehicle. Alcohol works well to clean lens covers, water and paper towels/non-lint cloths (depending if working near a seal or not) are good for the rest. Compressed air can clean quite efficiently.

Brief data inspection

- Be sure to communicate with your data or software POC regarding any issues with the previous deployment. You may need to glance through some troublesome segments yourself to see if the vehicle is deployable and in what condition.

2. Physical Preparation

Pre-ballasting

- Discuss with your scientist or POC regarding deployment location, length, and recovery location. This will allow you to obtain datasets for density and mission considerations.
- Get the glider mostly clean and seals greased (not necessarily replaced or cleaned) and prepared for multiple openings and closings (likely during ballasting).
- Print out necessary logs and ballasting targets.
- Consider changes to the vehicle, payload change, weight changes from batteries, often a repeat deployment is a small battery mass change which may only require one check dunk.

Ballasting

- There are tricks and shortcuts for ballasting but they are better left to be shown rather than written down.
- Make sure you account for temperature.
- Rutgers often performs static CTD comparison at this point between the Slocum and the BSP.
- Make sure to clear all air bubbles from the vehicle, and you can never type ballast too many times. Always be confident that m_ballast_pumped = 0 and m_battpos = 0.

Vehicle Hardening

- This step may or may not apply to your vehicle, but this is essentially nearing the final seal of the glider.
- Once you are comfortable with your ballast, or perhaps one last adjustment, you can begin preparing for the final seal. After the final seal you will begin to checkout the vehicles systems.
- Before moving on to checkout though, now is your time to make sure no connectors are strained, bottles are secured and present, zip tie any loose ends, make sure batteries secure, or begin to fix anything on the vehicle that you deem necessary for continued and current deployments.
- Often now is when continuity checks are performed for corrosion analysis, connectors are examined, tape and zip ties are replaced.

3. Checkout Procedures

The items in this section are outlined on the Rutgers checkout sheet. This will explain certain elements and give a flow to the process as opposed to explain the whole checkout. All these items came up from past deployments, if it's not on there, it hasn't happened to us yet you can wager.

Pre-Seal

- This will be the last time you seal the vehicle (hopefully). The main goal here will be to make sure all connectors are secure, software is upgraded, flash cards are clean and ready, batteries secured. The key areas to look at should be explained in person, pointing to common fail points.
- The battery check is important for older vehicles, and forces a consistency and slows down the vehicle sealing process. It has caught a couple faults and hopefully prevented a few more.
- Now is a good time to inventory the science bay, and iridium account number used for deployment.
- Rutgers O-ring procedure.
    - Remove all o-rings and clean them with water & lint free cloths as well as the grooves. Inspect for wear and shredding + rubber discharge onto the lint cloth, and pinches. An o-ring will last for several deployments if taken care of; if you're unsure or skeptical it's no reason to risk discard and replace with new cleaned (same fashion as above) o-rings.
    - Install the o-rings lubricated with fresh Parker O-Lube. A technicican could also install them dry onto a tough install. At this point one could grease using a folded lint-free cloth wrapped around the ring laden with grease. Then orbit the ring with the cloth applying a thin layer of grease to both sides while it sits in the groove, remove lint-free cloth.
    - Rub out the mating seal point on the hulls with a lint-free cloth.
- Each glider tends to have its own curves with the cables during sealing. Watch the trained curve of the cables and try to match it when mating connectors.
- Always understand the moving pitch battery cable will move with the battery so make sure the cable is free for the range of the battery.
- Bring the glider together carefully. Once you think it is together enough to engage the T Hex Key begin to thread carefully.
- NOTE: If at any time the key is difficult to turn and the glider is not sealed, immediately STOP sealing the glider and proceed to investigate the sealing joints. Often a cable or obstruction is preventing it from closing.
    - If inspection yields no findings, proceed to open and inspect the following:
        - Inspect mating surfaces and cables, check where the tie rod mates, look how the battery sits under the aft try and make sure there is clearance.
        - Make sure the o-rings are properly lubricated. Dry o-rings can make sealing the vehicle very difficult.
- Once the glider has come completely together and the T Hex Key begins to tighten, continue until you are comfortable with the seal. At that point back off the T Hex Key until it becomes quite loose again. You will re-tighten with the torque tool allowing the glider to come together ONLY under 15 in*lb of torque.
- Draw the vacuum high as opposed to low, as it can always be lowered later. This applies if you are unsure of your equipment or vacuum sensor on toolset.

Post-Seal

- Corrosion has been a tough demon to combat with the Slocum. We feel the best way to protect against the uncertainty of corrosion is to make all unknown aspects certain. What we mean by this is, to ground every piece of metal in the vehicle unless it's properly isolated. Webb has not done this purposely so that any grounding that does occur is more likely due to anodizing being eaten through on assembly screws. If all points of the glider either are isolated by their lonesome, or tied into the anode on the back we feel that is the best protection against vehicle damage to corrosion, or for that matter leaks. Hopefully this will become an issue of the past with proper isolation or grounding of parts not feasible to isolate. Also you can never have enough Zn protection. We have seen 10 g go fast, so I would invest in the 25 g or 40 g variants.
    - Digifins have been a calm exception to this fact and have not corroded, isolated steel & plastic
    - Slocum anodizing is suspect as well, this issue has been brought to attention
    - CTD is an important crux point in corrosion, recently causing a leak in a deep vehicle of ours
- While powered a 'use', 'wiggle on', 'wiggle off', 'use' check is always advised. Let this run for several minutes. In a high risk mission or one where modifications to the internals are performed, performing a wiggle in all glider pitch, roll and aspect angles is preferred.
- A pressure check would be ideal but may not feasibly be performed, as well as altimeter.
- Compass readings can vary greatly. Rutgers has not had trouble with vehicles deployed with compasses showing 30 degree drift from comparator. This is likely the upper limit Rutgers would use for safety.

Software Checks

A very successful approach to maintain a fleet of gliders includes strong software architecture for flight, plotting, and data management as well as inventory system. One way of making sure all vehicles fly the same way is to maintain an image of default software that is applied to each vehicle. This image is updated with new software releases, when new needs are found, and upon any general upgrade to how it works.

These software files include configuration files, mission files, and default mafiles. They are broken down by which vehicle type is being flown. 90% of the default files work as installed for most deployments, what usually needs to be customized is science specific items.

Typically, after a final seal and checks are performed. The technician would transfer the files in bulk via the freewave. The 'config\zmext.dat' file ensures that this bulk transfer of files is distributed to the right location on the vehicle. This process usually takes a minute or two.

I will state here the sensitivity of the SBD/TBD file. Perhaps one could simulate to find out SBD size but a safe bet would be to start small and increase in size until you are comfortable. Rutgers enjoys files between 100-150 KB / 3 hours for heavy data sets. Anything more increases surface time and phone bills. SBD's less than 100 KB for heavy data vehicles running 3-6 hour missions is an accomplishment but likely at a lowered data expense. Just remember though you can never go back and get the data if the vehicle is lost, so if the SBD does not have a bit of diagnostic or troubleshooting data you might not have a good record of what happened or how things have been changing over time. Slowly we are adding more and more sensors for troubleshooting purposes to the SBD but at highly delimited rates.

Key files included are:

- **mdblist.dat**
    - This file has recently become very important to us. This is the 2nd largest form of binary data on the glider and its importance surfaces for a variety of reasons.
        1. The file could be used for all data post-processing needs, however, we feel its better suited to keeping a moderate size for at sea troubleshooting. If you have the time and space, dbd should be used:
        2. However long deployments (> 4 months) will use all space available on a 2 GB system and the mbd file will be the only file you may be able to save.
        3. Thus it has become important for us to include the following in the mbd file:
            - GPS
            - Science data (non-SDL)
            - Engineering data
            - Vehicle state/decision data
        4. The above allows the file 2 functionalities, a backup in a loss of the dbd's for all science data to be present. The other is we can usually download this file via iridium for missions < 2-3 hours and be given a strong troubleshooting or problem solving dataset. If the vehicle aborts, often the mlg and mbd are downloaded for troubleshooting / abort severity decisions.

- **missions** & **mafiles**
    - Coastal example: default files provide 10 min missions to 15 meters depth
    - Deep example: 1 dive to 50 meters and back

- **config.srf** (for surface dialog consistency), **sbdlist.dat** (must be edited, certain sections uncommented to allow for science), and **highdens.dat** (it is ok for this file to include all the sensors that will go into the sbdlist.dat uncommented so no need for edit)

Other software checks include:

- Making sure logs\ folder is empty
- autoexec.mi last check
- setting of the waypoint parameters so that deployment the glider proceeds to the correct location (x_last_wpt_lat & x_last_wpt_lon). The vehicle will save them on power off.
- One new feature we have been implementing for cost, time, and effort savings is the pre-deployment conditioner for binary data transfer. Each binary file requires a header or cache file to decode into ascii, these are required. By default on the first segment of a mission, the glider will send the data along with the header. This often leads to duplicate transfers of 50 KB worth of data. To make a deployment much quicker, once a vehicle is ready (ie: mbdlist.dat, sbdlist.dat, and software version are up to par) the technician can initiate gliderDOS logging ('logging on'). Once the files are opened, they can be closed with 'logging off'. This creates the 4 sets of glider files. These should be transferred now using 'send *.*' and will come across header and all. Once decoded shore side, headers can be saved and accessed at any time. Now upon deployment, if one specifies for the vehicle to only send a cache file if it hasn't already, the initial files from the 10 min test mission (sbd and mbd) files will be very small and can be easily transferred for a quick analysis of glider performance. This is better shown than explained. It saves some cost and time on surface.

Final Considerations

- Is your vehicle ready to be put in the ocean for 30 days or more?
- Is it properly labeled with dates, phone numbers, affiliation, and motivations, reflective tape, etc.
- Rutgers has had success with a mission date end label along with phone number.
- Gather up all items needed for deployment, spare plugs, wings, freewave, antenna, and computer.

4. General Assembly Procedures

O-ring Etiquette

The glider's O-Rings are quite sturdy and can last awhile if they are cared for and the vehicle is kept clean. However, they are still cheap and anything suspect should be discarded.

The vehicle should be kept as immaculate as possible. Compressed air, paper towels (dry and moist), and low-lint wipes go a long way in keeping an open vehicle clean:

- inside and outside of hulls
- inner sealing lip of hulls
- batteries cleaned before insertion
- sand removed from sealing surfaces
- loose plastic and metal shavings cleaned and removed

Throughout the ballasting, preparatory, battery swapping process. A reminder in your head is you are planning to send a robot into the ocean for several months, minutes and good practice in the lab can go a long way in helping a glider succeed its mission. A good rule of thumb is to keep the rings clean and lubricated through the ballasting, re- battery process. Once ready for the final seal it may be a good time to replace the o- rings.

A 4 month deployment should be the lifetime of an O-ring, but as said above, the re- battery, ballasting process can use old o-rings that are well lubricated.

Rutgers uses Parker O Lube or petroleum barium grease as recommended from TWR. Some organizations and other sealed systems use silicon grease to lubricate o-rings. We find silicon to get on everything and unable to remove. However, there may be merit to silicon vs. barium grease in colder climates as the grease becomes more solid. Rutgers will continue to use barium grease until it finds a climate that can be proven its un- reliable.

NOTE: CARBON FIBER HULLS CAN SCRATCH EASILY. KEEP SEALING SURFACE OF HULL LUBRICATED WHEN INSERTTING AND REMOVING BATTERIES. TAKE CARE TO NOT BRING ANTHING HARD, PLASTIC OR METAL IN CONTACT WITH SEALING SURFACE. PLACES LABELS OF DATES OF NEW SCRATCHES, RETURN HULLS TO BE REFINISHED WHEN ABLE TO.

5. Glider Ballasting

Step 1: Weight Inventory

1. Label ballast sheet with glider, date, preparers, and mission. Make sure all parts of glider are well labeled from ballast bottles, hulls, to nose.
2. The main section of the ballast sheet pertains to vehicle weights. These weights are tracked for 2 reasons:
    a. Record keeping
        i. In the event a glider is shipped away for long term
        ii. Used in conjunction with another glider and parts may be swapped
        iii. Allows for swapping of parts without a scale or means of obtaining a weight
    b. Exact ballast (calculated volume)
        i. Knowing the exact volume of the vehicle aids in ballasting accuracy
        ii. When deploying new instruments, strap on (external riding) instruments, etc.; it is helpful and safe to find the total vehicle package volume
3. Clean and disassemble the vehicle to be worked on. Separate out main ballast bottles, batteries, and sections of vehicle.
4. Carefully record weights of the parts. Be sure to not leave parts out that are normally installed during flight (ie: power on/off plug, flash cards). Defaults are noted, but note any exceptions. Be careful not to double weigh parts as well (ie: nose batteries installed in fore section). Be careful to use only dry weights.

Step 2: Ballast Considerations / Target Density

1. Determine
    a. Day 0 (start of deployment)
        i. Surface Temperature
        ii. Surface Density
        iii. Bottom temperature
        iv. Bottom Density
    b. Day X (end of deployment) (x days later)
        i. Surface Temperature
        ii. Surface Density
        iii. Bottom Temperature
        iv. Bottom Density
2. The above data can be obtained from following sources
    a. Recent CTD casts, cruises, live datasets including gliders, observatories, ARGOS drifters
    b. Historical records from similar time periods of previous years from sources similar to above
    c. Ocean models
    d. If no good data is available, it is best to take a guess and ballast light. Proceed to deploy with a CTD sensors on hand, to make adjustments at the last minute. This is important for freshwater areas, or brand new uncertain zones.
3. Decide on a target density and temperature and enter these on the ballast sheets
4. If you have a previous volume measurement, enter that as starting point

How to pick the right density is beyond the scope of this article, but as a reminder for a trained individual, we have included the following information below:

- Priority in ballasting
    - Lowest surface density across deployment span
        i. Ballast no more than 2.5 sigma units higher than this (expert users disregard)
    - Coldest surface water across deployment span
        i. Glider will lose volume at surface, thus inhibiting surfacing. We must account for the coldest water in the ballasting decisions
    - Densest bottom water across deployment span
        i. Obtaining full dataset from top to bottom is important, but not as important as being able to surface. Coastal gliders may hit ranges they cannot operate to full depth in. Often bottom is given up in lieu of surfacing reliability.

Step 3: Ballasting

1. Be sure tank is thoroughly mixed and the water is less dense than your target density
2. Be sure glider is in ballast mode
3. Be sure glider is in ballast mode (it's that important)
4. Make judgment call on whether this dunk will be valid as a 'volume' dunk. This means is the glider close enough to obtain a highly accurate weight in tank for it to be used in the calculation of the glider's density. In other words:
    a. If it is hundreds of grams out of trim, fix first
    b. If it is hundreds to thousands too light or heavy, fix first (get it to more ballpark)
5. If glider is not the above, then proceed with highly accurate density dunk
6. Hang glider from scales, use weights as needed to submerge
7. Place scales equal distant from glider's center of gravity (about middle of science payload)
8. Prepare external CTD for measurements and place in tank while glider settles
9. Record
    a. Scale weight, fore and aft
        i. Log any external weight locations
    b. External CTD Temp
    c. External CTD Cond
    d. External CTD density
10. Remove from scales, record:
    a. Log external weight values (in water weight!)
    b. Glider CTD Temp
    c. Glider CTD Cond
11. Obtaining vehicle roll
    a. Get glider to just negatively buoyant via:
    b. Remove wings if way too heavy
    c. Use ballast pump to obtain +- ~ 200g
    d. While glider is hovering, barely submerged, obtain roll value of vehicle
12. Proceed with H-Moment calculation if new glider, new payload, or new configuration with external instrument. (not this document)
13. Remove vehicle from tank

Step 4: Ballasting Predictions / Corrections

1. Enter information from the dunk on the ballasting spreadsheet. The outputs from the dunk will be in the purple sections. Take note that 'Weight in Tank' = 'Scales' – 'External weight added'
    a. Calculated Glider Volume
        i. This is the volume of the vehicle, given the weights above in air are accurate as well as the tank density, temperature and weight of the vehicle in the tank.
    b. Glider Density 2, 3
        i. This is the density in target water, using the entered and calculated volume.
2. Dunk sanity check
    a. As a general rule of thumb, (glider volume in L) g ~ 1 sigma unit (50 g for G1). In other words if the glider pulls on the scales by 300 g in a tank of 1020 water, your glider is about 1026 density. Use that to make sure your entered values are within spec.
        i. G1 volume is 50-51 L
        ii. G2 volume is 55-60L
3. Finalize the dunk
    a. Enter the 'Calculated Glider Volume' as 'Entered Volume' if you trust the measurement and it makes sense
4. Print out the sheet and label Iteration 1
5. Make ballast adjustments by the pink section
6. Large changes should be logged in the weights section (payload bay, etc). Smaller changes (just bottles) can be logged on the iterations sheet as well as reflected in the weights section.
7. Be sure to add Trim corrections to the adjustment (out of scope of this document).

General Glider Communications

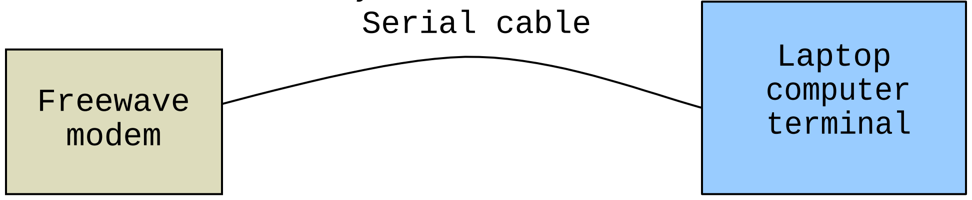
RUTGERS Coastal Ocean Observation Lab

Communications & Control

Glider communicates via terminal emulation software (ie: Symantec's Procomm Plus, Windows Hyperterminal, Webb Research's Dockserver), ZOC. A serial cable links the Freewave modem to the computer running terminal software.

Terminal Settings required for glider communications

- **BAUD:** 115200 bps
- **PARITY:** N
- **DATABITS:** 8
- **STOPBITS:** 1
- **Sending & Receiving Files:**
  - via **Z-Modem** protocol
  - *Crash Recovery* must be turned ***off*** for send & receive



General Glider Commands & Control

1. Applying plug will turn glider on into either PICODos or GliderDOS. GliderDOS should be default and is verified with 'boot app' command.
2. GliderDOS will load glider's custom calibrations/configurations from autoexec.mi file in \config. This is majority of spam on bootup. Keep in mind the air pump and buoyancy pump will default to surfacing mode.
   Glider is vehicle out in ocean, it will attempt to take action upon a restart, this action will be to run a station keeping mission (initial.mi 📬⌨ lastgasp.mi):

```
Automatically Sequencing initial.mi

Sequencing missions
load_mission(): Opening Mission file: initial.mi
for execution once  load_mission(): Opening
Mission file: lastgasp.mi for execution FOREVER
1 mission specifiers - sequencing 1 total missions (not
  counting lastgasp.mi):  initial.mi
  lastgasp.mi




SEQUENCE: About to run initial.mi on try 0
   You have 120 seconds to type a control-C to
   terminate the sequence.  The control-P
   character immediately starts the mission.
   All other characters are ignored.
```

3. *Ctrl^C* will obtain control of the glider. *Enter* will let you know if you have a command prompt, tap a few times to see what current glider status is if unsure; likely you will be at a prompt.

4. Glider will initiate iridium communications, *callback xx* (xx in minutes!) will halt this.
5. When operating in a lab environment and especially without a cowling, you must deflate the air bladder: *put c_air_pump 0*
6. If in lab it is good practice to now place the glider in lab mode, *lab_mode on*.

NOTE: from here on out your actions may change whether you are in water or on land

NOTE: The glider will try to run a mission (same as when booting up) after 10 minutes of inactivity in GliderDOS (*lab_mode on* will remove this feature!)! Be sure to hit *Enter* if glider is sitting idle!

Glider Directory Structure

Slocum Directory Structure (science similar, minus missions/mafiles)

c:\

app\

- main glider application resides here (glider.app)

bin\

- various glider routines reside here, ex:
    - zr, zs
    - burnapp (upgrade software)
    - burnhex (upgrade firmware)

config\

- KEY FOLDER – backup between deployments, backup & restore for software upgrade
- contains the following important files – must know how they work & manipulate
    - autoexec.mi
    - sbdlist.dat, mbdlist.dat, highdens.dat
    - config.srf
    - longterm.dat
    - simul.sim
    - zmext.dat

logs\

- 4 types of glider data are written here as a mission is undertook
    - 3 x binary data files (sbd, mbd, dbd)
    - 1 x ascii mission log (mlg)
- 'send' command adds functionality of any files sent via 'send' command get copied to \sentlogs\ folder
- sys.log file will show vehicles, start, restart, mission start/end, aborts and is usually transferable via iridium

sentlogs\

- all files in here are usually derived from the 'send' command

state\

- KEY FOLDER – backup between deployments, backup & restore for software upgrade
- longterm.sta file in this folder creates glider memory across reboots & power cycles
    - do not clear or loose this file unless directed by TWR
- \state\cache\ folder contains header/cache information for decoding binary data files

missions\

- houses your mission files, don't need to navigate to use

mafiles\

- houses your mission argument files, don't need to navigate to use

C:\config\autoexec.mi

Must know and understand:

- specific and required for each vehicle, gives:
  - Name
  - Calibrations
  - Settings
- this is one of 2 startup files:
  - Every sensor in there is set **once** and only **once** upon vehicle startup
  - **longterm.sta** will load sensors too, **after** autoexec.mi has loaded
- all other sensors will obtain your software version's masterdata defaults
- include this file (for that matter folder) in every glider software upgrade

If you follow the above rules and understandings, you should not run into any problems. It will be habit to properly maintain and use the autoexec.mi as a required file and a tool. Knowing at system startup that the sensors included in autoexec.mi will override software defaults (from masterdata) will enable you to carefully place, change, input, and maintain the autoexec.mi.

Always keep a date log at the top of the file and include any changes. It is OK but not a great habit to change this file via the freewave. It is **very** unsafe to change it via iridium and care and guidance should be taken if it is really necessary.

Save a state of your deployed and recovered glider flash card, this file may be crucial to diagnosing, troubleshooting, and backtracking your science.

Sample File:

```
# autoexec.mi for unit 034
# unit 034, ru05
# C-2060-2 Rev E, ser. no. 040 driver board
# tailfin glider
```

Comment [DKA1]: Inventory information

```
#   Date        e-maiL address           comment
#   3/29/05     pcollins@webbresearch.com        initial setting
#   3/31/05     mpalanza@webbresearch.com        replaced, re-cal fin motor
#   08-Apr-2005 creed@imcs.rutgers.edu       turned off pinger, changed phone number to Rutgers both commercial
and military,
#   5/4/05      pcollins@webbresearch.com        pressure ducer removed replaced and recaled,max working depth to
120m
#   5-4-05      mpalanza@webbresearch.com        replaced broken fin shaft, re-cal'd fin
#   5/30/06                                      commented out # Use CTD depth for flying # sensor:
u_use_ctd_depth_for_flying(bool) 1 used during deployment 062  Roarty, Haldeman
#   5/30/06     dkaragon@marine.rutgers.edu      added calibration coefficients for puck fl3 and bb3 and SAM and
lines for sam
#   6/22/2006   kerfoot@marine.rutgers.edu       added calibration coefficients for SAM 011
#   8/1/06      dkaragon@marine.rutgers.edu      remarked out SAM sensor after it was damaged and removed post
RIMPAC
#   30-Dec-06   cjones@webbresearch.com          added c_ctd41cp_num_fields_to_send(nodim) 4  to enable timestamp
#                                                updated persistor and card ser number f_fin_reqd_vel_frac .25 --
> .15 cold water
#   25-May-07   jdingess@webbresearch.com        recalibrated ballast pump motor
#   26-Jul-07   pcollins@webbresearch.com        replaced aft cap and fin.  Rebuilt fin motor, repaired pitch
motor, recalibrated fin and pitch motor
#   14-Aug-07   pcollins@webbresearch.com        installed BB3SLO from RU17 original, OCR 504I and calibration
coefficients.  Cal coefficients for 504I are defaults from master data.
#   4/2/09      dkaragon@marine.rutgers.edu      cleaned up autoexec, removed cal's for following instruments:
SAM 011, FL3SLO-482, BB3SLO-276
#   2008/6/25   dkaragon@marine.rutgers.edu      added new pitchmount, subtracted .325 from f_battpos_cal_b
#   21aug2008   bhess@marine.rutgers.edu         commented out all optics sensor cals, ru05 payload in ru01
```

Comment [DKA2]: This is record keeping and is very, very important. Log all changes to autoexec.mi here as this may be the last line of defense tracking of inventory, changes, troubleshooting, science calibrations, etc.

```
#   12DEC2008   dkaragon@marine                  bpump deadz width to 30 cc; added optics payload from ru06,
confirmed cals.
#   28-JULY-09  pcollins@webbresearch.com        Installed new science bay sn 180. Install and calibrate new
pressure transducer
#                                                rebuild and recalibrate ballast pump assy. Install TNT compass
and Aandera optode.
#   13-Aug-09   haskins@marine.rutgers.edu       Installed new fin 48 original fin is unlabeled. calibration
completed and added, cleaned up autoexec.
#   2009_08_19  dkaragon@marine.rutgers.edu      put fin speed (f_fin_reqd_vel_frac) back to .25 (a change from
Antarctica); added payload 0152, cleaned up autoexec; set f_ballast_pumped_deadz_width to 15 cc since shallow
deployment and we get a bit more speed
#                                                optode sends timestamp for DEP
#                                                m_tot_num_inflections = 33876, pump has been rebuilt, so this is
new pump from here on out, 0 inflections
#                                                cut puck sensors, optode has timestamp by default
```

```
name RU05
```

Comment [DKA3]: This defines the glider. This is what name it shows when calling, in data files, etc.

```
#Unit 034 RU05
#C-2060-2 Rev E #040
#ARGOS ID #04653
#Seimac Smartcat #61525
#Freewave slave #915-4409
#Freewave master #915-5916
#GPS #J04174499
#Compass #25390
#Persistor Main #52039
#Flash Card #SSD-C01GI-3005
#Persistor Payload #51461
#Flash Card Payload #17762
#Stack Card Payload #N/A
#U4art #N/A
#SBE #0036 rc, 350 psi
#Pressure Transducer #80090, 300 psi
#Bouyancy Pump #36
#Pump Motor 40
#Pitch Motor # 34
#Air pump assembly #AIR 0001
#Airmar Transducer/altimeter #395650
#Steering Motor #028
#Iridium Antenna #
#Pinger #PGR 0028
#Iridium Modem #300003000713240
#Iridium SIM Card #We owe you one! Liz put ru01 military card in
#BB3SLO_SERIAL_NUMBER: BB3SLO648
#OCR_504I SERIAL_NUMBER:
```

Comment [DKA4]: More inventory

```
installed gps
installed attitude_rev
installed ocean_pressure
installed vacuum
installed battery
installed argos
installed air_pump
installed pitch_motor
installed science_super
#installed pinger
installed fin_motor
installed altimeter
installed iridium
installed buoyancy_pump
```

Comment [DKA5]: This is where you tell the glider which hardware devices it has installed. You can manipulate this to add devices, take some out of service temporarily (ie: use glider without the pump or pitch battery). As you can see we don't use the pinger in this vehicle.

```
#################
# GENERAL GLIDER #
##########################################################################
# Max Vehicle Coastal Depth (100 m pump)
sensor: F_MAX_WORKING_DEPTH(m)    102.0

# current correction on
sensor: u_use_current_correction(nodim) 1

# Altimeter model, 0 for Benthos and 1 for Airmmar, -1 for experimental
sensor: f_altimeter_model(enum) 1  # airmar(mod1)
##########################################################################
```

Settings. Note: max vehicle depth rating here.

```
######################
# IRIDIUM INFORMATION #
##########################################################################
# IRIDIUM PHONE NUMBERS
# For a commercial card:    001508XXXXXXX (Example)
# sensor: c_iridium_phone_num(digits) 15085482446
  sensor: c_iridium_phone_num(digits) 17329322865  #IMCS phone bank

#     For a military card:    00697508XXXXXXX
#   sensor: c_iridium_phone_num(digits) 6975085482446 #  WRC phone number !no spaces!
#   sensor: c_iridium_phone_num(digits) 6977329322865 # IMCS phone bank

sensor: c_iridium_lead_zeros(nodim) 2 # number of leading zeros in phone number
                                      #   typically 2 for both commercial or military
##########################################################################
```

information. Note lead zeros are separate. Latest software versions must worry about alternate phone number, when it switches over, etc.

```
###########
# SCIENCE #
##########################################################################
# Turns science computer on
sensor: c_science_on(bool) 1

# NUMBER OF COLUMNS TO SEND FOR EACH INSTRUMENT
# CTD, Send timestamp
sensor: c_ctd41cp_num_fields_to_send(nodim)      4
# OPTODE, send timestamp
sensor: c_oxy3835_num_fields_to_send(nodim)      4

##############
# BB3SLO-601 #
#####################################################
# 2009_08_19    verified/initial by dkaragon@marine #
#               650 nm NOT 660 nm (b660)            #
#####################################################
sensor:  sci_bb3slo_is_installed(bool)       1   # in, t--> installed on science
sensor: c_bb3slo_num_fields_to_send(nodim)  3
sensor:  u_bb3slo_is_calibrated(bool)        1   # false, assume not calibrated

# sensor specific input calibration constants (defaults for BB3SLO-207)
sensor:  u_bb3slo_b470_do(nodim)         49      # dark offset, nodim == counts
sensor:  u_bb3slo_b470_sf(Mnodim)        11.22   # scale factor (0.000000117)
sensor:  u_bb3slo_b532_do(nodim)         48      # dark offset, nodim == counts
sensor:  u_bb3slo_b532_sf(Mnodim)        7.326   # scale factor (0.00000817)
sensor:  u_bb3slo_b660_do(nodim)         47      # dark offset, nodim == counts
sensor:  u_bb3slo_b660_sf(Mnodim)        3.442   # scale factor (0.00000385)
#####################################################

################
# BBFL2SLO-631 #
#####################################################
# 2009_08_19    verified/initial by dkaragon@marine #
#####################################################
sensor:  sci_bbfl2s_is_installed(bool)       1   # in, t--> installed on science
sensor: c_bbfl2s_num_fields_to_send(nodim)  3    # in, number of columns to send on each
sensor:  u_bbfl2s_is_calibrated(bool)        1   # false, assume not calibrated

# sensor specific input calibration constants (defaults for BBFL2SLO-234)
sensor:  u_bbfl2s_bb_cwo(nodim)          49      # clean water offset, nodim == counts
sensor:  u_bbfl2s_bb_sf(Mnodim)          2.54    # scale factor (0.00000247)
sensor:  u_bbfl2s_chlor_cwo(nodim)       43      # clean water offset, nodim == counts
sensor:  u_bbfl2s_chlor_sf(ug/l/nodim)   0.0121  # scale factor to get units
sensor:  u_bbfl2s_cdom_cwo(nodim)        46      # clean water offset, nodim == counts
sensor:  u_bbfl2s_cdom_sf(ppb/nodim)     0.0903  # scale factor to get units
#####################################################
```

Comment [DKA8]: Science Setup. Here are the calibrations for your instrument. They ideally would be placed on the science card but as is they are in the autoexec.mi so that the glider loads them each bootup. These let the glider manipulate the data correctly. Often these are taken from instrument calibration sheets.

Great care must be taken in scientific notation. See:
"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.

Also note the use of the columns to send. This is important for advanced users who want more accurate timestamps for sensors, perhaps higher sampling rates (by cutting sensors returning from science), or for other reasons.

```
######################
# DEVICE CALIBRATIONS #
##########################################################################
# vacuum
 sensor:    u_vacuum_cal_m(inHg/Volt) -11.443 # Factory Calibration data
 sensor:    u_vacuum_cal_b(inHg)       25.455 #      inHg = m V + b

# ocean_pressure
 sensor: f_ocean_pressure_full_scale(bar) 24.000 # pressure @ FS volts
 sensor: f_ocean_pressure_min(volts) 0.216 # voltage for 0 pressure
 sensor: f_ocean_pressure_max(volts) 1.974 # voltage for FS pressure
#  sensor: u_pressure_autocal_max_allowed(bar) 0.6

# ballast_pumped
 # max = safety_max - deadzone
 sensor: f_ballast_pumped_safety_max(cc) 243.0  # in, damage to glider
 sensor: f_ballast_pumped_deadz_width(cc)  30.0  # 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) 43  # in, nominal speed
 # 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  # max = safety_max - deadzone
 sensor: f_ballast_pumped_cal_m(cc/Volt) 489.788 # in, slope
 sensor:  f_ballast_pumped_cal_b(cc)        -424.810 # in, y-intercept

# battpos
 # max = safety_max - deadzone
 # x_max_battpos = f_safety_max_battpos - f_deadzone_width_battpos
 sensor: f_battpos_safety_max(inches) 1.2  # in, damage to glider
 sensor: f_battpos_deadzone_width(inches) 0.2 # 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)  0.56 # nominal speed
 # 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)  2.517  # slope
# Old Pitch-Mount Value:
#  sensor: f_battpos_cal_b(inches)        -3.154 # y-intercept
# New Pitch-Mount Value: (-.325)
 sensor:  f_battpos_cal_b(inches)        -3.479 # y-intercept

# fin, motor.c motor_drivers . fin calibrated at 35 degrees
 # max = safety_max - deadzone
 sensor: f_fin_safety_max(rad) 0.610  # in, damage to glider
 sensor: f_fin_deadzone_width(rad) 0.035 # in, Sets x_ limit
 sensor:  f_fin_db_frac_dz(nodim)       1.0   # deadband as fraction of dead zone
 sensor: f_fin_nominal_vel(rad/sec) 0.0981 # in, nominal speed
 # 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)  0.556# slope
 sensor: f_fin_cal_b(rad)        -0.737# y-intercept
 sensor: f_fin_reqd_vel_frac(nodim) 0.25    # in, fraction of nominal
                                            # required before saying not
                                            # moving fast enuf
##########################################################################
```

Comment [DKA9]: These are the calibration informat specific to your potentiometers (motors) for your fin, pump, and pitch motors. Care and knowledge must be taken if swapping parts, motors, etc as to what needs to be changed, etc.

Most often reasons for this changing are changeparts (pitch mount) and new fin (old-style) being installed.

Missions & MAfiles

```
####################
# 100 meter Slocum #
####################

sensor: c science all on enabled(bool)          0   # control science individually
sensor: u_dbd_sensor_list_xmit_control(enum)    2   # 2  = transmit header if THIS glider hasn't sent it before
```

```
# ABORT BEHAVIOR
behavior: abend
    # OVERDEPTH: glider finds itself in > 204m of water
    b_arg: overdepth(m)                          104
    b_arg: overdepth_sample_time(sec)            12

    # OVERTIME: disabled
    b_arg: overtime(sec)                         -1.0

    # COP TICKLE: watchdog not tickled for 12.5% before hardware jumper
    b_arg: no_cop_tickle_percent(%)              12.5

    # SAMEDEPTH: 15 min every 60 seconds
    b_arg: samedepth_for(sec)                    900.0
    b_arg: samedepth_for_sample_time(sec)        60.0

    # STALLED FOR: 15 min every 60 seconds
    b_arg: stalled_for(sec)                      900.0
    b_arg: stalled_for_sample_time(sec)          60.0

    # HARDWARE: vacuum, battery
    b_arg: undervolts(volts)                     9.5
    b_arg: undervolts_sample_time(sec)           36.0
    b_arg: vacuum_max(inHg)                      12.0
    b_arg: vacuum_sample_time(sec)               36.0

    # WAYPOINT TOO FAR ABORT: 500km (500,000m)
    b_arg: max_wpt_distance(m)                   500000
```

```
# SURFACE : No waypoints commanded
behavior: surface
    b_arg: args_from_file(enum)      10  # mafiles/surfac10.ma
    b_arg: start_when(enum)          3  # 3-heading idle
```

```
# SURFACE : Nothing commanded
behavior: surface
    b_arg: args_from_file(enum)      10  # mafiles/surfac10.ma
    b_arg: start_when(enum)          1  # 1-stack idle
# SURFACE : Yo completes (this is caused if an altimeter hit causes a dive & climb in same cycle)
behavior: surface
    b_arg: args_from_file(enum)      10  # mafiles/surfac10.ma
    b_arg: start_when(enum)          2  # 2-pitch idle
# SURFACE : Surfacing interval
behavior: surface
    b_arg: args_from_file(enum)      20 # mafiles/surfac20.ma
    b_arg: start_when(enum)          9 # 9-every when_secs
# SURFACE : Hit a waypoint
behavior: surface
    b_arg: args_from_file(enum)      30  # mafiles/surfac30.ma
    b_arg: start_when(enum)          8  # 8-when hit waypoint
# SURFACE : No comms 0
behavior: surface
    b_arg: args_from_file(enum)      40  # mafiles/surfac40.ma
    b_arg: start_when(enum)          12  # 12 No comms
```

```
# GOTO_LIST : Waypoint list
behavior: goto_list
    b_arg: args_from_file(enum)      10  # mafiles/goto_l10.ma
    b_arg: start_when(enum)           0  # 0-immediately
```

```
# YO : Flight params
behavior: yo
    b_arg: args_from_file(enum)      10  # mafiles/yo10.ma
    b_arg: start_when(enum)           2  # 2-depth idle
    b_arg: end_action(enum)           2  # 2 resume
```

```
# SAMPLE : CTD
behavior: sample
    b_arg: args_from_file(enum)     01  # c profile on
# SAMPLE : BB3
behavior: sample
    b_arg: args_from_file(enum)     12  # c bb3slo on
# SAMPLE : FL3
```

```
behavior: sample
    b_arg: args_from_file(enum)     11  # c fl3slo on
# SAMPLE : BBFL2S
behavior: sample
    b_arg: args_from_file(enum)     1   # c bbfl2s on
# SAMPLE : BBAM
behavior: sample
    b_arg: args_from_file(enum)     4   # c bbam on
# SAMPLE : OCR
behavior: sample
    b_arg: args_from_file(enum)     1   # c ocr504i on
# SAMPLE : OPTODE
behavior: sample
    b arg: args from file(enum)     1   # c oxy3835 on
```

```
# PREPARE TO DIVE : wait 12 min for GPS
behavior: prepare_to_dive
    b_arg: start_when(enum)         0   # 0-immediately
    b_arg: wait_time(sec)           720 # wait time


# SENSORS_IN : not sure what this does, assuming remaining sensor update
behavior: sensors_in
```

Key MAfiles

```
behavior_name=yo


<start:b_arg>
    b_arg:  start_when(enum)                2   # pitch idle
    b_arg:  num_half_cycles_to_do(nodim)    -1  # infinite dive/climbs
# DIVE_TO ARGUMENTS
    ###################################
    # Depth Arguments
    b_arg:  d_target_depth(m)       15
    b_arg:  d_target_altitude(m)    5
    ###################################

    ##########################################################
    # Advanced Flight Controls (do not change)
        # Ballast Pump Controls
            b_arg:  d_use_bpump(enum)    2          # 2  Buoyancy Pump absolute
            b_arg:  d_bpump_value(X)     -1000.0
        # Dive Angle Arguments (using AP)
            b_arg:  d_use_pitch(enum)   3          # servo, rad, < 0 dive
            b_arg:  d_pitch_value(X)    -0.454
        # Dive Angle Arguments (using battpos)
            # b_arg:  d_use_pitch(enum)      1     # 1  BattPos in
            # b_arg:  d_pitch_value(X)       0.362
    ##########################################################

# STUCK SCENARIO
    b_arg:  d_stop_when_stalled_for(sec)    240
    b_arg:  d_stop_when_hover_for(sec)      180

# CLIMB_TO ARGUMENTS
    ###################################
    # Depth Arguments
    b_arg:  c_target_depth(m)       3
    b_arg:  c_target_altitude(m)    -1
    ###################################

    ##########################################################
    # Advanced Flight Controls (do not change)
        # Ballast Pump Controls
            b_arg:  c_use_bpump(enum)    2          # 2  Buoyancy Pump absolute
            b_arg:  c_bpump_value(X)     1000.0
        # Dive Angle Arguments (using AP)
            b_arg:  c_use_pitch(enum)   3          # servo, rad, < 0 dive
            b_arg:  c_pitch_value(X)    0.454
        # Dive Angle Arguments (using battpos)
            # b_arg:  c_use_pitch(enum)      1     # 1  BattPos in
            # b_arg:  c_pitch_value(X)       0.362
    ##########################################################

# STUCK SCENARIO
    b_arg:  c_stop_when_stalled_for(sec)    240
    b_arg:  c_stop_when_hover_for(sec)      180

    b_arg:  end_action(enum)                2
<end:b_arg>
```

```
behavior_name=goto_list

# ENDURANCE LINE (OFFSHORE WAYPOINT FIRST)
# 1 km RADIUS


<start:b_arg>
    b_arg:  num_legs_to_run(nodim)  -1  # -1 loop, -2 run once, > 0 = this many waypoints
    b_arg:  start_when(enum)         0   # 0 baw_immediately
    b_arg:  list_stop_when(enum)     7   # 7 baw_when_wpt_dist

    # SATISFYING RADIUS
        b_arg:  list_when_wpt_dist(m)     3000

    # LIST PARAMETERS
        b_arg:  initial_wpt(enum)        -1  # 0 to n-1, -1 first after last, -2 closest
        b_arg:  num_waypoints(nodim)     2   # num of waypoints in list
<end:b_arg>

<start:waypoints>
    -7414.000   3926.250
    -7252.500   3900.000
<end:waypoints>
```

```
behavior_name=surface

# SURFAC20.MA (Timeout Expired)

<start:b_arg>
    # Surface for timeout expired
        b_arg:  when_secs(sec)          600

    # Flight Controls
        b_arg:  c_use_bpump(enum)       2
        b_arg:  c_bpump_value(X)        1000
        b_arg:  c_use_pitch(enum)       3       # servo, rad, >0 = climb
        b_arg:  c_pitch_value(X)        0.4528  # 26 degrees

    # Surface Timeouts & Other Params
        b_arg: report_all(bool)             0       # F->just gps
        b_arg: end_action(enum)             1       # Surface menu
        b_arg: gps_wait_time(sec)           600     # GPS wait
        b_arg: keystroke_wait_time(sec)     300     # Surface time
        b_arg: printout_cycle_time(sec)     40.0    # Surface menu print rate
<end:b_arg>
```

```
behavior_name=sample

# SAMPLE01.MA (CTD)

<start:b_arg>
    b arg: sensor type(enum)                    1   # 1 c profile on
    b_arg: sample_time_after_state_change(s)    0   # start sampling right away

    # Sampling Arguments
        b_arg: intersample_time(sec)    2   # if < 0 then off, if = 0 then fast as posible, > 0 secs
        b_arg: state_to_sample(enum)    15   # 7 diving|hovering|climbing
<end:b_arg>
```

C:\config\sbdlist.dat

C:\config\tbdlist.dat (science computer)

C:\config\highdens.dat

Must know and understand:

- this file is loaded at startup, but can be changed at a surfacing or during gliderDOS
- this file defines what sensors are written to the SBD/TBD file for iridium deployment transfer
- there is a feel to this file for size (and subsequently surfacing time) so be consistent and you can be predictive to size
- unlike the mbdlist.dat, this file accepts arguments to help decrease file size.
- example file is pre 7.x SBDLIST.DAT file, science WILL NOT be included on a new 7.x SBDLIST.DAT file as no data is sent back to the glider computer
    - vice-versa, the TBD file will have only sci_xxx specific sensors

This file will be your data you receive from the vehicle.

Good Habit:

1. Rutgers typically sends upcast and downcast, every other or third yo, back at 4 sec resolution for each science sensor (TBD)
2. Occasional glider pressure (m_pressure) record can be used to monitor depth sensor performance
3. GPS data is key
4. m_water_depth as well as m_altitude may help monitor bottom tracking performance
5. Occasional roll, pitch, and heading data may help monitor vehicle performance

Sample File:

```
# INTERVAL                Seconds since last stored value, 0 is store every value
# STATE                   State of glider (dive, hover, climb), 15 is always store
# HALFYOS                 # of dive/climbs in this segment to record for, -1 store for all
# YO DUTY CYCLE           Store data every n'th yo in this segment, -1 is every dive/climb

# ------------------------------------------------------------------
# SENSOR NAME             INTERVAL  STATE     HALFYOS   YO_DUTY_CYCLE
# (Defaults)              0         1         -1        -1
# ------------------------------------------------------------------
```

```
# 2010_01_21 dkaragon, added m_gps_full_status, removed @ surface requirement for GPS and
water velocities
```

```
# Basic Data
  m_present_time
  m_water_depth
  m_gps_lon
  m_gps_lat
  m_gps_full_status
  m_final_water_vx
  m_final_water_vy
  m_tot_num_inflections
```

```
# Flight Parameters
  c_alt_time                 120   15   2
  u_min_water_depth
  f_fin_offset
  u_hd_fin_ap_gain
  u_hd_fin_ap_igain
  u_pitch_ap_gain
  u_pitch_ap_deadband
  u_pitch_max_delta_battpos
  u_hd_fin_ap_inflection_holdoff
  u_hd_fin_ap_hardover_holdoff
```

```
# Engineering Data
  m_pitch               0   3
  m_roll                60  5   2

  c_heading             360 7   -1  3
  m_heading             60  7   -1  3
  m_fin                 60  7   -1  3

  m_battpos             120 5   2

  m_pressure            60  7
```

Comment [DKA1]: Note the arguements passed to sensors.

FIRST – sampling rate, 0 as fast as you can, > 0 is record every n seconds

SECOND – state of the vehicle, look in masterdata 'state_to_sample'

THIRD – number of halfyos to sample for. 2 means one dive and one climb, it will sample n halfyos and then stop

FOURTH – sampling duty cycle. 2 means every 2nd **yo** is recorded.

TBDLIST.DAT File

```
# INTERVAL              Seconds since last stored value, 0 is store every value
# STATE                 State of glider (dive, hover, climb), 15 is always store
# HALFYOS               # of dive/climbs in this segment to record for, -1 store for all
# YO_DUTY_CYCLE         Store data every n'th yo in this segment, -1 is every dive/climb

# ---------------------------------------------------------------------
# SENSOR NAME           INTERVAL  STATE     HALFYOS   YO_DUTY_CYCLE
# (Defaults)            0         15        -1        -1
# ---------------------------------------------------------------------

# 2010_02_22 dkaragon; SDL initial default, tbdlist.dat

# Basic Data
  sci_m_present_time


# CTD41CP Sea-bird  CTD(SBE-41)
# proglet = ctd41cp
# ----------------------------
  sci_ctd41cp_timestamp            2   7   -1  2
  sci_water_cond                   2   7   -1  2
  sci_water_pressure               2   7   -1  2
  sci_water_temp                   2   7   -1  2


# Wet Labs BBFL2SLO custom 3-param fluorescence/scattering meter
# proglet = bbfl2s
# ---------------------------------------------------------------
  sci bbfl2s bb scaled        2   7   -1  2
  sci_bbfl2s_chlor_scaled     2   7   -1  2
  sci_bbfl2s_cdom_scaled      2   7   -1  2


# Wet Labs BB3SLO custom 3-param backscatter meter
# proglet = bb3slo
# -------------------------------------------------
  sci bb3slo b470 scaled      2   7   -1  2
  sci_bb3slo_b532_scaled      2   7   -1  2
  sci_bb3slo_b660_scaled      2   7   -1  2


# Wet Labs FL3SLO custom 3-param fluorescence meter
# proglet = fl3slo
# ---------------------------------------------------
# sci_fl3slo_chlor_units      2   7   -1  2
# sci_fl3slo_phyco_units      2   7   -1  2
# sci_fl3slo_cdom_units       2   7   -1  2


# Satlantic OCR504I Irradiance Sensor
# proglet = ocr504I
# ----------------------------------
# sci ocr504i irrad1       0   7   -1  2
# sci_ocr504i_irrad2       0   7   -1  2
# sci_ocr504i_irrad3       0   7   -1  2
# sci_ocr504i_irrad4       0   7   -1  2


# Anderra Oxygen Optode 3835
```

Comment [DKA2]: This for example writes (not samples!!!) CTD data every 2 seconds to the TBD file. This only occurs during state 7 (dive | climb | hover). It runs infinitely but only every 2 yo's (basically 50%). We will obtain 1 downcast, following upcast, and then skip the next entire yo.
Does the data exist? Of course as defined in your mission, its likely logging every yo. This data will be in your EBD file.

```
# proglet = ?
# -------------------------
  sci_oxy3835_oxygen          2   7   -1  2
  sci_oxy3835_saturation      2   7   -1  2
  sci_oxy3835_temp            2   7   -1  2
# sci_oxy3835_timestamp       2   7   -1  2


# Wetlabs BAM beam attenuation meter
# proglet = bbam
# ----------------------------------
# sci bbam beam c       2   7   -1  2
# sci_bbam_corr_sig     2   7   -1  2
# sci_bbam_raw_sig      2   7   -1  2
# sci_bbam_raw_ref      2   7   -1  2
# sci_bbam_therm        2   7   -1  2
# sci_bbam_timestamp    2   7   -1  2


# Wet Labs FLNTU Flourometer & Turbidity
# proglet = flntu
# --------------------------------------
# sci_flntu_chlor_units 0   7   -1  2


# Wetlabs ECO-Puck
# proglet flbbcd: Wet Labs flbbcd fluorometer
# -------------------------
# sci_flbbcd_chlor_units
# sci_flbbcd_bb_units
# sci_flbbcd_cdom_units
# sci_flbbcd_chlor_sig
# sci_flbbcd_bb_sig
# sci_flbbcd_cdom_sig
# sci_flbbcd_chlor_ref
# sci_flbbcd_bb_ref
# sci_flbbcd_cdom_ref
# sci_flbbcd_therm
# sci_flbbcd_timestamp
```

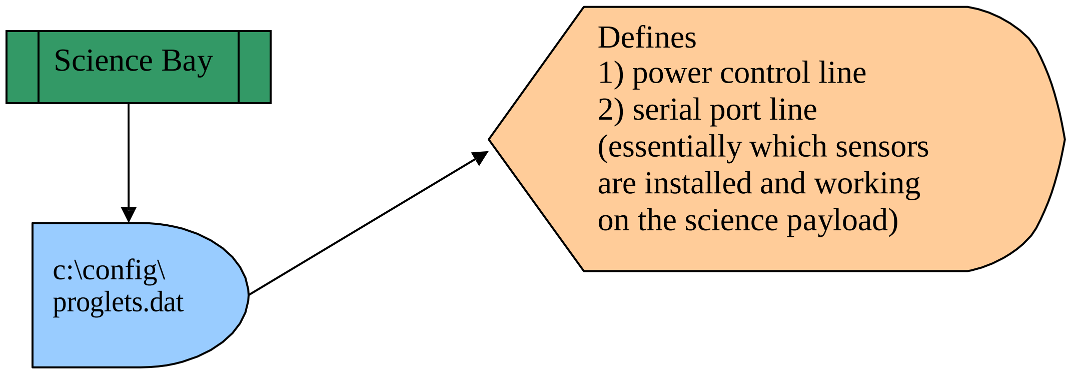
Science Bay Operation and Checks

Science Bay Operation and Checks

<u>CORE TASKS</u>

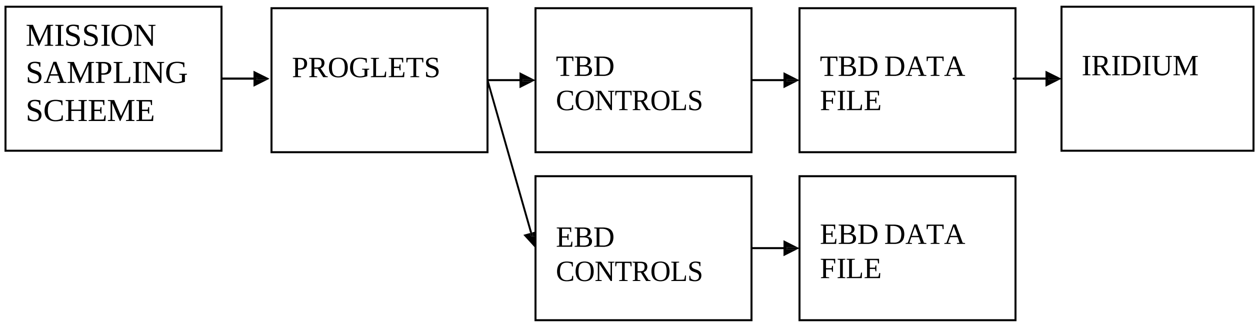
1. Understand Payload Bay Operation
2. FI lack of data, pilot and lab

<u>Payload Bay Operation</u>
The science computer now operates in a similar fashion to the glider as far as data logging, but its operation is a bit more file controlled and autonomous.



The proglets.dat file is the most important file on the payload bay. This file defines how the vehicle is able to turn on and talk to installed sensors. It also determines how to talk to the particular type of instrument installed on that serial line.

The hierarchy as to what data is generated can be confusing, the following diagram read from left to right. This is not a priority but it's a stack and each subsequent item can control the creation and storing of data, so the problem can be anywhere in the 'line':



The explanation of the above is:

- MISSION determines which sensors to turn on
    - PROGLETS.DAT describes how to talk to each science device and what is actually installed.
    - Checks are done if a called device is not responding, but if a device is not called proglets does not inform that a device is not being called.
        - TBD controls what gets written to the TBD file and thus what comes back via iridium. Often this is the source of the sampling

problem. To aid troubleshooting be consistent so you don't often troubleshoot the entire tree above when data is mission.
- EBD this may contain data the TBD does not depending on how you define your TBD.

<u>TBD CHECK</u>

TBD does not contain data from your vehicle. The first check is to verify your current configuration of TBD data logging. The file that controls this resides on the science computer.

1. Interface with the science bay:
    a. Issue command *Ctrl^t* if currently in a mission, or *consci* if drifting at surface or in gliderDOS.
2. Issue command *type ..\config\tbdlist.dat*
    a. Note the arguments for the sensors you are trying to monitor.
    b. Make sure you are using the right timestamp for the sensor, default, *sci_m_present_time*
    c. If the arguments show you that data should have been logged than assume your TBD is setup properly. Type *quit* to return to the glider. Also verify the glider is actually flying, ie: going up and down, yo'ing. An MBD or test missions with a more diagnostic *sbdlist.dat* with pitch and pressure record, etc. Proceed to next section if still a problem.

<u>Manual Science Test</u>

1. If you think the data file is properly configured is now recommended for manual troubleshooting. At this point it is likely safe to take the glider out of the mission because you will be inputting commands manually at the surface (or in the lab) to troubleshoot the vehicle.
    d. if in a mission, usually the surface dialog will offer the option to end the mission via a *ctrl^c*
2. Often it is desirable to operate your glider in an individual sensor controlling manner. If this is the case we will disable that functionality to test the entire science bay at once and will be the first step of our process to ensure the same starting point
    1. Obtain starting state of your glider so you can return it to 'the way it was' before you begin flying again, record the values for the following sensors somewhere:
       get c_science_all_on_enabled
       get c_science_send_all
       get c_science_on
    2. Issue command *put c_science_all_on_enabled 1*
    3. We need to tell the SDL enabled payload to return data values to the glider so we can diagnose:
       put c_science_send_all 1
    4. The next step will be to command the glider to turn the entire science bay on at a certain sampling rate. < 0 is off, 0 is as fast as possible, > 0 is that many seconds per sample. For our purposes 12 seconds will be good enough:
       put c_science_all_on 12
    5. Give the system several moments and then proceed to command the glider to return all exchanged data, between science and glider (actually this command increase verbosity on the link, nominally this value is 1 which

opens the science communications port):
put c_science_on 3

6. With a good eye, you should see values returning from science from all your installed sensors, the values should be changing, and perhaps makes sense for where the glider is located (temperature, pressure, etc). Some examples of data lines are:
science wrote:sci_bb3slo_b470_scaled(nodim) 1.198E-05
science wrote:sci_water_temp(degC) 22.7939
science wrote:sci_bb3slo_b470_scaled(nodim) 8.386E-05
science wrote:sci_water_temp(degC) 22.7840
7. If data is coming through on screen then your problem lies in your SBD or EBD file or the glider is not flying correctly. If you have no data output proceed to next section – **No Data Return from Manual Science Test**
8. Return the glider to Step 1's state and turn off the science bay's instruments:
put c_science_on __
put c_science_send_all __
put c_science_all_on_enabled __
put c_science_all_on -1

<u>How Science Can Be Controlled</u>

c_science_all_on_enabled 1

- Turn all science on/off, control all science as one unit
    - c_science_all_on (Sampling Rate)

c_science_all_on_enabled 0

- Don't control science as one unit
    - Use c_flbbcd_on, c_oxy4_on, c_profile_on, etc

Defaults:

- c_science_all_on_enabled 1 (true)
- c_science_send_all 0 (false)
    - should science computer send data to glider computer?
    - can overload glider if true
    - if true, lets me see science data via freewave terminal (e.g. in lab)
- c_science_on 1
    - turn on science computer

Glider Deployment

RUTGERS Coastal Ocean Observation Lab

Glider Deployment (short format)

- Make sure you have Glider Deployment Checklist
- Glider equipment
- Spare wings!

THIS GUIDE FOLLOWS THE GLIDER DEPLOYMENT CHECKLIST AND SHOULD BE USED AS A 2ND HAND REFERENCE WHEN DEPLOYING

1. **Obtain control of the glider** – do as so in class and the general communications sheet. The *enter* button pressed repeatedly will let you know if you are at a prompt.
2. **Allow glider to call Rutgers** – Once you have the following dialog, it is OK to type *callback xx* to obtain better control of the glider.

```
18631    Iridium modem matched: CONNECT 4800
18631    Iridium connected...
18631    Iridium console active and ready...
Vehicle Name: ru16
```

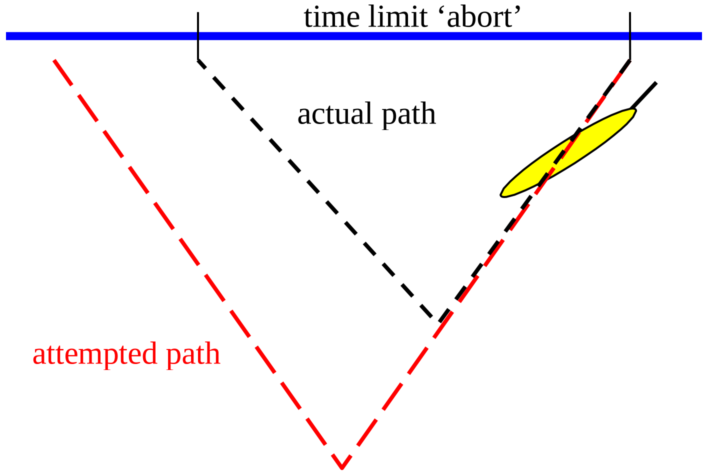
3. ***boot app*** – this is a crucial double check, entering the command should report (if the vehicle resets, it was NOT in *boot app* mode, obtain control after reset and continue):
   ```
   Boots Application at 0xE40000
   ```
4. **confirm boot app** – type *boot*
5. ***consci*** – This should switch the terminal control over to the science computer, your prompt will change to sci_dos. If this does not occur, call Rutgers or supervisor for further instruction.
6. **on boat – *run status.mi***:
   a. **What is this mission doing?**
      i. This mission is checking general mission parsing, input sensors, and GPS position.
   b. **What is end result of mission?**
      i. Glider should attempt GPS hits:

```
185.76 14 behavior surface_2: SUBSTATE 2 ->3 : waiting for GPS fix
185.84    init_gps_input()
186.15    sensor: m_gps_lat = 1754.2646 lat
186.21    sensor: m_gps_lon = -6701.6409 lon
186.31    sensor: m_gps_status = 0 enum
```

      ii. Mission should complete with following information:

```
201.29 16 behavior surface_2: STATE Active -> Mission Complete
201.39    behavior ?_-1: layered_control(): Mission completed normally
201.46    behavior ?_-1: run_mission(): Mission completed:
MS_COMPLETED_NORMALLY(-1)
```

7. **Place glider in water**
8. **zero_ocean_pressure**
    a. glider should report that the pressure sensor has been re-calibrated. This step is very important, and could be a solution to problems down the line with pressure sensors being out of calibration. This must be done with glider in the water.
9. **run overtime.mi: (SEE DEPLOYMENT CHECKLIST IF NECESSARY TO RUN!) (Rutgers no longer runs this mission during deployments)**
    a. **What is mission doing?**
        i. This mission tests an abort capability of glider detecting time, and responding to a time limit.
        ii. Tests buoyancy of vehicle, because it will dive



    b. **What is end result of mission?**
        i. Glider will dive but a time limit will expire and glider will 'abort' the overtime mission.
        ii. Glider will submerge for several minutes, witness it surface by monitoring Freewave or computer terminal.
        iii. Mission will end with an abort, if you have glider on terminal, hit enter to see if you are at a command line. You should either witness the following:

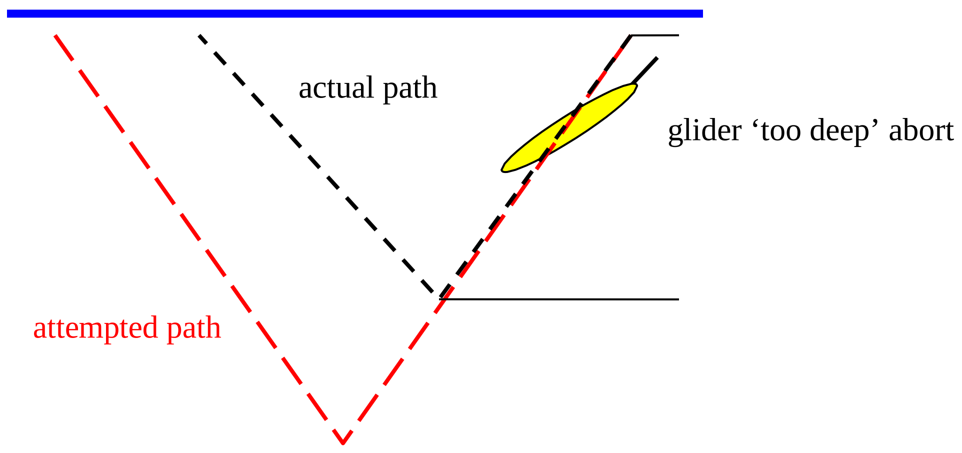
```
233.32    ERROR behavior ?_-1: we_are_done(): At the surface, return (-
2)MS_COMPLETED_ABNORMALLY

233.40    behavior ?_-1: we_are_done(): Restoring U_CYCLE_TIME from
15.000000 to 4.000000
233.50    restore_sensors()....
          Restored u_depth_rate_filter_factor from -1 to 4
233.59    behavior ?_-1:    ABOVE WORKING DEPTH
233.64    behavior ?_-1:    drop_the_weight = 0
234.87    behavior ?_-1: run_mission(): Mission completed:
MS_COMPLETED_ABNORMALLY(-2)
```

        iv. ***why?*** – That should indicate the reason for abort, in this case, ms_abort_overtime in case you missed the above messages.

```
ABORT HISTORY: total since reset: 1
ABORT HISTORY: last abort cause: MS_ABORT_OVERDEPTH
ABORT HISTORY: last abort time: 1987-09-16T12:27:14
ABORT HISTORY: last abort segment: ru17_ghost_deep-
1987-258-0-0 (0150.0000)
ABORT HISTORY: last abort mission: ODCTD7.MI
```

10. **run od.mi:**
    a. **What is this mission testing?**
        i. This mission tests the ability of the glider to detect depth and abort for being in water deeper than it thinks it should be in. The operator's task is to witness the glider submerge and surface. This verifies proper ballast of the vehicle. Occasionally for certain deployments a float will be used on the tail until ballast is confirmed.
    b. **What is end result of this mission?**
        i. Glider should dive and surface, this time aborting just as in overtime.mi but for overdepth.



        ii. Attempt to witness the following at mission completion:

```
172.03    ERROR behavior ?_-1: we_are_done(): At the surface, return (-
2)MS_COMPLETED_ABNORMALLY

172.09    behavior ?_-1: we_are_done(): Restoring U_CYCLE_TIME from
15.000000 to 4.000000
172.16    restore_sensors()....
          Restored u_depth_rate_filter_factor from -1 to 4
172.23    behavior ?_-1:    ABOVE WORKING DEPTH
172.27    behavior ?_-1:    drop_the_weight = 0
173.46    behavior ?_-1: run_mission(): Mission completed:
MS_COMPLETED_ABNORMALLY(-2)
```

        iii. If you do not see above, type ***why?*** and this should indicate reason for aborting, overdepth. Note abort count is now at 1-2 aborts.

11. **Receiving data files from test missions:**
    a. If you are on a terminal equipped with Z-modem protocol you can transfer the files from the test missions to the laptop.
    b. **send *.sbd *.mlg** ***.dbd *.tbd**

12. **Run the following missions:**
    a. **sequence 100_tn.mi(5)**
    b. **Ctrl-P** – will hasten the process of running the mission.

ONCE THE GLIDER DIVES FROM THIS MISSION, RUTGERS WILL OBTAIN CONTROL FROM THE NEXT SURFACING. DO THE FOLLOWING ITEMS:

1. ONCE GLIDER DIVES, UNPLUG FREEWAVE MODEM POWER
2. NOTIFY/CALL RUTGERS ALERTING THEM YOU PLACED THE GLIDER ON A 15 MINUTE MISSION
3. WEATHER CONDITIONS PENDING, TAKE A CTD CAST
4. WEATHER CONDITIONS PENDING, SLOWLY START STEAMING HOME
5. CONTACT RUTGERS IN 20-30 MINUTES FOR A STATUS, RUTGERS WILL CONTACT YOU EARLIER IF A SITUATION ARISES

Slocum Deployment (coastal focus long format)

1. Was software image applied to glider during checkout?
   a. SBD, MBD, TBD set accordingly
   b. Core mission MA files should be set correctly for test mission
   c. Confirm log folder is clear (of at least SBD's)
2. status.mi
   a. Mission completed normally and confirm GPS hit achieved
3. Vehicle Sanity Check
   a. Battery level
   b. Vacuum level (> 7 in Hg)
   c. Confirm 'boot app' with 'boot' command
   d. How is the clock, did you sync_time the day before? If not get a 3 minute GPS hit (callback 3) and issue a sync_time
4. Stage 1 deployment (glider in water)
   a. With or without float
      i. Typically float is used when glider is shipped and deployed in a new area or uncertain ballast
      ii. Floats are not used when confident of ballast
   b. zero_ocean_pressure
   c. run odctd.mi
      i. confirm abort is for overdepth
      ii. confirm boat witnessed submergence and reemergence
      iii. note abort time and mark (this will become deployment start time)
      iv. note GPS location and insert into GE or SFMC. Create spatial awareness of glider's location and first waypoint
   d. Boat side (if possible) (not essential)
      i. Transfer DBD and MLG's unless directory is full
5. Stage 2 deployment (Test Mission)
   a. Test Mission Parameters
      i. Runs for < 20 min
      ii. Lightly samples all data on science bay, only CTD is included in the SBD & MBD
      iii. Mission completes to gliderDOS
      iv. Backup timer of 30 minutes
   b. Confirm goto_l10.ma makes sense given GPS mission above
   c. Run Test Mission
   d. Transfer SBD, MBD, and TBD via iridium from the test mission. If unsure of the data file check c:\logs\sys.log
   e. Data analysis (depending on tool used, glider Plot or matlab scripts)
      i. Flight Dynamics
         1. Note average roll of vehicle, across up's and downs
         2. Note dive and climb pitch angles

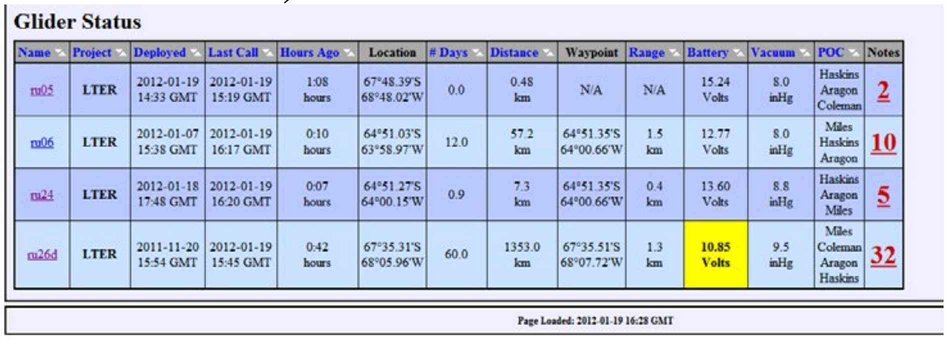
                a. Should at least be positive on climbs and negative on dives
                b. Not to exceed 35 degrees, if so take note, usually a glider should step up to right pitch angle, not overshoot
                c. Pitch not responsive?
            3. Note if vehicle tracks a heading to within +- 40 degrees
                a. Note if heading is tracked consistently port or starboard to intended
                b. If heading tracks about 0 error, fin should also cross over 0 point, confirm this
            4. Altimeter
                a. Confirm that we are seeing bottom (if possible, bottom < 80 m away)
                b. Any false hits or bottoms?
                c. Strong return on bottom > 2,3 m_water_depth's updated on the dive
        ii. Ballast considerations
            1. Dive and climb time should be equal given equal magnitude pitch on dive and climbs.
            2. Plot of water density can help decipher results
        iii. Pressure / Depth Checks
            1. CTD and glider pressure should agree (TBD data + MBD data)
            2. Confirm glider not impacting bottom
            3. Confirm if glider is breaching or near surface, note approximate climb depth.
            4. Note if glider appears 'out of the water' or negative depth
        iv. Science Checks
            1. Temperature, salinity, density sanity check
            2. Other data exists?
                a. Optode phases
                b. All necessary optical channels
            3. Timestamp check – Make sure both pressure sensors line up and no lag or missing data. If missing data possible science computer overload
    f. Note surfacing GPS and mark in GE, waypoint location still OK?
6. Stage 3 Deployment (Final Mission)
    a. Make necessary MA adjustments
        i. goto still pertinent?
        ii. adjust no_comms to 1 hour missions with backup set to 1 hour past eventual surfacing interval (ie: 4 hours for 3 hours)
        iii. yo10.ma
            1. adjust dive_to depth if glider was seeing bottom satisfactorily
            2. adjust climb_to depth if glider was breaching
        iv. if glider is not obtaining proper pitch angles quickly, make adjustment of doubling u_pitch_max_delta_battpos (usually .02 to .04)
        v. Correct any SBD, MBD, TBD file errors
    b. Run final mission

Daily Monitoring During Deployment

Slocum Daily Monitoring

STAGE 1: Situation and Location Awareness

1. Slocum Glider Mission Control Page (Situation Awareness)
   - **a.** Check for any updated notes and read them
   - **b.** Note last call in time and hours ago
   - **c.** Glance at battery and vacuum, any sensors there out of norm (sensors may be colored if out of norm)

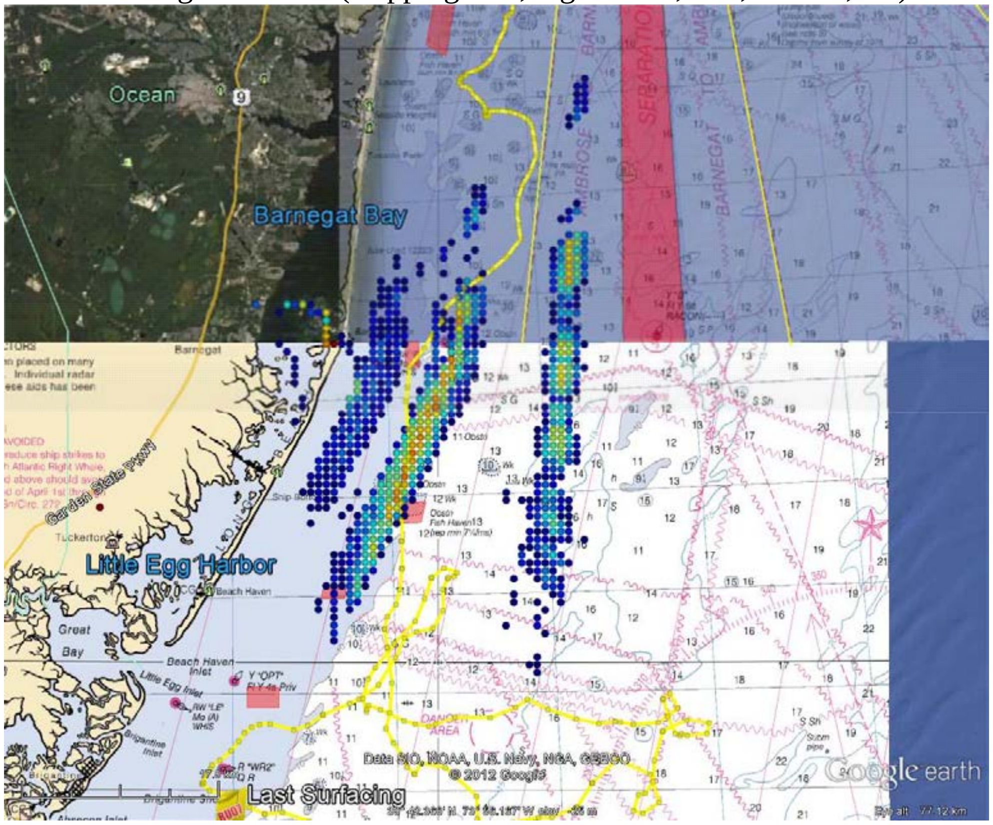


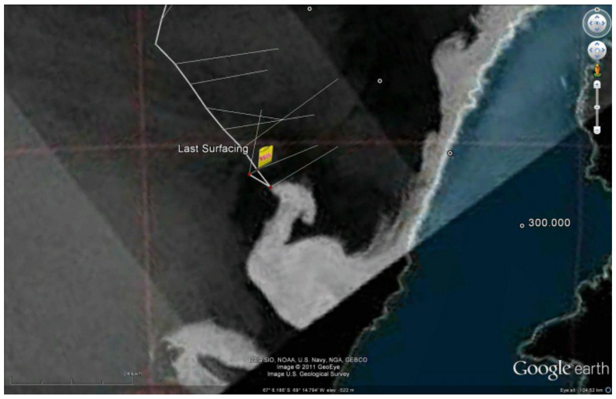
Glider Status

Name	Project	Deployed	Last Call	Hours Ago	Location	# Days	Distance	Waypoint	Range	Battery	Vacuum	POC	Notes
ru05	LTER	2012-01-19 14:33 GMT	2012-01-19 15:19 GMT	1:08 hours	67°48.39'S 68°48.02'W	0.0	0.48 km	N/A	N/A	15.24 Volts	8.0 inHg	Haskins Aragon Coleman	2
ru06	LTER	2012-01-07 15:38 GMT	2012-01-19 16:17 GMT	0:10 hours	64°51.03'S 63°58.97'W	12.0	57.2 km	64°51.35'S 64°00.66'W	1.5 km	12.77 Volts	8.0 inHg	Miles Haskins Aragon	10
ru24	LTER	2012-01-18 17:48 GMT	2012-01-19 16:20 GMT	0:07 hours	64°51.27'S 64°00.15'W	0.9	7.3 km	64°51.35'S 64°00.66'W	0.4 km	13.60 Volts	8.8 inHg	Haskins Aragon Miles	5
ru26d	LTER	2011-11-20 15:54 GMT	2012-01-19 15:45 GMT	0:42 hours	67°35.31'S 68°05.96'W	60.0	1353.0 km	67°35.51'S 68°07.72'W	1.3 km	10.85 Volts	9.5 inHg	Miles Coleman Aragon Haskins	32

Page Loaded: 2012-01-19 16:28 GMT

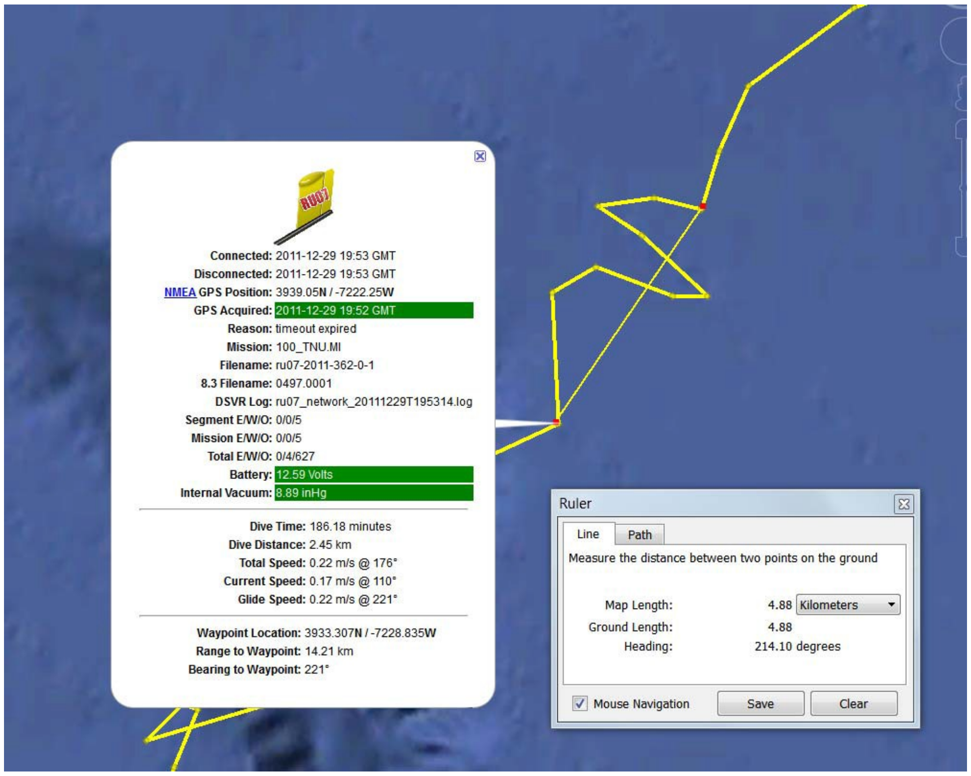
2. Check Google Earth (Location awareness)
   - **a.** Proper Overlays
   - **b.** Glider in Dangerous Area (shipping lane, high traffic, reef, wrecks, etc)





ICE! SAR, MODIS imagery

c. Speed Check, distance made good towards waypoint
ex: below, glider not making good distance towards waypoint, in this case .2 km / hr!



Connected: 2011-12-29 19:53 GMT
Disconnected: 2011-12-29 19:53 GMT
NMEA GPS Position: 3939.05N / -7222.25W
GPS Acquired: 2011-12-29 19:52 GMT
Reason: timeout expired
Mission: 100_TNU.MI
Filename: ru07-2011-362-0-1
8.3 Filename: 0497.0001
DSVR Log: ru07_network_20111229T195314.log
Segment E/W/O: 0/0/5
Mission E/W/O: 0/0/5
Total E/W/O: 0/4/627
Battery: 12.59 Volts
Internal Vacuum: 8.89 inHg

Dive Time: 186.18 minutes
Dive Distance: 2.45 km
Total Speed: 0.22 m/s @ 176°
Current Speed: 0.17 m/s @ 110°
Glide Speed: 0.22 m/s @ 221°

Waypoint Location: 3933.307N / -7228.835W
Range to Waypoint: 14.21 km
Bearing to Waypoint: 221°

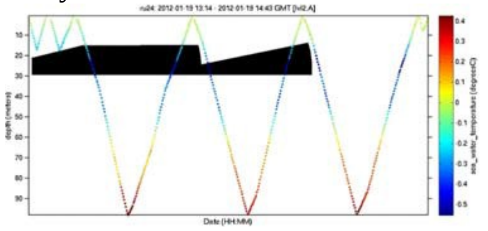
Ruler
Measure the distance between two points on the ground
Map Length: 4.88 Kilometers
Ground Length: 4.88
Heading: 214.10 degrees

Stage 2: Monitoring

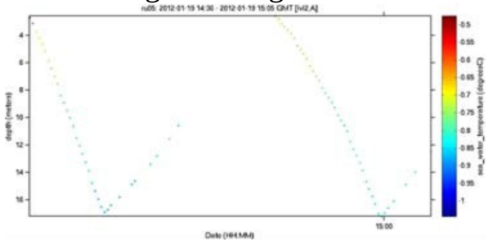
3. Slocum Glider Mission Control Page – data analysis

a. Data issues

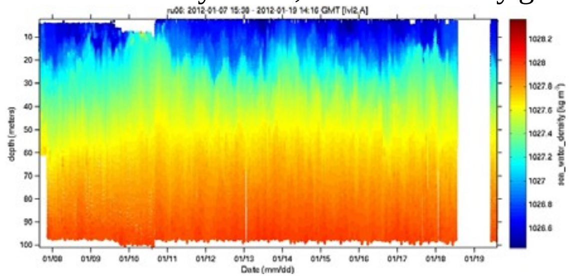
i. Early altimeter returns



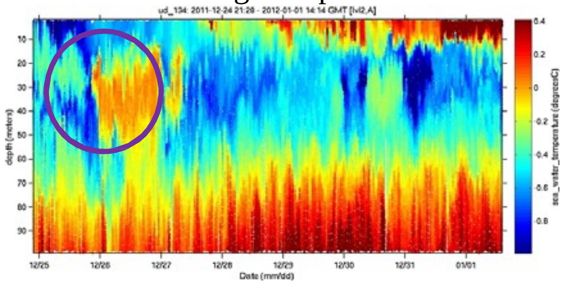
ii. Vehicle Lag / missing data



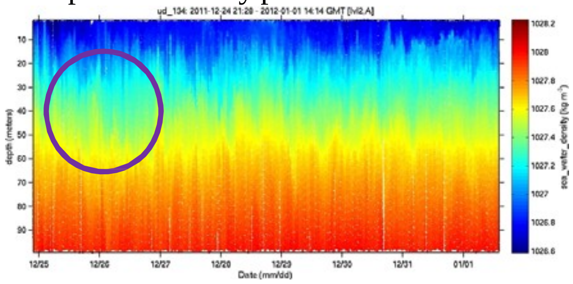
b. CTD data sanity check, uniform density gradient



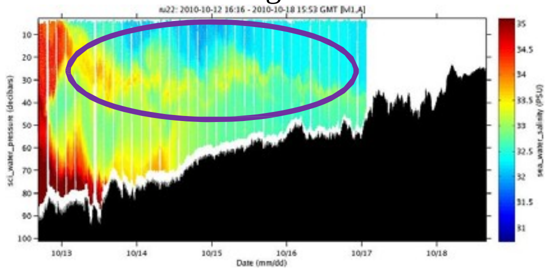
c. Take note of strange temperature anomalies



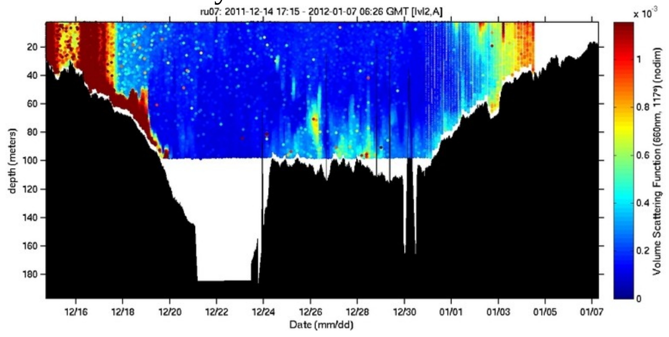
Compare to density plot



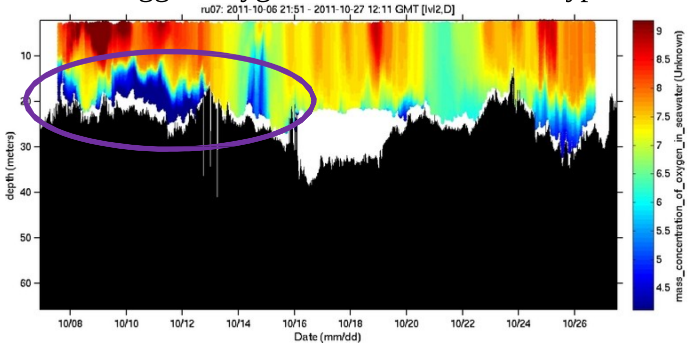
d. Check for thermal lag



e. Typical Backscatter (returns near bottom and in coastal / mixed environments) and not too noisy



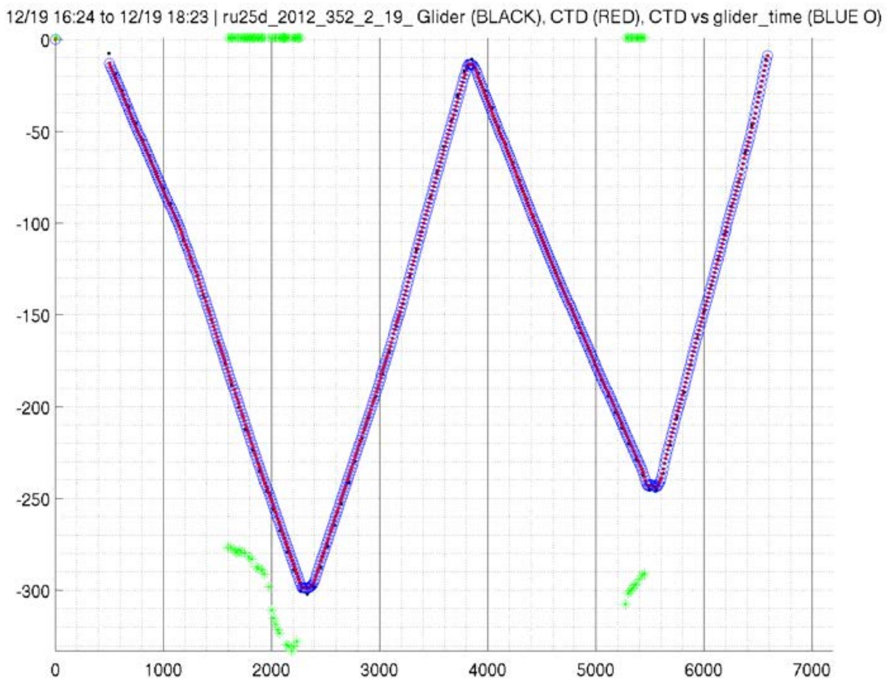
f. No time lagged oxygen data and check for hypoxia zones



4. Diagnostic Plots

a. Depth plot

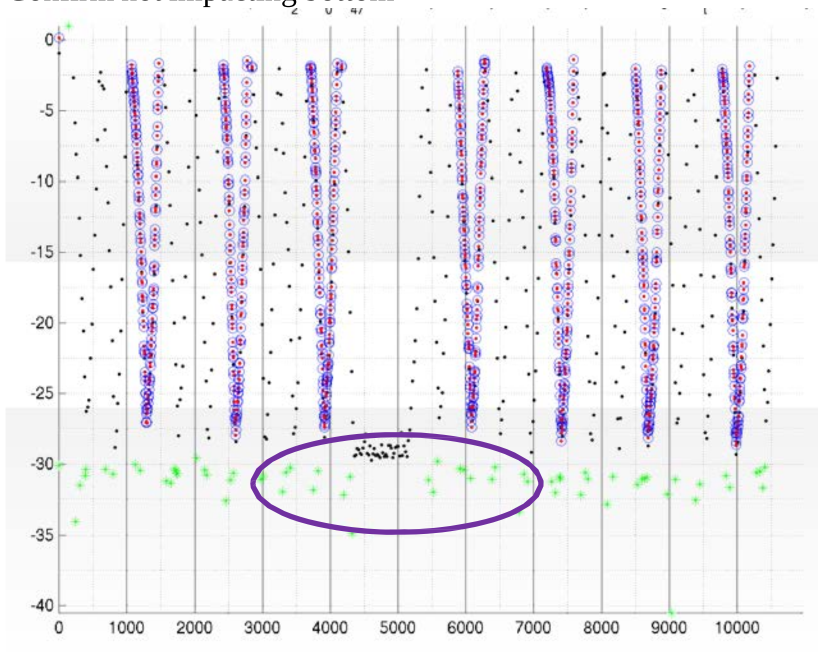
This plot should clearly show the gliders path through the water, vertically across time. Any anomalies in glider path take mental note of. Plot should also clearly show when the glider inflects and how/what it sees as the bottom. This plot can also compare the glider and science pressure sensors to make sure they agree.



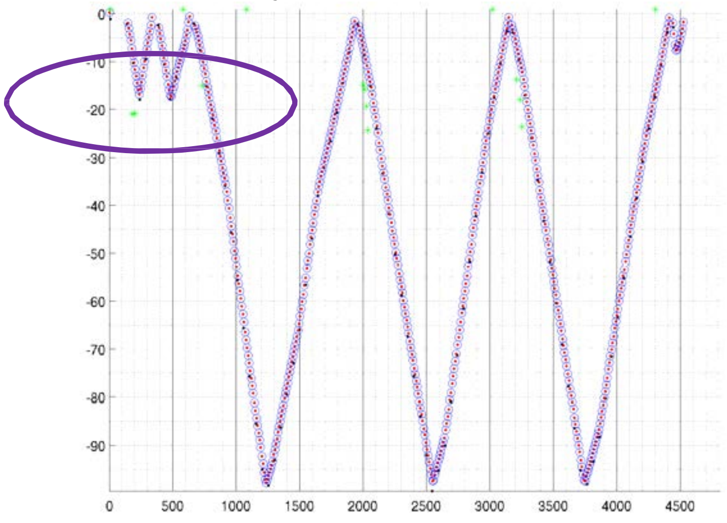
i. Confirm good ballast

ii. Confirm pressure sensors agree

iii. Confirm not impacting bottom



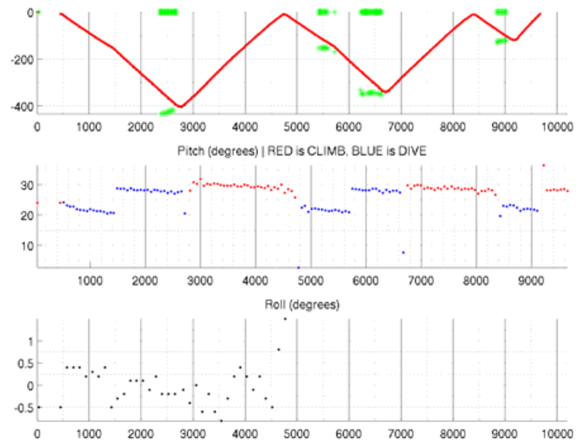
iv. Confirm no early altimeter hits



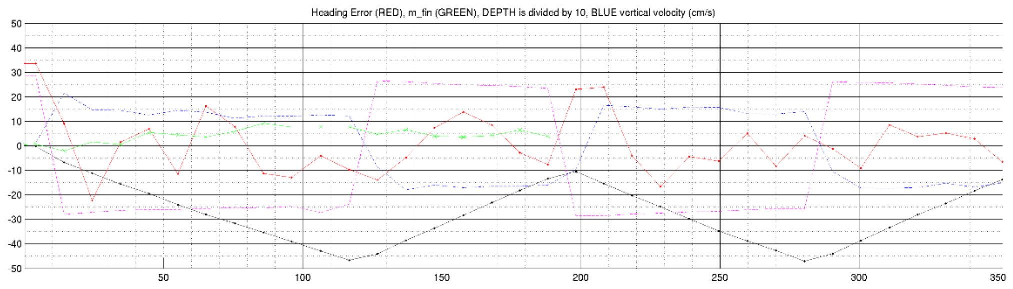
b. Engineering Plot

This plot should clearly show the glider's pressure record for reference. The plot should show the entire dive and climb pitch angles. The plot should show the roll values, preferably different colors for up/down, however the depth plot takes reference of such. Confirm pitch angles are desired. They are not toggling, taking too long to obtain pitch, etc.

i. Confirm roll is steady and same throughout mission. This is better suited to a longer time series (mission, not segment)



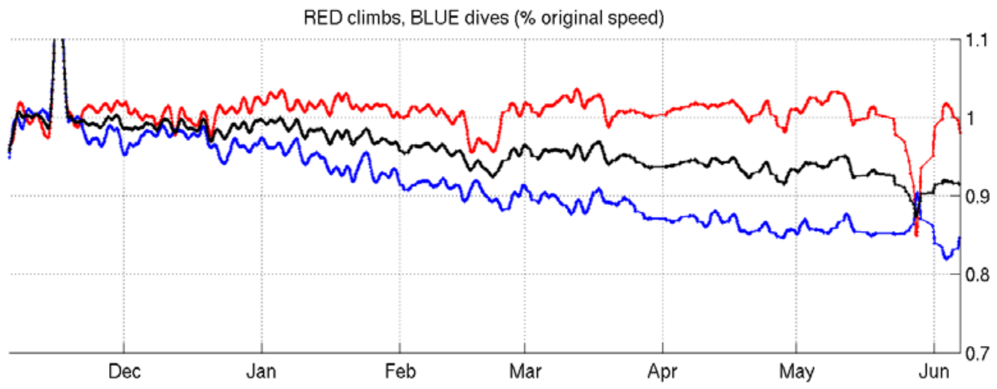
ii. Check heading performance (correlate with GE speed and progress estimate) These heading plots are for long term monitoring of glider steering and speed. This would be a low priority item but having a diagnostic in place could prevent long term problems from sneaking up.



c. **Vertical Velocity Plot**

An up and coming plot important for all missions but very important for > 30 day missions is vertical velocity monitoring. A quick plot

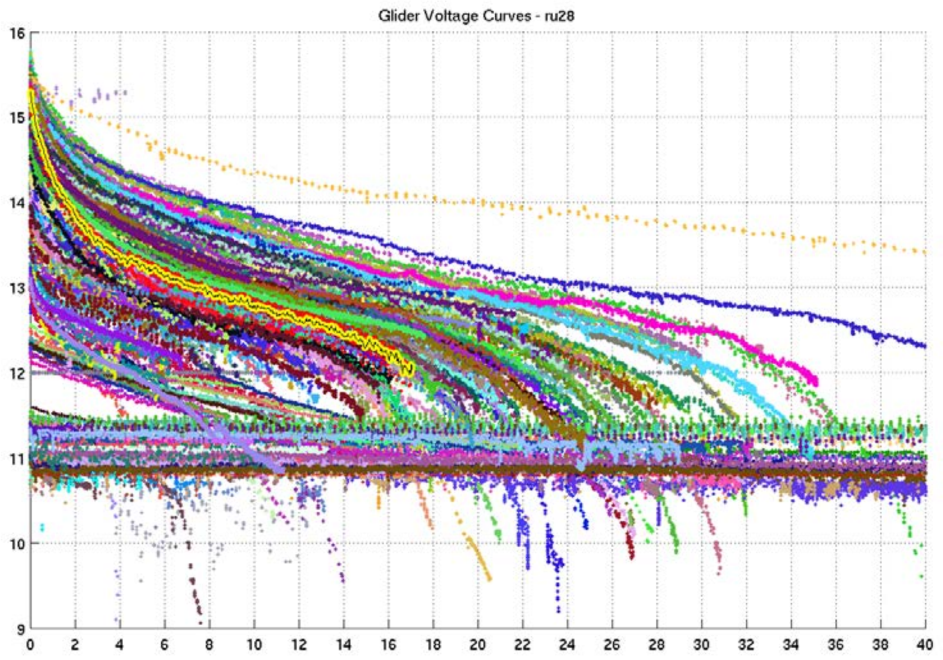
can show trends in the vertical velocity for piloting purposes. A more averaged plot of the combined up/down vertical velocity corrected for ballast and pitch can show biofouling speed decreases.



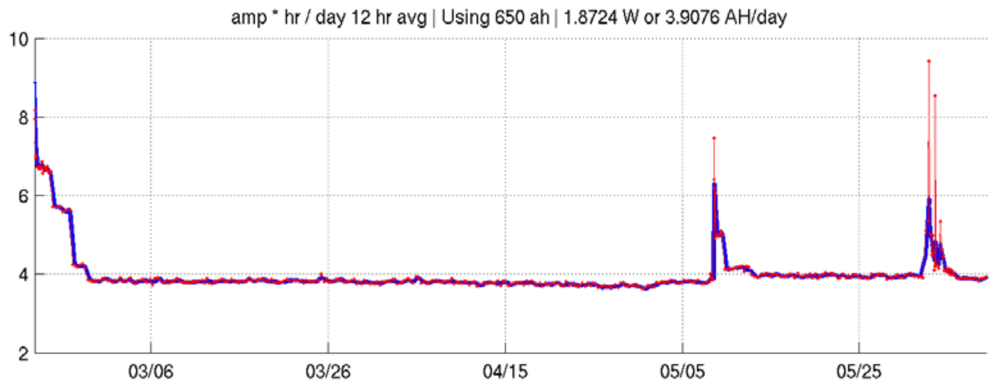
d. **Battery Diagnostic (primarily Aklaline)**

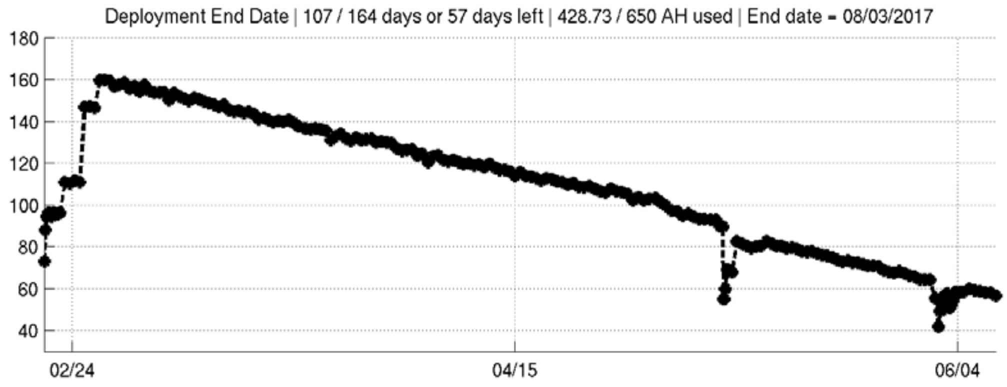
This plot should have some sort of error range which could be an envelope plot of all previous glider deployments. The current trajectory should be plotted or overlayed to show a projection to battery depth. As coulomb counters become more prevalent this plots useful decreases but monitoring could should a trend or problem with the glider, science instrument, or other.

- **i.** Check for strange drops
- **ii.** Predict days remaining (distance we can travel = days remaining * speed made good)



e. **Power / Energy Diagnostic (primary Lithium but relevant to Alkaline)**
This plot should show the current energy drain and make predictions as to the length of mission remaining. This is useful to combine with the glider speed/day for mission planning purposes.





Glider Recovery

RUTGERS UNIVERSITY
COASTAL OCEAN OBSERVATION LAB
OPERATIONS CENTER

Glider Recovery

IMPORTANT NOTE:
GPS and Iridium antennas are shared. You must issue a **callback xx** command to insure a timely GPS once glider communications are established.

1. Glider will call Dockserver and issue its GPS location. These should be used prior to leaving dock. Communication from Rutgers personnel or an email/text message to Sat phone from a Dockserver email can facilitate this.
2. To drift a glider 2 items must be taken into consideration:
    a. *u_max_time_in_gliderdos* is normally set to 10 min in seconds. You must change this to something longer to be safe the glider will not run a mission and disappear from surface. For instance ***put u_max_time_in_gliderdos 3600***
    b. for longer drifts, *u_iridium_max_time_til_callback* can be increased to be able to issue the command ***callback 60*** (longer than 30 minutes). This will drift glider only calling in every 60 minutes. Normally this is not needed for a standard recovery
3. Shore-side personnel can be called for latest GPS locations as well if need be.
4. GPS positions are obtained with a ***callback xx*** command.
5. type ***where***, glider will respond with the following:

```
GliderLAB I -3 >where Vehicle
Name: ru01
Curr Time: Tue Jan    8 20:48:17 2008 MT:      13931
DR   Location:    3928.824 N -7412.074 E measured        13930.6 secs ago
GPS TooFar:     69697000.000 N 69697000.000 E measured   1e+308 secs ago
GPS Invalid :     3929.233 N -7418.050 E measured          1.667 secs ago
GPS Location: 69697000.000 N 69697000.000 E measured       1e+308 secs ago
   sensor:m_final_water_vx(m/s)=0                         1e+308 secs ago
   sensor:m_final_water_vy(m/s)=0                         1e+308 secs ago
   sensor:c_wpt_lat(lat)=0                                1e+308 secs ago
   sensor:c_wpt_lon(lon)=0                                1e+308 secs ago
   sensor:x_last_wpt_lat(lat)=3927.492                     13931 secs ago
   sensor:x_last_wpt_lon(lon)=-7413.635                    13931 secs ago
   sensor:m_battery(volts)=11.5033497532925                1.933 secs ago
```

    a. Note the highlighted region, this is the glider's GPS location. Note the age, this is a bad/old hit.

Appendix

1. Checklists

Glider Prep and Deployment Checklists

Glider - deployment		Project	
Deployment dates		Location/notes	

	Extant	Notes
1) Glider check-out sheet	☐	☐
2) Ballasting/dunk sheets	☐	☐
3) Deployment checklists (on boat, shore side)	☐	☐
4) Glider check-in sheet	☐	☐
5) Misc. (science, etc.)	☐	☐
	☐ CTD ☐ Optode	☐ LISST
6) Other	☐	☐

			MASS (g)	COMMENTS
Deployment	GLIDER	FORE STEM (minus FBB1,2)		
		FORE HULL		
		AFT STEM (red plug, card)		
Glider		AFT HULL		
		COWLING		
		SCREWS (vacuum, cowling, aft battery)		
Date	PAYLOAD	PAYLOAD BAY		
		WINGS		
		OTHER		
Preparer	BATTERIES	AFT BATTERY		
		PITCH BATTERY		
		FORE BATTERY 1, 2, EMER		
	WEIGHT BOTTLES	AFT BOTTLE		
		FORE BOTTLE 1 (stbd) (FBB1)		
		FORE BOTTLE 2 (port) (FBB2)		
		OTHER		

ENTIRE VEHICLE (Ohaus Scale)	

Tank Specifics		**Glider Specifics**	
Tank Density (kg/m^3)	1023.00	Glider Volume (L)	57.000
Tank Temperature (C)	23.00	Total Mass (kg)	0.000
Weight in Tank (g)	0.00	Glider Density (in air)	0.00
Target Specifics		**Volume Change (temperature induced)**	
Target Density (kg/m^3)	1023.00	Volume Change (target) (mL)	0.0
Target Temperature (C)	23.00	Coeffcient of Thermal Expansion	2.35E-05
		Carbon hulls	2.35E-05
Glider Volume (at lab temp) (L)	**0.000**	Aluminum hulls	7.00E-05

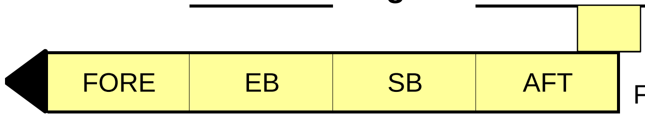
H MOMENT (rad)		**(deg)**	
Angle of Rotation (before)		0.0	
Angle of Rotation (after)		0.0	
Angle of Rotation	0	0.0	
Weight on Spring (after)			
Weight added	290		
Radius of Hull	107		125 for G2+, deeps
H-distance	####		

Ballasting Using Volume		**Ballasting Using Mass**	
Should Hang (in tank) (g)	0.0	**Adjust Glider Mass (entered volume) (g)**	58311.0
Adjust by (g)	0.0	**Glider Density (target water, using mass)**	**0.0**
Weight Change (no dunk) (g)			
Glider Density (target)	**1023.0**		

MISC MASSES & VOLUMES
Pick point - 40 mL - 107 g air - 66 g water
Wing Rail Weights - 1.8 mL @ 15.4 g each ~ 13.5 g in water
VMT Transceiver - 173 mL - 162 g water
FIRE Shroud SN02 (ru01) - 266 mL - 112 g water
Optode - 130 mL - 92 or 100 g (plastic or titanium)
LISST Bay - roughly 6.55 L

GLIDER: ________

Iteration ________ **Log File** ________ **Date / Location** ________



Ballast

FBB1 stbd ________

FBB2 Port ________

Aft BB ________

Front Scale ________ **Aft Scale** ________

Instrument: ________	**Instrument:** ________
T =	T =
C =	C =
D =	D =

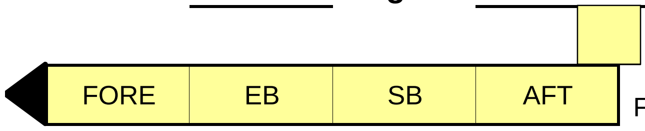
Roll ________

Ballast ________

Battery ________

Notes

Iteration ________ **Log File** ________ **Date / Location** ________



Ballast

FBB1 stbd ________

FBB2 Port ________

Aft BB ________

Front Scale ________ **Aft Scale** ________

Instrument: ________	**Instrument:** ________
T =	T =
C =	C =
D =	D =

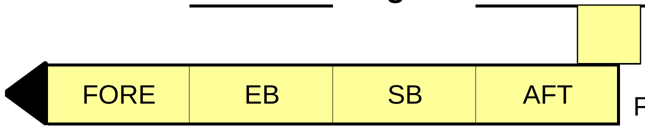
Roll ________

Ballast ________

Battery ________

Notes

Iteration ________ **Log File** ________ **Date / Location** ________



Ballast

FBB1 stbd ________

FBB2 Port ________

Aft BB ________

Front Scale ________ **Aft Scale** ________

Instrument: ________	**Instrument:** ________
T =	T =
C =	C =
D =	D =

Roll ________

Ballast ________

Battery ________

Notes

GLIDER	
PREPARER	
PREP DATE	
LOCATION / MISSION	
DENSITY @ TEMP	
INSURED?	

SCIENCE BAY SERIAL NUMBERS		Calibration Date (user/factory)
	1)	
	2)	
	3)	
	4)	
	5)	
	6)	

PRE-SEAL

TAKE PICTURES OF CONNECTORS AT EACH SEALING JOINT ________

FORE CHECK

Check pump & pitch threaded rods (clean and grease) ________

Grounded nose? ________

Dessicant Exposed? ________

Leak detect in place, batteries secure, grab & wiggle pitch battery to check secure, white guides free, no metal shavings, bottles installed ________ ________

PAYLOAD CHECK

Special Sensors / Additional Sensors? ________

CTD cable clear, no leak at CTD joint, no leak at pucks ________

Grounded? Fore Sci Ring ________ CTD ________

Corrosion? Aft Sci Ring ________ Other? ________

AFT CHECK

Iridium Card Installed (SIM #) (if not standard) ________

Flash Card Check (remove old files, backed up? See **Software** section) ________

Inspect strain on connectors/worn connectors ________

Battery secured ________

Ballast bottle present ________

Aft cap clean/clear of leak ________

Ejection weight stem grounded? Should it be? (Version specific) ________

Thruster greased? ________

Ensure safety of ballast pump prior to powering glider

Battery check: G2/G1 turn glider on with only 1 battery connected; G3 use BMS current

Aft Pack Voltage ________

Pitch Pack Voltage ________

Nose Packs Voltage ________

Emer (if possible) Voltage ________

Cabling/connectors - lithium vs. alkaline circuit correct? ________

POST-SEAL, pre-ballast

GENERAL

Pick Point Present? ________ Special Cargo? ________

HARDWARE

Nose cone and pump bladder inspection ________

Anode grounded? ________ Anode size / remainder ________

Pressure Sensor Check (corrosion, clear)

Aft sensor ________ Payload sensor ________

Ejection weight assembly ok/not seized? ________

POWERED

Put m_coulomb_amphr_total accordingly (0 = new batteries) ________
Put f_coulomb_battery_capacity (Alk=155, LiIon=200, li=450,625) ________
Vacuum @ T @ ballast ________ Stabilized m_battery ________
Get m_tot_num_inflections. Verify relative < 20000 or sufficient ________
Get m_leakdetect_voltage, science, forward (>2.3) ________
Get m_digifin_leakdetect_reading (less than 1019 requires service) ________
Altimeter test - put c_alt_time 0, verify chirp, note m_altimeter_voltage ________
Verify Argos ping ________ Wiggle for 5 minutes ________

SOFTWARE

(paths are RU specific)

GENERAL

Backup Glider and Science Cards ________
COOL//gliderData/glider_OS_backups/"glider name"
Format both CF cards - FAT Format ________
Apply new copy of latest TWR Software Image ________
For Glider: COOL/gliderData/gliderDos_releases/archived/"version"/target-glider
For Science: COOL/gliderData/gliderDos_releases/archived/"version"/target-science

Copy/overwrite STATE and CONFIG Folders ________
FW Transfer latest RU Software Image ________
COOL/Gliders/Glider Software Image/"use most recent image"
Software Version ________ Configure TBDlist ________
Date OK? ________ Configure NBDlist ________

\CONFIG

simul.sim deleted ________

\MAFILES

goto_l10.ma (set x_last_...) ________
yo*.ma, surfac*.ma pertinent for each glider and test missions ________

\MISSIONS

b_arg: undervolts: 10.75V alkaline, 9-10 V Li3S, 13.5V Li4S, 12.75V LiIon ________
Remove unused sample behaviors in missions ________

AUTOEXEC.MI

Iridium: Numbers may vary. Listed: Main - Rutgers Alternate - TWR
Irid Main: 88160000592 ________ Irid Alt: 17818711614 ________
u_iridium_failover_retries = 10 ________ Ver 7.15 u_iridium_idle -1? ________
sci timestamp sensors (ctd41cp) ________ Calibration coefficients ________
Reset the glider, observe any errors get f_max_working_depth ________

CACHE MANAGEMENT

del ..\state\cache\*.* ________
after *bdlist.dat are set (exit reset): ________
logging on; logging off ________
send ..\state\cache\*.cac ________
send *.mbd *.sbd *.tbd ________

DOCKSERVER

Version ________
Check script ________

TWR BACKUP

Confirm to-glider folder clear ________
Confirm correct script running ________

* **Software Burning Tips :** if using Procomm or local folder, copy all the files from the software image locally. Then proceed to edit them for the glider and do a mass freewave transfer of the files. Save these files or prepare the to-glider with these files

** Do a logging on for all these checks, take note of log and transfer before deployment*

SENSOR RETURN

put c_science_send_all 1	______
put c_science_all_on 8	______
put c_science_on 3	______
All sensors reporting values?	______

CTD

Tank static comparison OK?	______
Pumped CTD operational?	______
Plot ballast *BD log, sci_water_pressure non-noisy and near < .5 m	______

OPTODE

Check in completed?	______
Saturation reading in air	______

OPTICS

Check max return using fluoro sticks	______
Check dark counts with sensor covered	______
Optics file name	______

LISST

Clean LISST and perform ZSCAT	______

OTHER

______	______

OUTSIDE

GPS Alamanc/firmeware updated? ______

GPS check Latitude	______	Longitude	______
Iridium connect	______	Alternate number	______
zero_ocean_pressure	______	Get m_pressure	______
Air bladder shutoff (time)?	______	Sync_time (proper date?)	______
Compass calibration	______	Compass check	______

For deep gliders, put c_de_oil_vol -1000 to fully retract oil inside reservoir ______

ADDITIONAL

I *****WARNING: Advanced knowledge required to avoid damage/injury**

Check burn wire - disconnect, then put c_weight_drop 1, confirm 12 V ______

Fore leakdetect ______ Science ______ Aft leakdetect ______

THRUSTER

Report ++ m_thruster_current	______
Put c_thruster_on 20	______
Verify thruster spins clockwise and current value updates regularly	______
Put c_thruster_on 0 to turn off	______

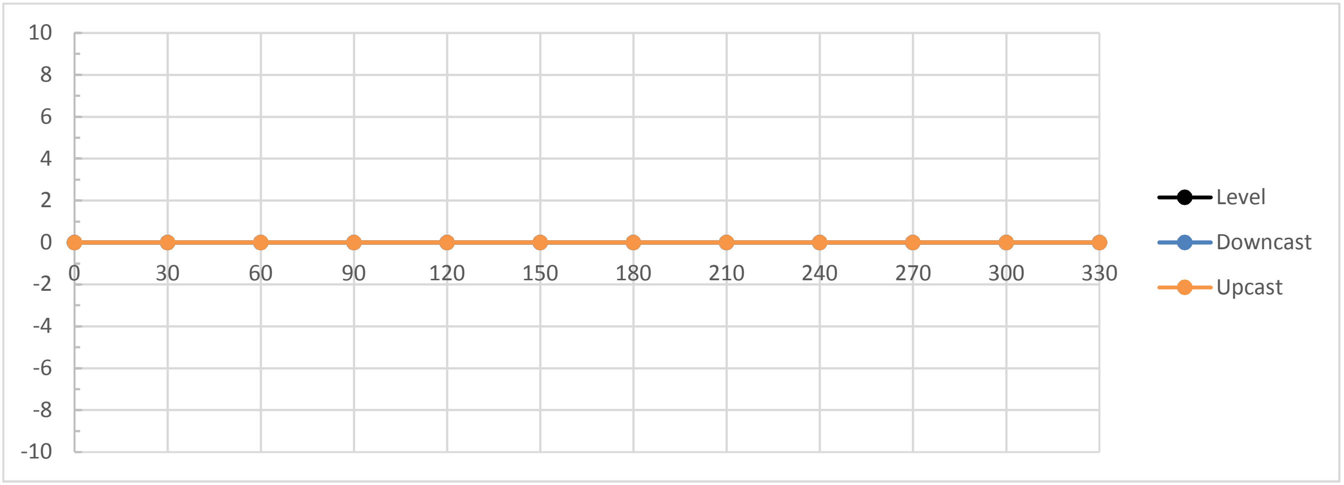
NOTES

Glider / Mission: ____________ **Cal Location** ____________ **Date** ____________ **Operator** ____________

	→ LEVEL →		
	HAND	GLIDER	ERROR
0			0
30			0
60			0
90			0
120			0
150			0
180			0
210			0
240			0
270			0
300			0
330			0

	↓ DOWNCAST ↓		
	HAND	GLIDER	ERROR
0			0
30			0
60			0
90			0
120			0
150			0
180			0
210			0
240			0
270			0
300			0
330			0

	↑ UPCAST ↑		
	HAND	GLIDER	ERROR
0			0
30			0
60			0
90			0
120			0
150			0
180			0
210			0
240			0
270			0
300			0
330			0



Glider ____________ Date ____________

Pilots ____________ Where ____________

Laptop ____________ Vehicle Powerup: CTRL ^ C (until you get to prompt)!!!

On boat **(Remember after 10 min glider will go into mission, as well as on powerup!)**	Battery Voltage	______	get m_battery
	Vacuum Pressure	______	get m_vacuum, should be > 7 for bladder inflation
	Iridium Connection	______	look for connect dialog & surface dialog, let it dial at prompt
	boot app	☐	boot app
	boot (should report application)	☐	reports boot application
	run status.mi	☐	mission completed normally?
In Water	zero_ocean_pressure	☐	while glider in water
	run od.mi (with or without float, ask RU)	☐	glider should dive and surface, type why? Should say overdepth, if not
	send *.dbd *.mlg *.sbd	☐	(FW) "send *.sbd" is most important
	(do nothing)		(IRID)
	run shallow.mi or deep.mi	☐	(glider should dive and not reappear) (report to Rutgers or steam out sl
	Verify dive; **disconnect freewave**		
	Report to Rutgers		
	send xxxxxx.mbd (from test mission)		(IRID)
	Perform CTD Comparison CAST	☐	typically done with RU provided SBE19 or Cast Away CTD
	LAT: **LON**		

Glider Deployment Checklist – Shore Side

Glider	**Date**	**Project/Location**

Field Participants, Vessel	**Pilot**

Pre-deployment

/to-glider folder populated & recent	☐	del large/numerous SBD & TBD's	☐

Glider Power Up - Pre-deploy

Confirm 'boot app' with 'boot'	☐		
Battery Voltage		m_vacuum (> 7)	
Coulomb AH total set	☐	Digifin & glider leakdetect OK	☐
sync_time (after GPS hit)	☐		

Glider In Water - Deployed

zero_ocean_pressure	☐	m_depth < 1 m	☐
run od, od5.mi - confirm overdepth abort	☐		
run shallow, deep.mi	☐		
Download shallow.mi MBD and NBD file		segment #:	
Boat – perform CTD comparison cast	☐	CTD s/n:	Laptop:

Test Mission Check

Vehicle Altimeter Works	☐
Flies to commanded depth and to surface	☐
Average vehicle roll	

+- 26 (or desired pitch) obtained, no overshoot or undershoot	☐		
Average battery position on dives and climbs			
Does vehicle track heading or m_heading cross c_heading	☐	Fin not hardover entire time (avg fin)	
Avg Dive Rate		Avg Climb Rate	

<u>Science Checks</u>

Surface Water Density		Bottom Water Density	
CTD and m_pressure agree	☐	Average offset	
CTD temp & salinity downcast = upcast (no lag)	☐		
Remaining Sensors reporting reasonable values	☐		

<u>Prepare for Primary Mission</u>

SBD/TBD's prior to od.mi deleted	☐
Transfer SBD's and TBD's	☐
Adjust yo to bottom if altimeter works	☐

<u>Post Dive</u>

Verify SBD and TBD are in tact	☐
Verify .cac availa for SBD/TBD	☐

RUTGERS
Center for Ocean Observing Leadership

Slocum Glider Check-IN

DATE: _____________ GLIDER: _________ SB: ________

Vehicle Powered

1. Power on vehicle in order to fully retract pump, and/or to deflate air bladder. ________
2. Wiggle vehicle for 5 minutes. _________

Vehicle Cleaning (hose down with pressure)

Nose cone ___________

1. Remove nose cone
2. Loosen altimeter screws, and remove altimeter or leave temporarily attached
3. Retract pump
4. Remove altimeter and hose diaphragm removing all sand, sediment, bio oils
5. Clean nose cone and altimeter

Tail cone ____________

1. Remove tail cone
2. Hose and clean anode and air bladder making sure air bladder is completely clean
3. Clean cowling

Wing rails ___________

1. Remove wing rails and hose down

Tail plug cleaning _________

1. Dip red plug in alcohol and clean plug if especially dirty
2. Re-dip red plug and repeatedly insert and remove to clean the glider plug
3. Compress air glider female connector
4. Lightly silicon red plug and replace in glider once silicon has been dispersed evenly in the plugs

CTD Comparison Check ___________

1. Inspect CTD sensor for any sediment buildup, take pictures of anything suspicious or make note.
2. Record results of Static Tank Test on CTD Check-in/out sheet

Optode Check/Calibration _____________

1. Record results on Optode Check Sheet

LISST Check/ZSCAT _____________

1. Record results on LISST Check Sheet

Vehicle Disassembled

1. Check leak points for water or salt buildup __________
2. **BACKUP FLASH CARDS** in /coolgroup/gliderData/glider_OS_backups/<glider>/<glider-deploymentID>/<from glider>,<from sb_0xxx> **** DO NOT DELETE DATA OFF CARDS****

____

3. Change permissions on <glider-deploymentID> folder to read, write, execute for owner and group, and read, execute for everyone ___________
4. Remove used batteries and place in return crate ________________
5. Re-assemble glider with a vacuum _______________

<u>Update Glider/Sensor History/Notes/Inventory</u> ____________

1. If needed, add notes to deployment page, glider binder, payloads binder, etc.

<u>Compile Deployment Checklist Packet Check</u> ____________

2. Print/fill out checklist packet title page
3. Make sure all pages are accounted for.
4. Scan entire packet and save to: /coolgroup/gliderData/deployments/<YEAR>/<glider-missionID>/meta/<Glider-missionID_checklists>
5. Put packet into the appropriate year deployment binder.

Slocum CTD Comparison Check

GLIDER: _______________ SB: ___________ DEPLOYMENT: _____________________

Pre-Deployment

Date: _____________

SBE19 s/n:	**Glider:**
Temperature:	**Temperature:**
Conductivity:	**Conductivity:**

Notes:

Post-Deployment

Date: _____________

SBE19 s/n:	**Glider:**
Temperature:	**Temperature:**
Conductivity:	**Conductivity:**

Notes:

******* CTD Maintenance if comparison is not acceptable (reference SeaBird Application Note 2D)

1. Perform CTD backward/forward flush with 1% Triton X-100 solution
2. Perform CTD backward/forward flush with 500 – 1000 ppm bleach solution
3. Perform the same on a pumped unit, just different approach
4. Repeat comparison test if above results not within T < .01 C, C < .005 S/m

Oxygen Optode Check & Calibration

OPTODE SN		**DATE**	
FOIL ID		**AIR PRESSURE (hPa)**	
PRE SALINITY		**CALIBRATED?**	

**** REMEMBER TO ISSUE THE SAVE COMMAND AFTER CHANGING VALUES***

100% SOLUBILITY		**TITRATION**		
	* µM = ppm * 1000 / 32	**EPA Na2S2O3 Check**		mL
		Sodium Sulfite / mL		0%

PRE-CHECK			
100%		**0%**	
Conc (µM) =		Conc (µM) =	
Saturation (%) =		Saturation (%) =	
Temp (°C) =		Temp (°C) =	
Phase =		Phase =	

POST-CAL			
100%		**0%**	
Conc (µM) =		Conc (µM) =	
Saturation (%) =		Saturation (%) =	
Temp (°C) =		Temp (°C) =	
Phase =		Phase =	

GLIDER CONFIG	
POST SALINITY	
TEXT OUTPUT OFF	

**** REMEMBER TO ISSUE THE SAVE COMMAND AFTER CHANGING VALUES***

RUTGERS
Center for Ocean Observing Leadership

Sequoia LISST Background Check-Out/Check-In

GLIDER: ____________ LISST: __________ DEPLOYMENT: __________________

How to Do a ZSCAT to collect background data

1. Obtain filtered Seawater and let sit out overnight to degas.
2. Cover LISST with black tape to create a chamber.
3. Slowly fill chamber with degassed FSW. Try not to create bubbles. Make sure chamber is not leaking.
4. Make sure there are no bubbles on the LISST sensor windows.
5. Cover the top of the chamber to make it dark.
6. Perform a zscat on the LISST to collect background data (u4stalk to LISST). Do 3 in a row that pass, and then save the zscat.
   consci, type proglets.dat, look up uart and bit in proglets
   u4stalk uart 9600 bit
   zs
7. Turn on the LISST to collect an RBN file. (through glider)
   put c_science_on 1
   put c_science_all_on_enabled 0
   put c_science_send_all 1
   put c_lisst_on 4
   put c_science_on 3
8. Wait a minute or two and then turn off the LISST
   put c_lisst_on -1
9. Write down RBN file name displayed on screen (sci_lisst_rbn1_file)

Check-Out, Pre-Deployment **Date:**		**Check-In, Post-Deployment** **Date:**	
Clean LISST windows use Lens Paper/Alcohol, don't scratch windows.	______________	Do NOT clean LISST windows.	______________
Perform ZSCAT (see above)	______________	Perform ZSCAT (see above)	______________
RBN file name	______________	RBN file name	______________
Once data saved off LISST, append to RBN filename: *_preMission_zscat*	______________	Once data saved off LISST, append to RBN filename: *_postMission_preCleaning_zscat*	______________
		Clean LISST windows use Lens Paper/Alcohol, don't scratch windows	______________
Notes:		Notes:	

2. Common Glider Commands

gliderDOS Command	Description / Use	Explanation
consci	consci [-f rf|irid] switch 'console' (control) to science computer via methods above	switch command to the science computer, 'quit' or lost carrier detect (ie: unplug freewave) for 5 seconds will return you to glider control
date	date [mdy[hms[a|p]]] /IEUMCP] date 01/01/10 16:00:00 set system clock (see sync_time)	sets system date
li_ion	query Li-Ion battery pucks see Lithium Ion Manual	li_ion toolset
longterm_put	longterm_put <sensor name> <new value>	add sensor value to the \state\longterm.sta file as well as permanently keep track of this sensors value across vehicle reset via \state\longterm.dat this has a higher functionality than autoexec.mi so be careful not to have the same sensor in either file
mbd OR sbd	mbd list mbd + <sensor_name> mbd - <sensor_name> mbd clearall mbd load <filename>	lists currently logged *.mbd sensors adds sensor to mbdlist.dat removes sensor from mbdlist.dat clears the mbdlist.dat loads mbd file listing, can load from any file (even mbdlist.dat) although preferred to send any file named x.dat to vehicle and then mbd load x.dat
put	put <sensor_name> <value_>	set a value of a vehicle sensor to a value (note: doesn't make sense to put m_x most of the time)
sequence	sequence <mission_name>(x) sequence <mission_name_1> <mission_name_2> … sequence -report sequence -resume OR -resume_next	run a mission x times run a sequence of different missions reports state of sequence resume the sequence or proceed to next mission in sequence
simul?	simul?	are we simulating and how
use	use use + <device_name_1> <device_name_2> … use - <device_name_1> <device_name_2> … use all use none	print report of devices, see 'device report explanation' put back into service following devices remove from service following devices use all devices use none
why?	why?	report to screen last abort time, mission, and reason
zs	zs -f rf|irid path\'file wildcard or name'	send files using zmodem Notes: need to specify path or be in current directory of the file. This has different functionality than the 'send' command, reccomended for all files other than *.*bd, *.mlg
ballast	ballast	shuts off air bladder, puts buoyancy engine to 0 cc (nuetral) and draws pitch battery back to 0 in (nuetral)
cd	cd path	change directory

dellog	dellog <all> or <mlg> <dbd> <sbd	deletes type of log files specified
longterm	longterm + <sensor_name> longterm - <sensor_name> longterm clearall longterm list	Similar to longterm_put but this requires you don't immediately set the value, but glider will remember the values across reboot. Effectively updates \config\longterm.dat with sensor to be remembered
run	run <mission_name>	runs a mission, not sequence, run once (if mission can end)
setdevlimit setnumwarn	setdevlimit <device_name> <os> <w/s> <w/m> setnumwarn x (num of warnings allowed, period)	advanced command: os - number of times a device can be put back in service w/s - allowable warnings/segment w/m - allowable warnings/minute
srf_display	srf_display + <sensor_name> srf_display - <sensor_name> srf_display clearall srf_display list	add sensor to config\config.srf to surface dialog remove sensor clear config.srf list sensors in config.srf
ver	ver	get software version
wiggle	wiggle on|off	move all motors to full extents indefinitely track usage errors with occasional use command
boot	boot boot application boot pico	report if OS or glider program is set to startup set gliderDOS to boot set PICODOS to boot (careful!)
get	get <sensor_name> <sensor_value>	report to screen last value
loadmission	loadmission <mission_name>	loads mission sensors & checks for syntax errors
report	report + <sensor_name_1> <sensor_name_2> … report ++ <sensor_name_1> <sensor_name_2> … report clearall	report a sensor or list of sensors on every data change report on each cycle stop reporting sensor values to screen
where	where	Spit surface dialog to console, useful for recovery
zero_ocean_pressure	zero_ocean_pressure	zeros ocean pressure to surface
callback	callback <xx(minutes)> <0(pri) or 1(sec)>	limited to 30 minutes maximum, remember 10 in gliderDOS without a command will start a sequence
lab_mode	lab_mode on|off	puts glider into gliderLAB, this mode will not sequence a mission after 10 minutes Important: this is only difference!
logging	logging on followed by logging off	logs data variables (*bd) files in gliderDOS or gliderLAB
send	send -f=rf|irid -num=<number_to_send> files	This is a smart command, typically used to send data files (*bd and mlg) files across freewave or iridium This command looks in the \logs folder and after the files are sent successfully they are moved to the \sentlogs folder
sync_time	sync_time	if you have GPS timestamp, system clock will sync to GPS time
type	type path\filename	type the contents of any file to the screen, very useful
zr	zr (dockzr) [options] file	[?][-v N][-nosmartdir][-f rf | irid][-cd dir][-noremove] [-archive] [-d <path1>] <file11> <file12> ... <file1N>
u4stalk	U4stalk uart baud bit	talk directly to science sensor. Uart and bit found in proglets.dat

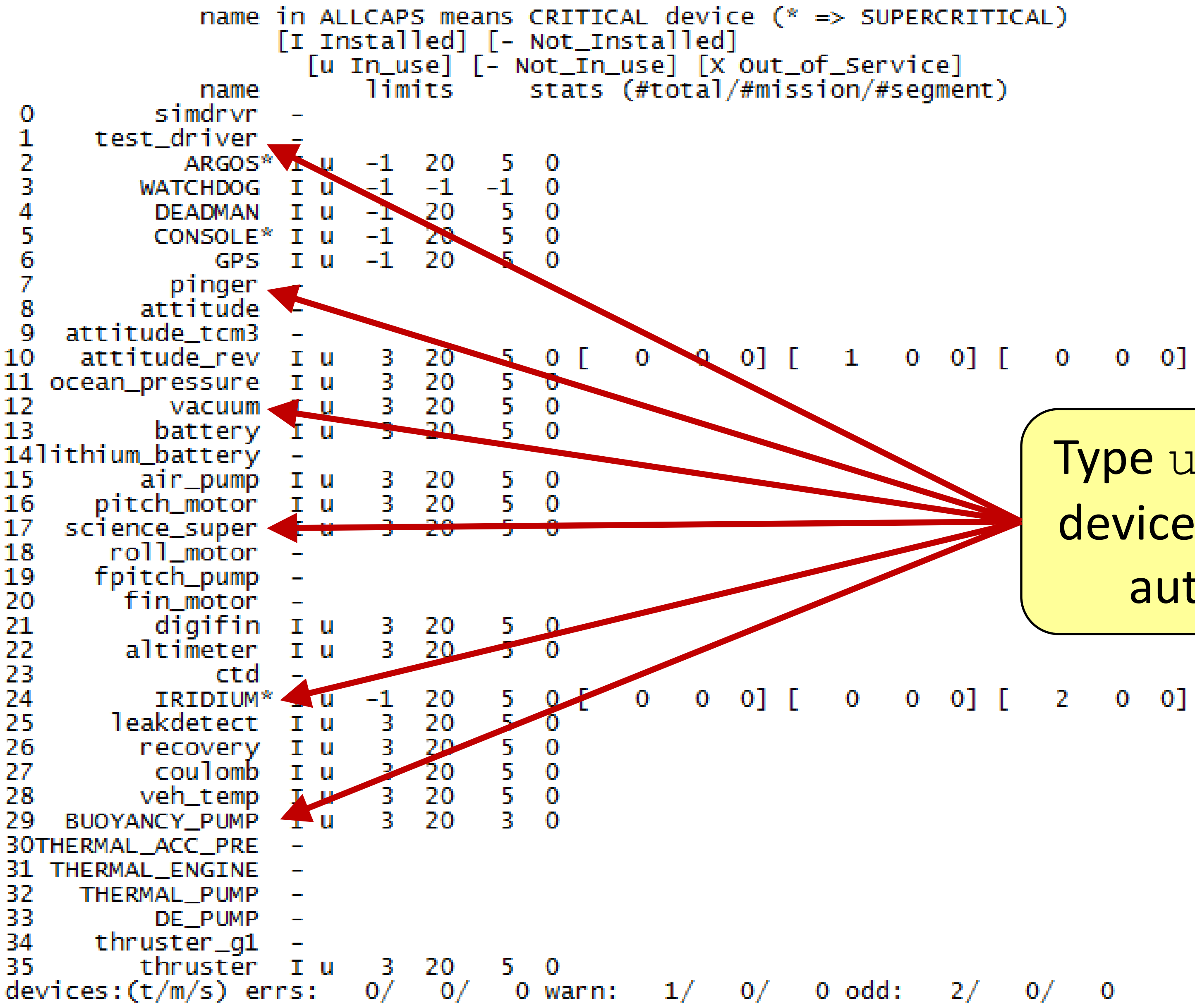
3. Installed Devices

TELEDYNE WEBB RESEARCH
Everywhereyoulook™

Installed Devices

```
              name in ALLCAPS means CRITICAL device (* => SUPERCRITICAL)
                   [I Installed] [- Not_Installed]
                     [u In_use] [- Not_In_use] [X Out_of_Service]
              name        limits     stats (#total/#mission/#segment)
 0        simdrvr  -
 1    test_driver  -
 2         ARGOS* I u  -1  20   5  0
 3       WATCHDOG I u  -1  -1  -1  0
 4        DEADMAN I u  -1  20   5  0
 5       CONSOLE* I u  -1  20   5  0
 6            GPS I u  -1  20   5  0
 7         pinger  -
 8       attitude  -
 9  attitude_tcm3  -
10   attitude_rev I u   3  20   5  0 [   0   0  0] [   1   0  0] [   0   0  0]
11 ocean_pressure I u   3  20   5  0
12         vacuum I u   3  20   5  0
13        battery I u   3  20   5  0
14lithium_battery  -
15       air_pump I u   3  20   5  0
16    pitch_motor I u   3  20   5  0
17  science_super I u   3  20   5  0
18     roll_motor  -
19    fpitch_pump  -
20      fin_motor  -
21        digifin I u   3  20   5  0
22      altimeter I u   3  20   5  0
23            ctd  -
24       IRIDIUM* I u  -1  20   5  0 [   0   0  0] [   0   0  0] [   2   0  0]
25     leakdetect I u   3  20   5  0
26       recovery I u   3  20   5  0
27        coulomb I u   3  20   5  0
28       veh_temp I u   3  20   5  0
29  BUOYANCY_PUMP I u   3  20   3  0
30THERMAL_ACC_PRE  -
31 THERMAL_ENGINE  -
32   THERMAL_PUMP  -
33        DE_PUMP  -
34    thruster_g1  -
35       thruster I u   3  20   5  0
devices:(t/m/s) errs:   0/   0/   0 warn:   1/   0/   0 odd:   2/   0/   0
```

Installed Devices



```
            name in ALLCAPS means CRITICAL device (* => SUPERCRITICAL)
                   [I Installed] [- Not_Installed]
                     [u In_use] [- Not_In_use] [X Out_of_Service]
            name          limits      stats (#total/#mission/#segment)
 0       simdrvr   -
 1     test_driver -
 2         ARGOS*  I u  -1  20   5  0
 3       WATCHDOG  I u  -1  -1  -1  0
 4        DEADMAN  I u  -1  20   5  0
 5        CONSOLE* I u  -1  20   5  0
 6            GPS  I u  -1  20   5  0
 7         pinger  -
 8       attitude  -
 9  attitude_tcm3  -
10   attitude_rev  I u   3  20   5  0 [    0    0   0] [    1    0   0] [    0    0   0]
11 ocean_pressure  I u   3  20   5  0
12         vacuum  I u   3  20   5  0
13        battery  I u   3  20   5  0
14lithium_battery  -
15       air_pump  I u   3  20   5  0
16    pitch_motor  I u   3  20   5  0
17  science_super  I u   3  20   5  0
18     roll_motor  -
19    fpitch_pump  -
20      fin_motor  -
21        digifin  I u   3  20   5  0
22      altimeter  I u   3  20   5  0
23            ctd  -
24       IRIDIUM*  I u  -1  20   5  0 [    0    0   0] [    0    0   0] [    2    0   0]
25     leakdetect  I u   3  20   5  0
26       recovery  I u   3  20   5  0
27        coulomb  I u   3  20   5  0
28       veh_temp  I u   3  20   5  0
29  BUOYANCY_PUMP  I u   3  20   3  0
30THERMAL_ACC_PRE  -
31 THERMAL_ENGINE  -
32   THERMAL_PUMP  -
33        DE_PUMP  -
34    thruster_g1  -
35       thruster  I u   3  20   5  0
devices:(t/m/s) errs:   0/    0/    0 warn:    1/    0/    0 odd:    2/    0/    0
```

Type `use` to see all devices installed in autoexec.mi

TELEDYNE WEBB RESEARCH
Everywhereyoulook™

Installed Devices

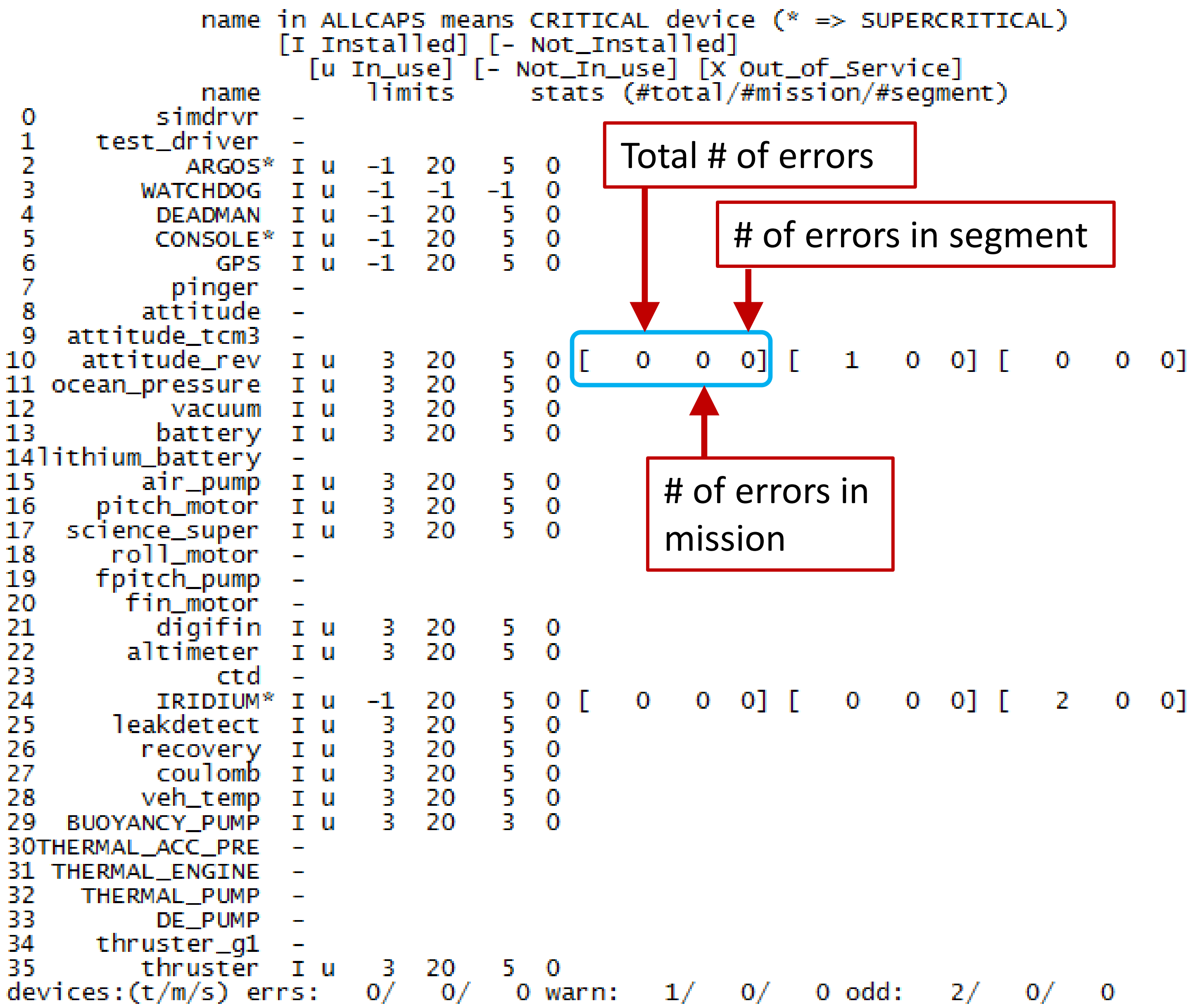
```
              name in ALLCAPS means CRITICAL device (* => SUPERCRITICAL)
                   [I Installed] [- Not_Installed]
                     [u In_use] [- Not_In_use] [X Out_of_Service]
              name         limits     stats (#total/#mission/#segment)
 0          simdrvr  -
 1      test_driver  -
 2            ARGOS* I u  -1  20   5  0
 3          WATCHDOG I u  -1  -1  -1  0
 4           DEADMAN I u  -1  20   5  0
 5          CONSOLE* I u  -1  20   5  0
 6               GPS I u  -1  20   5  0
 7           pinger  -
 8         attitude  -
 9    attitude_tcm3  -
10     attitude_rev  I u   3  20   5  0 [   0   0  0] [   1   0  0] [   0   0  0]
11   ocean_pressure  I u   3  20   5  0
12           vacuum  I u   3  20   5  0
13          battery  I u   3  20   5  0
14lithium_battery   -
15         air_pump  I u   3  20   5  0
16      pitch_motor  I u   3  20   5  0
17    science_super  I u   3  20   5  0
18       roll_motor  -
19      fpitch_pump  -
20        fin_motor  -
21          digifin  I u   3  20   5  0
22        altimeter  I u   3  20   5  0
23              ctd  -
24          IRIDIUM* I u  -1  20   5  0 [   0   0  0] [   0   0  0] [   2   0  0]
25       leakdetect  I u   3  20   5  0
26         recovery  I u   3  20   5  0
27          coulomb  I u   3  20   5  0
28         veh_temp  I u   3  20   5  0
29    BUOYANCY_PUMP  I u   3  20   3  0
30THERMAL_ACC_PRE    -
31  THERMAL_ENGINE   -
32    THERMAL_PUMP   -
33          DE_PUMP  -
34      thruster_g1  -
35         thruster  I u   3  20   5  0
devices:(t/m/s) errs:   0/   0/   0 warn:   1/   0/   0 odd:   2/   0/   0
```

Type `use - sensor_name` to temporarily remove devices

Type `use + sensor_name` to reinstall devices that have been taken out of service

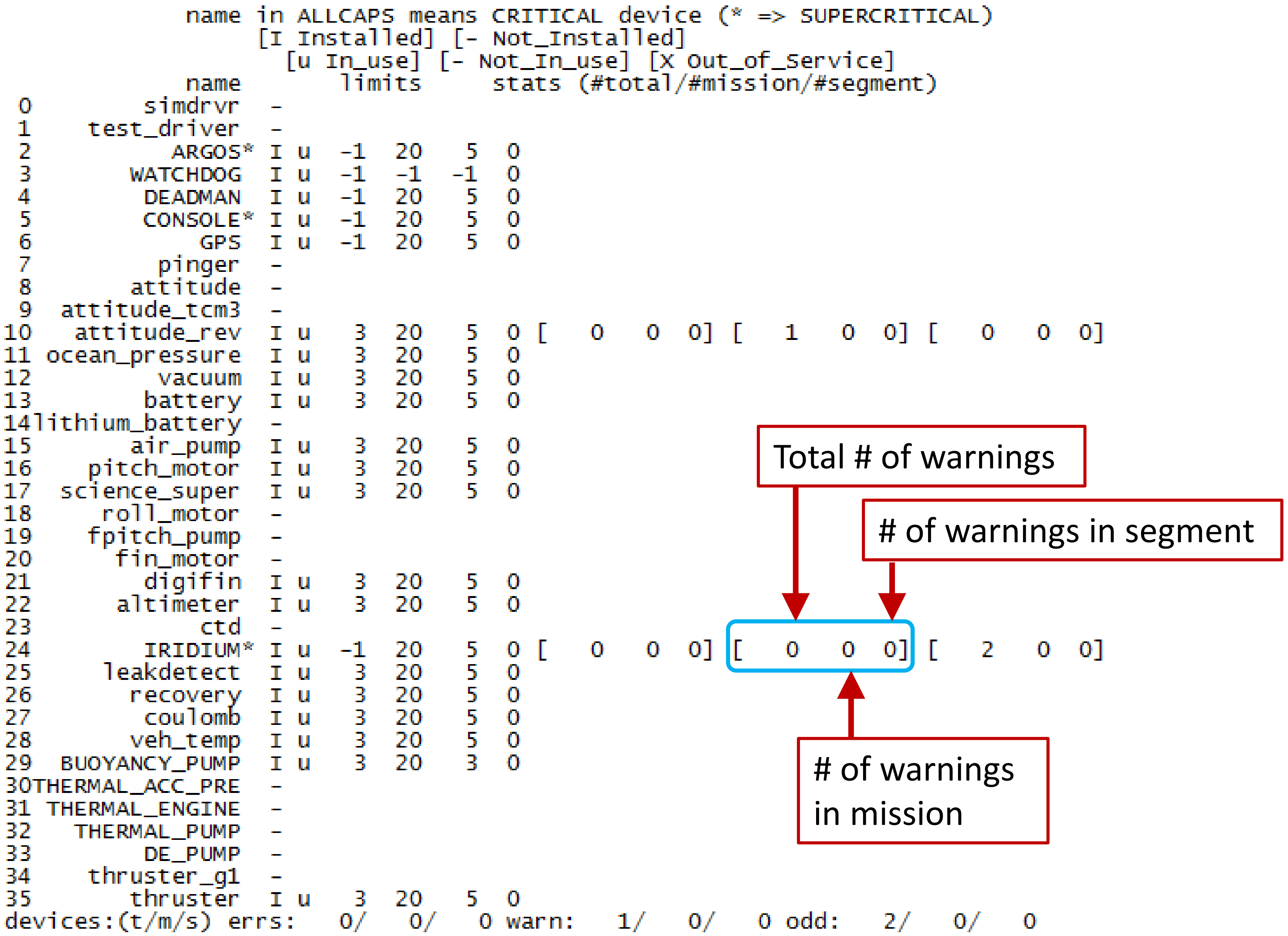
Type `use all` or `use none` to install or remove all devices

Installed Devices



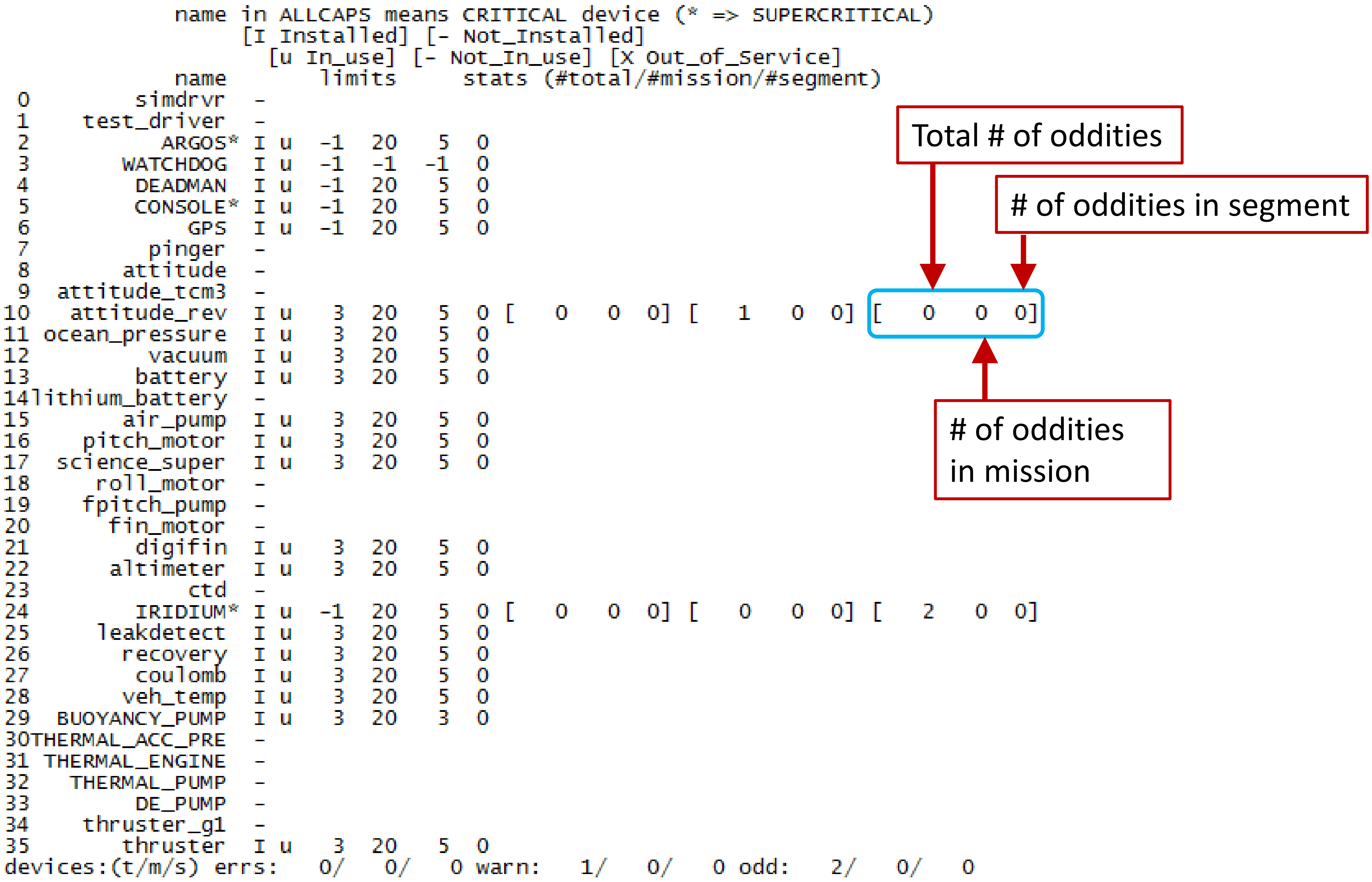
```
           name in ALLCAPS means CRITICAL device (* => SUPERCRITICAL)
                [I Installed] [- Not_Installed]
                  [u In_use] [- Not_In_use] [X Out_of_Service]
           name       limits     stats (#total/#mission/#segment)
 0       simdrvr  -
 1   test_driver  -
 2        ARGOS* I u  -1  20   5  0
 3      WATCHDOG I u  -1  -1  -1  0
 4       DEADMAN I u  -1  20   5  0
 5      CONSOLE* I u  -1  20   5  0
 6           GPS I u  -1  20   5  0
 7        pinger  -
 8      attitude  -
 9 attitude_tcm3  -
10  attitude_rev I u   3  20   5  0 [   0   0  0] [   1   0  0] [   0   0  0]
11 ocean_pressure I u  3  20   5  0
12        vacuum I u   3  20   5  0
13       battery I u   3  20   5  0
14lithium_battery -
15      air_pump I u   3  20   5  0
16   pitch_motor I u   3  20   5  0
17 science_super I u   3  20   5  0
18    roll_motor  -
19   fpitch_pump  -
20     fin_motor  -
21       digifin I u   3  20   5  0
22     altimeter I u   3  20   5  0
23           ctd  -
24      IRIDIUM* I u  -1  20   5  0 [   0   0  0] [   0   0  0] [   2   0  0]
25    leakdetect I u   3  20   5  0
26      recovery I u   3  20   5  0
27       coulomb I u   3  20   5  0
28      veh_temp I u   3  20   5  0
29 BUOYANCY_PUMP I u   3  20   3  0
30THERMAL_ACC_PRE -
31 THERMAL_ENGINE -
32  THERMAL_PUMP  -
33       DE_PUMP  -
34   thruster_g1  -
35      thruster I u   3  20   5  0
devices:(t/m/s) errs:   0/   0/   0 warn:   1/   0/   0 odd:   2/   0/   0
```

Installed Devices



```
             name in ALLCAPS means CRITICAL device (* => SUPERCRITICAL)
                  [I Installed] [- Not_Installed]
                    [u In_use] [- Not_In_use] [X Out_of_Service]
             name       limits     stats (#total/#mission/#segment)
 0        simdrvr  -
 1    test_driver  -
 2          ARGOS* I u  -1  20   5  0
 3       WATCHDOG  I u  -1  -1  -1  0
 4        DEADMAN  I u  -1  20   5  0
 5        CONSOLE* I u  -1  20   5  0
 6            GPS  I u  -1  20   5  0
 7         pinger  -
 8       attitude  -
 9  attitude_tcm3  -
10   attitude_rev  I u   3  20   5  0 [   0   0  0] [   1   0  0] [   0   0  0]
11 ocean_pressure  I u   3  20   5  0
12         vacuum  I u   3  20   5  0
13        battery  I u   3  20   5  0
14lithium_battery  -
15       air_pump  I u   3  20   5  0
16    pitch_motor  I u   3  20   5  0
17  science_super  I u   3  20   5  0
18     roll_motor  -
19    fpitch_pump  -
20      fin_motor  -
21        digifin  I u   3  20   5  0
22      altimeter  I u   3  20   5  0
23            ctd  -
24        IRIDIUM* I u  -1  20   5  0 [   0   0  0] [   0   0  0] [   2   0  0]
25     leakdetect  I u   3  20   5  0
26       recovery  I u   3  20   5  0
27        coulomb  I u   3  20   5  0
28       veh_temp  I u   3  20   5  0
29  BUOYANCY_PUMP  I u   3  20   3  0
30THERMAL_ACC_PRE  -
31 THERMAL_ENGINE  -
32   THERMAL_PUMP  -
33        DE_PUMP  -
34    thruster_g1  -
35       thruster  I u   3  20   5  0
devices:(t/m/s) errs:   0/   0/   0 warn:   1/   0/   0 odd:   2/   0/   0
```

Installed Devices



```
             name in ALLCAPS means CRITICAL device (* => SUPERCRITICAL)
                  [I Installed] [- Not_Installed]
                    [u In_use] [- Not_In_use] [X Out_of_Service]
             name      limits     stats (#total/#mission/#segment)
 0         simdrvr -
 1     test_driver -
 2           ARGOS*I u  -1  20   5  0
 3        WATCHDOG I u  -1  -1  -1  0
 4         DEADMAN I u  -1  20   5  0
 5         CONSOLE*I u  -1  20   5  0
 6             GPS I u  -1  20   5  0
 7          pinger -
 8        attitude -
 9   attitude_tcm3 -
10    attitude_rev I u   3  20   5  0 [   0   0  0] [   1   0  0] [   0   0  0]
11  ocean_pressure I u   3  20   5  0
12          vacuum I u   3  20   5  0
13         battery I u   3  20   5  0
14lithium_battery -
15        air_pump I u   3  20   5  0
16     pitch_motor I u   3  20   5  0
17   science_super I u   3  20   5  0
18      roll_motor -
19     fpitch_pump -
20       fin_motor -
21         digifin I u   3  20   5  0
22       altimeter I u   3  20   5  0
23             ctd -
24         IRIDIUM*I u  -1  20   5  0 [   0   0  0] [   0   0  0] [   2   0  0]
25      leakdetect I u   3  20   5  0
26        recovery I u   3  20   5  0
27         coulomb I u   3  20   5  0
28        veh_temp I u   3  20   5  0
29   BUOYANCY_PUMP I u   3  20   3  0
30THERMAL_ACC_PRE -
31  THERMAL_ENGINE -
32    THERMAL_PUMP -
33         DE_PUMP -
34     thruster_g1 -
35        thruster I u   3  20   5  0
devices:(t/m/s) errs:   0/   0/   0 warn:   1/   0/   0 odd:   2/   0/   0
```

TELEDYNE WEBB RESEARCH
Everywhereyoulook™

Installed Devices

```
          name in ALLCAPS means CRITICAL device (* => SUPERCRITICAL)
               [I Installed] [- Not_Installed]
                 [u In_use] [- Not_In_use] [X Out_of_Service]
          name       limits      stats (#total/#mission/#segment)
 0      simdrvr  -
 1  test_driver  -
 2       ARGOS*  I u  -1  20   5  0
 3      WATCHDOG I u  -1  -1  -1  0
 4      DEADMAN  I u  -1  20   5  0
 5     CONSOLE*  I u  -1  20   5  0
 6          GPS  I u  -1  20   5  0
 7       pinger  -
 8     attitude  -
 9 attitude_tcm3 -
10  attitude_rev I u   3  20   5  0 [   0   0  0] [   1   0  0] [   0   0  0]
11 ocean_pressure I u  3  20   5  0
12       vacuum  I u   3  20   5  0
13      battery  I u   3  20   5  0
14lithium_battery -
15     air_pump  I u   3  20   5  0
16  pitch_motor  I u   3  20   5  0
17 science_super I u   3  20   5  0
18   roll_motor  -
19  fpitch_pump  -
20    fin_motor  -
21      digifin  I u   3  20   5  0
22    altimeter  I u   3  20   5  0
23          ctd  -
24     IRIDIUM*  I u  -1  20   5  0 [   0   0  0] [   0   0  0] [   2   0  0]
25   leakdetect  I u   3  20   5  0
26     recovery  I u   3  20   5  0
27      coulomb  I u   3  20   5  0
28     veh_temp  I u   3  20   5  0
29 BUOYANCY_PUMP I u   3  20   3  0
30THERMAL_ACC_PRE -
31 THERMAL_ENGINE -
32  THERMAL_PUMP -
33      DE_PUMP  -
34  thruster_g1  -
35     thruster  I u   3  20   5  0
devices:(t/m/s) errs:   0/   0/   0 warn:   1/   0/   0 odd:   2/   0/   0
```

Can modify value of SETDEVLIMIT & SETNUMWARN to increase/decrease glider sensitivity

4. SOP for programming Freewave radios

1. Open terminal emulation program (i.e. Procomm or Hyperterminal).
2. Set communication preferences – comm. port (typically port 1 if plugged into back of laptop), baud rate 19200 bps, data bits 8, parity none, and stop bits 1.
3. Plug Freewave in via 9 pin serial cable - one end to db-9 connector on Freewave, one end to db-9 connector on laptop. Plug Freewave power cord in. Lights on front of Freewave should all be red.
4. Using a pen, press small black button on back of Freewave, labeled ‘setup’ or ‘reset’. A menu should pop up in the terminal, and the lights on the front of the Freewave should all be green.
5. Press ‘2’ for ‘Edit Call Book’.
6. If the glider’s phone # is not listed, press the number of the entry to change. In the example below, press ‘1’, then add the new phone #. When asked, press escape, as no repeater is used in this setup.

```
                     MODEM CALL BOOK
                     Entry to Call is (ALL)
Entry   Number    Repeater1   Repeater2
(0)    896-4197
(1)    000-0000
(2)    000-0000
(3)    000-0000
(4)    000-0000
(5)    000-0000
(6)    000-0000
(7)    000-0000
(8)    000-0000
(9)    000-0000
(C)    Change Entry to Use (0-9) or A(ALL)
(Esc)  Exit to Main Menu
Enter all zeros (000-0000) as your last number in list
```

7. Press ‘C’ to ‘Change entry to use’. Select the number to call, or, if only one glider is being used at a time, ‘All’ can be selected. This will make the master Freewave talk to any glider whose number is listed in the Modem Call Book.
8. Press ‘Esc’ to go back to main menu, then press ‘Esc’ again to exit. The lights on the front of the Freewave should be red again, unless it is connecting to a glider.

5. Glider Missing Writing

SLOCUM Glider Mission Writing

```
--------------------------------------------------------------------------------
Authors: John Kerfoot, David Aragon
Email: kerfoot@marine.rutgers.edu, dkaragon@marine.rutgers.edu
Creation Date: March 5, 2010
--------------------------------------------------------------------------------
```

Introduction

The Slocum Coastal Electric Glider is commanded via user-generated missions executed from the glider shell (GliderDOS). GliderDOS is a subset of picoDOS, which itself is a subset of the Disk Operating System (DOS). As such, the file system uses the 8.3 naming convention, meaning that all files are restricted to 8 characters for the filename and 3 characters for the file extension.

Missions are comprised of the following file-types:

- MISSION files (.mi extension)
- MAFILES files (.ma extension)

While all of the mission directives can be specified in .mi files only, use of .ma files allows for much more flexibility in altering intended waypoints and science payload sampling as well as changing surface intervals, all of which may be done frequently during a deployment as deployment goals change.

Mission files are composed of **behaviors** (**behavior**: prefix), which themselves are composed of **behavior arguments** (**b_arg**: prefix) and **sensor** values (**sensor:** prefix). A typical mission file contains a series of behaviors and assigns priorities to each behavior to create a behavior stack or log_c_stack. This stack is then used to control the glider. Once a mission is started, the glider cycles through the behavior stack to determine which behavior is given priority. Once a behavior is started, it must finish before another behavior can begin.

Background: BEHAVIORS

A mission should always contain at least one instance of each of the following 7 behaviors:

- **behavior: abend**
- **behavior: surface**
- **behavior: goto_list**
- **behavior: yo**
- **behavior: sample**
- **behavior: prepare_to_dive**
- **behavior: sensors_in**

Each of these behaviors plays a critical role in defining the mission parameters. The first 5 behaviors (**abend**, **surface**, **goto_list**, **yo**, and **sample**) are the most commonly modified behaviors, while the last 2 must ALWAYS be present in the mission, but are almost never modified by the end-user. As such, this document focuses on the 5 behaviors commonly used to modify glider behavior by the end user.

The following locations of the pertinent documents provided below are referenced from the top-level directory contained in the gliderDOS **production_*** folder:

- ./src/doco: this directory contains all C-language source code and header files for all sensor definitions and behaviors. Perusing this source can provide a more detailed understanding of how each sensor and behavior actually work.
- ./src/code/masterdata: this text document contains the default values for all sensors known to the glider. Any **sensor:** not directly set in a mission or ma file takes it's default value from masterdata. The default **sensor:** values are set from this file during the compilation of the source code. This is a critical point as any change in a value in this file will **NOT** take effect unless the source code is re-compiled.
  Typically, new **sensor:** values are set directly in the glider's autoexec.mi file, a mission file or via the GliderDos prompt.
- ./src/doco: contains readme files on the general design/description of the glider, filetypes and specfications.

The following is a summary of the behaviors used to control sampling on the slocum glider. Default values for each **b_arg** are specified in the masterdata document and are used when not explicitly specified in the mission or ma file.

Please consult the masterdata version corresponding to the software version your glider is running. There are a number of documents contained in the source code release that can provide a complete description of all **sensor:** values as well as **behavior:** and **b_arg:** parameters.

behavior: abend
The **abend** behavior defines all of the critical reasons for the glider to abort the mission, climb to the surface and call the dockserver. Any abort triggered by the behavior arguments listed in this behavior can be thought of as a software abort, meaning that while significant, the deployment can most likely be completed.

behavior: surface
The **surface** behavior defines all of the reasons for the glider to surface and call the dockserver as well as the settings that define the climb to the surface (ie: pitch angle for the climb).

behavior: goto_list
The **goto_list** behavior defines the following:
- The mission waypoints
- The starting waypoint
- The number of times to cycle through the waypoint list
- The distance the glider must be from the current waypoint to satisfy the behavior.

behavior: yo
The **yo** behavior defines:
- The dive to and climb to depths
- The dive and climb pitch angles
- The number of yos (dive/climb cycles) to perform
- The minimum height off of the bottom

behavior: sample
The **sample** behavior, along with **sensor:** settings, controls the sampling state of each of the science payload sensors. It allows the user to turn on/off instruments, control the sampling rate of individual instruments and the portion

of the yo (dive, climb, inflecting, hovering) for which the instrument is sampling.

behavior: prepare_to_dive
The **prepare_to_dive** behavior tells the glider how long to wait, while on the surface, to obtain a valid gps fix. Following the successful or unsuccessful GPS acquisition, the behavior then prepares for the dive (deflates airbag, turns off argo and turns off surface sensors) and, finally, intitiates the dive.

This behavior is typically place at the bottom of the behavior stack to initiate a dive.

behavior: sensors_in
The **sensors_in** behavior simply sets input sensor sampling times. By default it turns all the input sensors off.

This behavior is typically put at the bottom (lowest priority) of the behavior stack to shut everything off and then higher priority behaviors turn them on as needed or specified by the **sample** behaviors.

Mission Design

Mission are designed and written in 2 distinct ways:

- All behaviors and sensor values are placed in the mission (**.mi**) file. While this consolidates the mission into a single file, any changes to the mission parameters (dive/climb depths, waypoint lists, surfacing intervals, etc.) require that the mission be exited (via **^C**) and restarted before the changes take effect. The new mission file must be transferred to the glider and then the mission must be restarted. This process significantly increases the time the glider spends on the surface as well as the associated Iridium satellite communication cost.
- While all behaviors and sensors values are placed in the mission file, most of the specific behavior arguments (**b_arg:**) defining the behaviors are specified in associated .ma files. The mission (.mi) defines the names of these .ma files using a specific behavior argument (**b_arg: args_from_file(enum)**) so that the glider knows which mafiles to include in the specified mission. This style of mission writing allows for a single mission to be run by a fleet of gliders while allowing the end-user to customize a single glider’s behavior to achieve the mission end goals. The most significant drawback of this style of mission is an increased complexity in the number of and types of files required.

Our recommendation is to use the second mission style explained above and the rest of this document details the requirements for writing missions using this style.

Mission Writing:

This section covers the specifics of each of the 7 main behaviors contained in a mission file and the behavior arguments (**b_arg:**) used to define them. The referenced mission file is **tn.mi** and is included with this manual. This mission, besides defining safety settings, dive parameters and sampling, allows for a surface event to be triggered based upon a timeout interval as well as a surface event to be triggered when the glider has not established communications for a specified time interval.

COMMENTS
Mission (**.mi**) and ma (**.ma**) files can be commented by prefacing the comment with a pound (#) sign. Everything after the # is recognized as a comment and ignored by the glider. Commenting of missions and ma files is strongly encouraged. For example, the **tn.mi** mission contains a commented header that describes the mission and lists the names of all supporting ma files used to customize the mission parameters.

BEHAVIORS:

behavior: abend
The **abend** behavior is a behavior which monitors various mission safety conditions and triggers a software abort if any of the conditions are violated. These types of aborts are almost always recoverable; that is, they almost never require the recovery of the glider. The **abend** behavior in **tn.mi** is displayed below:

```
# ABORT BEHAVIOR
behavior: abend

    # OVERDEPTH: glider finds itself in > 204m of water
    b_arg: overdepth(m)                     204
    b_arg: overdepth_sample_time(sec)       12

    # OVERTIME: disabled
    b_arg: overtime(sec)                    -1.0

    # COP TICKLE: watchdog not tickled for 24 hours
    b_arg: no_cop_tickle_for(sec)           50400.0
    b_arg: no_cop_tickle_percent(%)         -1

    # SAMEDEPTH: 15 min every 60 seconds
    b_arg: samedepth_for(sec)               900.0
    b_arg: samedepth_for_sample_time(sec)   60.0

    # STALLED FOR: 15 min every 60 seconds
    b_arg: stalled_for(sec)                 900.0
    b_arg: stalled_for_sample_time(sec)     60.0

    # HARDWARE: vacuum, battery
    b_arg: undervolts(volts)                9.5
    b_arg: undervolts_sample_time(sec)      12.0

    b_arg: vacuum_max(inHg)                 13.0
    b_arg: vacuum_min(inHg)                 5.0
    b_arg: vacuum_sample_time(sec)          36.0

    # WAYPOINT TOO FAR ABORT: 500km (500,000m)
    b_arg: max_wpt_distance(m)              500000
```

The rest of the more common arguments are discussed below:

- **b_arg: overdepth(m):** triggers an abort if the glider exceeds this depth, measured in meters.

- **b_arg: overtime(sec):** triggers an abort if the total mission time exceeds this value, measured in seconds.
- **b_arg: no_cop_tickle(sec):** the glider contains a hardware jumper controlling a watchdog circuit, which initiates the burnwire sequence used to eject the emergency weight at the aft of the glider. As long as the software is operating properly and “tickling” the watchdog circuit, the weight is not ejected. If the jumper time is exceeded, the burnwire sequence is intiated and the weight is ejected, necessitating a recovery of the glider. There are 2 different watchdog circuit configurations: a 2 hour and 16 hour version.
  This behavior argument instructs the glider to trigger a “software” abort if the watchdog circuit has not been tickled for this amount of time, specified in seconds. In practice, the value specified should always be less than the length of the watchdog timer (either 2 or 16 hours).
  In this example, with a glider that has a 16 hour watchdog circuit, we’ve specified that a software abort should be triggered at 50400 seconds (14 hours), giving us a 2 hour window before the hardware generates an abort.
- **b_arg: same_depth_for(sec):** triggers an abort if the glider is stuck at depth for the specified time, measured in seconds.
- **b_arg: stalled_for(sec):** triggers an abort if the glider is not moving horizontally through the water for the specified time, measured in seconds.
- **b_arg: undervolts(volts):** triggers an abort if the battery voltage on the glider drops below the specified value, measured in volts. This is more of an issue with alkaline batteries than with lithium primaries or secondaries.
- **b_arg: vacuum_max(inHg):** triggers an abort if the internal vacuum exceeds the specified value, measured in inches of Mercury.
- **b_arg: vacuum_min(inHg):** triggers an abort if the internval vacuum drops below the specified value, measured in inches of Mercury.
- **b_arg: max_wpt_distance(m):** triggers an abort if the distance to the current waypoint exceeds the specified value, measured in meters.

All behavior arguments (**b_arg**:) ending with ***sample_time(sec)** instruct the glider to check for the corresponding condition every **x** seconds. Setting the ***sample_time(sec)** behavior argument to a value less than 0 disables the check.

For example:

```
    b_arg: overdepth(m)                    204
    b_arg: overdepth_sample_time(sec)      12
```

The first line instructs the glider to check that it’s current depth does not exceed 204 meters. The second line specifies how often to check for this condition.

An abort code, corresponding to the type of abort triggered by the behavior argument, is generated in the event this behavior is triggered. Here are a few examples of a specific abort type and the resulting code:

- **MS_ABORT_OVERDEPTH**: results from exceeding the depth specified in **b_arg: overdepth(m)**.
- **MS_ABORT_OVERTIME**: results from exceeding the maximum allowable mission time specified in **b_arg: overtime(sec)**.

- **MS_ABORT_WPT_TOOFAR**: results from exceeding the maximum allowable distance to the current waypoint, as specified by **b_arg: max_wpt_distance(m).**
- **MS_ABORT_BEH_ERROR**: normally results from an improperly defined behavior in a mission or ma file such as syntax or wrong number of waypoints in **behavior: goto_list.**

The full list of abort codes is specified in the gliderDOS release under:

```
./src/code/mission_status.h
```

Each code has a numeric termination code (also listed in command.h) that is displayed as part of the GliderDos prompt after the abort. For example, in the event that an overdepth abort was triggered, the GliderDos prompt would appear as:

GliderDos A 7 >

A more in-depth explanation of how aborts are triggered is presented in the GliderDos source under:

```
./src/doco/how-it-works/abort-sequences.txt
```

behavior: surface
Surface behaviors define the conditions under which a glider should attempt to surface as well as what to do while on the surface. The **tn.mi** mission utilizes 5 surfacing behaviors. All five surface behaviors are listed below:

```
# SURFACE BEHAVIOR: NOTHING COMMANDED
behavior: surface
    b arg: args from file(enum)     10  # mafiles/surfac10.ma
    b_arg: start_when(enum)          1  # 1-stack idle

# SURFACE : No waypoints commanded
behavior: surface
    b arg: args from file(enum)     10  # mafiles/surfac10.ma
    b_arg: start_when(enum)          3  # 3-heading idle

# SURFACE : SURFACING INTERVAL
behavior: surface
    b_arg: args_from_file(enum)     20 # mafiles/surfac20.ma
    b_arg: start_when(enum)         9 # 9-every when_secs

# SURFACE : HIT A WAYPOINT
behavior: surface
    b arg: args from file(enum)     30  # mafiles/surfac30.ma
    b_arg: start_when(enum)          8  # 8-when hit waypoint

# SURFACE : NO COMMS
behavior: surface
    b arg: args from file(enum)     40  # mafiles/surfac40.ma
    b_arg: start_when(enum)         12  # 12 No comms
```

A surface behavior can be activated under a number of conditions. You may specify what condition/state the glider must be in before a behavior becomes active by using the **b_arg: start_when(enum)** behavior argument. The full list of codes is contained in the GliderDos release source under:

```
    ./src/code/beh_args.h
```

The behavior becomes active when this condition is met. For example, the first surface behavior has a **b_arg: start_when(enum)** value set to **1**, which corresponds to the command stack being idle. This occurs when the glider is not being commanded, ie: not doing anything. All surface behaviors should have this behavior argument specified.

Each of the surface behaviors contain another behavior argument, **b_arg: args_from_file(enum)**. This behavior argument allows the behavior to be defined using an **.ma** file. In the case of the surface behavior, the ma filename is created by appending the number specified in **b_arg: args_from_file(enum)** to the string "surfac", which results in **surfac10.ma**. This file must be placed in the MAFILES directory under the glider root. The contents of the **surfac10.ma** file for **tn.mi** are displayed below:

```
# Getting to the surface
behavior_name=surface

<start:b_arg>

    b_arg:  c_use_bpump(enum)          2
    b_arg:  c_bpump_value(X)        1000

    b_arg:  c_use_pitch(enum)          3 # servo, rad, >0 = climb
    b_arg:  c_pitch_value(X)      0.4528 # 26 degrees

<end:b_arg>
```

All behaviors defined in an external **.ma** file must have obey the following syntax rules:

1) The first non-comment line must contain the following line: behavior_name=***BEHAVIOR*** where ***BEHAVIOR*** is the desired behavior name (ie: surface, goto_list, etc.)
2) The behavior arguments must be enclosed in **<start:b_arg><end:b_arg>** tags.

There are 2 pieces to this behavior which define the use of the air bladder and the glider pitch for getting to the surface. The 2 behavior arguments containing **c_use_*** specify how the glider should measure each of the corresponding movements.

For example, **b_arg: c_use_bpump(enum) 2** instructs the glider to push the buoyancy pump forward by an absolute volume (cc) measurement, which is specified by the next behavior arguments **b_arg: c_bpump_value(X) 1000** (1000 cc), which is clipped to maximum allowable volume.

The next pair of behavior arguments defines the glider pitch. The first argument, **b_arg: c_use_pitch(enum) 3** instructs the glider to move the pitch battery using feedback provided by the pitch servomechanism (or **servo,** for short). In other words, set the pitch angle to 0.4528 (**b_arg: c_pitch_value(X) 0.4528**) radians (or 26 degrees).

To sum up, this surface behavior tells the glider to climb to the surface at 0.4528 radians (26 degrees) and to fully inflate the air bladder.

A complete listing of device modes can be found near the end of the masterdata document.

The last 3 surface behaviors used in this mission control when the glider should surface. As each behavior contains the **b_arg: args_from_file(enum)** argument, each of the conditions making the behavior active are contained in the corresponding surfac*.ma file.

Surfacing at a Specified Time Interval:
This mission specifies a surfacing behavior triggered at a specified interval through the surfac20.ma file. The argument specifying when the behavior becomes active is:

b_arg: start_when(enum) 9

which corresponds to exceeding the specified time interval contained in surfac20.ma:

```
behavior_name=surface

# SURFAC20.MA (Timeout Expired)

<start:b_arg>
    # Surface for timeout expired
        b_arg:  when_secs(sec)          3600

    # Flight Controls
        b arg:  c use bpump(enum)       2
        b_arg:  c_bpump_value(X)        1000
        b_arg:  c_use_pitch(enum)       3       # servo, rad, >0 = climb
        b_arg:  c_pitch_value(X)        0.4528  # 26 degrees

    # Surface Timeouts & Other Params
        b arg: report all(bool)             0       # F->just gps
        b_arg: end_action(enum)             1       # Surface menu
        b_arg: gps_wait_time(sec)           600     # GPS wait
        b_arg: keystroke_wait_time(sec)     300     # Surface time
        b arg: printout cycle time(sec)     40.0    # Surface menu print rate
<end:b_arg>
```

The first behavior argument specified here is:

b_arg: when_secs(sec)   3600

The behavior instructs the glider to surface once every hour, regardless of any other surfacing behavior. The next 4 **b_arg:** parameters are identical to those specified in the prior surfacing behavior (surfac10.ma) and describe how the glider will surface (ie: at a pitch of 0.4528 radians with the air bladder fully inflated).

The next 5 **b_arg:** parameters define the glider's behavior after it surfaces:

- **b_arg: report_all(bool):**
- **b_arg: end_action(enum):** instructs the glider on what to do once the gps position is acquired. In this case, the glider is instructed to wait for the user to end the mission (via **^C**). Otherwise, resume the mission.

- **b_arg: gps_wait_time(sec)**: how long to wait to acquire a GPS fix, measured in seconds.
- **b_arg: keystroke_wait_time(sec):** how long to wait for a keystroke before resuming the mission, measured in seconds.
- **b_arg: printout_cycle_time(sec):** how often to print each surface dialog paragraph to the screen.

Surfacing When a Waypoint is Reached
This mission also specifies a surfacing behavior triggered when the glider reaches the current waypoint. The argument specifying when the behavior becomes active is:

b_arg: start_when(enum) 8

and the behavior arguments for this behavior are specified in the surfac30.ma (**b_arg: args_from_file(enum) 30**):

```
behavior_name=surface

# SURFAC30.MA (Waypoint Surface Radius)

<start:b_arg>
    # Surface Near Waypoint
        b_arg:  when_wpt_dist(m)        500

    # Flight Controls
        b arg:  c use bpump(enum)       2
        b_arg:  c_bpump_value(X)        1000
        b_arg:  c_use_pitch(enum)       3       # servo, rad, >0 = climb
        b_arg:  c_pitch_value(X)        0.4528  # 26 degrees

    # Surface Timeouts & Other Params
        b arg: report all(bool)             0       # F->just gps
        b_arg: end_action(enum)             1       # Surface menu
        b_arg: gps_wait_time(sec)           600     # GPS wait
        b_arg: keystroke_wait_time(sec)     300     # Surface time
        b arg: printout cycle time(sec)     40.0    # Surface menu print rate
<end:b_arg>
```

The only new **b_arg** in this file is the first one:

b_arg: when_wpt_dist(m) 500

which tells the glider that it must be a minimum of 500 meters from the current waypoint to satisfy the behavior and surface.

Surfacing After a Specified Interval of No Communication
This mission also specifies a surfacing behavior triggered if the glider has not had contact with shore for a specified interval. The argument specifying when the behavior becomes active is:

b_arg: start_when(enum) 9

which corresponds to exceeding the specified time interval contained in surfac40.ma:

```
behavior_name=surface

# SURFAC40.MA (No Comms)

<start:b_arg>
    # Surface for no comms (file transfer uncompleted)
        b_arg:  when_secs(sec)           5400

    # Flight Controls
        b arg:  c use bpump(enum)       2
        b_arg:  c_bpump_value(X)        1000
        b_arg:  c_use_pitch(enum)       3       # servo, rad, >0 = climb
        b_arg:  c_pitch_value(X)        0.4528  # 26 degrees

    # Surface Timeouts & Other Params
        b arg: report all(bool)             0       # F->just gps
        b_arg: end_action(enum)             1       # Surface menu
        b_arg: gps_wait_time(sec)           600     # GPS wait
        b_arg: keystroke_wait_time(sec)     300     # Surface time
        b arg: printout cycle time(sec)     40.0    # Surface menu print rate
<end:b_arg>
```

In this case:

b_arg: when_secs(sec) 5400

tells the glider to surface if it has not had a successful connection for 90 minutes. This behavior is typically used in conjunction with a timeout surfacing behavior in the following way: the timeout surfacing behavior (surfac20.ma) is used as the preferred surfacing method and, in this case, is set to 60 minutes (3600 seconds). If the glider surfaces but is not able to successfully connect to the dockserver, it will dive after 5 minutes (**b_arg: keystroke_wait_time(sec) 300**). 30 minutes later, however, the **no comms** behavior will be triggered and the glider will surface and attempt to connect again, rather than wait another 60 minutes for the **timeout** surfacing behavior to trigger.

behavior: goto_list
The **goto_list** behavior is used to specify the list of waypoint, which waypoint to head to first, and how many times to cycle through the waypoint list. As with the **surface** behaviors discussed above, the arguments defining this behavior are specified in a ma file as follows:

```
# Waypoint GOTO_LIST
behavior: goto_list
    b_arg: args_from_file(enum)     10  # mafiles/goto_l10.ma
    b_arg: start_when(enum)         0   # 0-immediately
```

which, in this case, is the **goto_l10.ma** file:

```
behavior_name=goto_list

# ENDURANCE LINE (OFFSHORE WAYPOINT FIRST)
# 1 km RADIUS
```

```
<start:b_arg>
    b_arg:  num_legs_to_run(nodim)  -1 # -1 loop,-2 run once,> 0 this many legs
    b_arg:  start_when(enum)        0   # 0 baw_immediately
    b_arg:  list_stop_when(enum)    7   # 7 baw_when_wpt_dist

    # SATISFYING RADIUS
    b_arg:  list_when_wpt_dist(m)    3000

    # LIST PARAMETERS
    b_arg:  initial_wpt(enum)  -1  # 0 to n-1,-1 first after last,-2 closest
    b_arg:  num_waypoints(nodim)    2 # num of waypoints in list
<end:b_arg>

<start:waypoints>
    -7414.000   3926.250
    -7252.500   3900.000
<end:waypoints>
```

The first 6 behavior arguments define how to treat the waypoint list:

- **b_arg: num_legs_to_run(nodim):** specifies how many times to traverse the waypoint list. A value of -1 runs it indefinitely, -2 runs through the list once and a value greater than 0 runs through the list that many times.
- **b_arg: start_when(enum):** specifies when to start the behavior, in this case, immediately.
- **b_arg: list_stop_when(enum):** only used if set to 7, this **b_arg** applies a stopping condition to all of the waypoints in the list. In this case, the stopping condition is specified with the next behavior argument.
- **b_arg: list_when_wpt_dist(m):** each of the waypoints in the list are achieved when the glider’s distance is less than 3000 meters.
- **b_arg: initial_waypoint(enum):** specifies which waypoint to head to first. A value of 0 to n-1 number of waypoints specifies that waypoint as the first in the list. A value of -1 selects the first waypoint in the list after the last one encountered and a value of -2 selects the closest. The setting of -2 needs some clarification, however. The closest waypoint is not actually selected. Instead, the midpoints between each successive pair of waypoints are calculated first. The waypoint selected is that which comes after the midpoint that is closest to the glider. This selection criteria can result in a puzzling waypoint selection, depending on the geometry of the waypoints desired.
- **b_arg: num_waypoints(enum):** a safety argument. This **b_arg** must specify the number of you wapoints in the specified list, otherwise the glider will abort the mission.

behavior: yo
The **yo** behavior is used to specify the diving and climbing behavior of the glider. As with the other behaviors discussed above, the arguments defining this behavior are specified in a ma file as follows:

```
# YO : FLIGHT SETTINGS
behavior: yo
    b_arg: args_from_file(enum)     10  # mafiles/yo10.ma
    b_arg: start_when(enum)         2   # 2-depth idle
    b_arg: end_action(enum)         2   # 2 resume
```

which, in this case, is the **yo10.ma** file:

```
behavior_name=yo
# 2009_05_05 dkaragon   shrunk size, pitch settings 5/5/09
<start:b_arg>

      b_arg: start_when(enum)              2   # pitch idle
      b_arg: num_half_cycles_to_do(nodim) -1   # infinite

      # DIVE #
      b_arg: d_target_depth(m)             98
      b_arg: d_target_altitude(m)          5
      b_arg: d_use_bpump(enum)             2   # absolute (cc)
      b_arg: d_bpump_value(X)             -1000.0
      b_arg: d_use_pitch(enum)             3   # servo
      b_arg: d_pitch_value(X)             -0.4537  # radians
      b_arg: d_stop_when_stalled_for(sec)  240
      b_arg: d_stop_when_hover_for(sec)    180

      # CLIMB #
      b_arg: c_target_depth(m)             5
      b_arg: c_target_altitude(m)         -1
      b_arg: c_use_bpump(enum)             2   # absolute (cc)
      b_arg: c_bpump_value(X)              1000.0
      b_arg: c_use_pitch(enum)             3 # servo
      b_arg: c_pitch_value(X)              0.4537 # radians
      b_arg: c_stop_when_stalled_for(sec)  240
      b_arg: c_stop_when_hover_for(sec)    180
      b_arg: end_action(enum)              2

<end:b_arg>
```

The first behavior argument (**b_arg: start_when(enum)**) instructs the glider to begin a new yo when the pitch is idle (ie: not being actively adjusted). The next behavior argument (**b_arg: num_half_cycles_to_do(nodim):)** specifies the number of **yos** (dive/climb cycles) to perform. Setting to -1 results in an infinite number of yos. This setting is typically used when surfacing behaviors triggered based on either a timeout or no comms are included in the mission. Setting this value to a number greater than 0 causes the glider to perform this many yos before satisfying the behavior.

The rest of the behavior arguments define the diving (**d_***) and climbing (**c_***) behavior of the glider:

- **b_arg: d_target_depth(m):** the depth to dive to, specified in meters.
- **b_arg: d_target_altitude(m):** while diving to **d_target_depth**, stay this far off of the bottom, specified in meters.
- **b_arg: d_use_bpump(enum):** use an absolute measurement for the buoyancy pump.
- **b_arg: d_bpump_value(X):** pull (negative value) the buoyancy pump piston this far backwards (large values are clipped), measured in cubic centimeters due to **b_arg: d_use_bpump(enum).**
- **b_arg: d_use_pitch(enum):** use the servomechanism to measure the pitch angle.
- **b_arg: d_pitch_value(X):** dive at a pitch angle of 0.4537 radians (~26 degrees).

- **b_arg: d_stop_when_stalled_for(sec):** keep diving until the glider determines it is no longer flying horizontally for this long, specified in seconds.
- **b_arg: d_stop_when_hover_for(sec):** keep diving until the glider determines it is no longer flying vertically for this long, specified in seconds.

All of the same behavior arguments are also used to define the climbing behavior; however the **d_** is replaced with a **c_** and the **c_bpump_value(X)** is positive, signaling that the pump should be pushed forward, and the **c_pitch_value(X)** value is positive.

behavior: sample
The **sample** behavior can be used to control either all of the scientific instrumentation as a unit or to customize sampling strategy on an instrument by instrument basis. In order to use individual **sample** behaviors to control each instrument, the following **sensor** must be set:

sensor: c_science_all_on_enabled 0

This **sensor** value can either be set in the glider's **autoexec.mi** configuration file or in the individual mission file (the preferred spot), in this case, **tn.mi**. As with the other behaviors discussed above, the arguments defining this behavior are specified in a ma file as follows:

```
# SAMPLE : CTD
behavior: sample
    b_arg: args_from_file(enum)     10  # mafiles/sample10.ma
```

Arguments controlling the sampling of the CTD are defined in the ***sample10.ma:***

```
behavior_name=sample

# SAMPLE10.MA (CTD)

<start:b_arg>
    b_arg: sensor_type(enum)                 1  # CTD
    b_arg: sample_time_after_state_change(s) 0  # start sampling right away

    # Sampling Arguments
    b_arg: intersample_time(sec)             4  # if < 0 then off
                                                # if = 0 then fast as posible,
                                                # >0 that number
    b_arg: state_to_sample(enum)         7  # 7  diving|hovering|climbing
<end:b_arg>
```

The behavior argument settings are as follows:
- **b_arg: sensor_type(enum):** an integer corresponding the the type of instrument this behavior is controlling. A full list of instrument type specifications is located in: ./src/code/sample.c.
- **b_arg: sample_time_after_state_change(s)**: how long after a state change to continue sampling.
- **b_arg: intersample_time(sec):** how often to sample, specified in seconds. A negative value turns the sensor off, 0 samples as fast as the instrument can and a value larger than zero samples that often.
- **b_arg: state_to_sample(enum):** a bit field used to describe which states to sample during. The glider has four states (specified with their

individual bit values): 1) diving, 2) hovering, 4) climbing and 8)on the surface. The value is set to 7, which instructs the CTD to sample during **dives, hovers and climbs** (1 + 2 + 4).

Using ma Files to Modify a Mission on the Fly

With a general understanding of the relationship between mission (.mi) and ma files (.ma), it's possible to modify deployment goals while minimally interfering with the glider. This is accomplished through the modification of ma files and re-compiling the mission/ma files from with a currently running mission.

For example, if the user wanted to extend the timeout surfacing behavior interval from 1 hour to 3 hours, they just need to create a new **surfac20.ma** file with the desired timeout (ie: 10800 seconds or 3 hours) and either place it on the glider prior to deployment (with different filename, then renaming it to **surfac20.ma**) or transfer it to the glider, from the dockserver terminal, using the following command:

!dockzr surfac20.ma

Once the **.ma** file has been received, the user must send a **^F** sequence from the dockserver terminal. This sequence results in a re-compiling of the mission and associated ma files. The new timeout then takes effect. Likewise, the list of target waypoints can be modified by creating a new **goto_l10.ma** file and performing the same sequence of events.

In comparison, a mission that is contained in a single **.mi** file can only be started after exiting the current mission and then restarting the new mission. This has a number of drawbacks: 1) The mission file is often larger (>5x the size) than even a few **.ma** files and 2) A lot of time is wasted on the surface, where the glider is subject to damage and unintended movement due to surface currents and wind fields.

This illustrates the flexibility provided by specifying mission parameters via ma files.

6. Glider Data Decoding

Slocum Glider Data Decoding

Modified from 'Slocum Glider Data Decoding' written by John Kerfoot (https://github.com/kerfoot/spt/wiki/Slocum-Glider-Data-File-Primer)

Introduction

This document presents a discussion of the Slocum glider native files formats and gives an example of the typical steps for renaming, decoding, and merging these files in preparation of downstream processing. While either of the formats may be used to create instances of the Dbd and DbdGroup classes, we recommend using the single dba format, as these files contain more extensive metadata records.

- **dba**: **dba** data files are ascii data files decoded from the original binary data file and contain a header with file metadata.
- **.m & .dat**: Pairs of segment data files used to facilitate the loading of the segment data using Matlab. The **.m** file is a script file used to load the data stored in the **.dat** sibling. These file pairs are missing much of the metadata found in the **dbd** header.

The Slocum glider is built with 2 processors. The first processor, referred to as the **glider (flight)** processor, is used for flight navigation and logging of all engineering sensor data. The second processor, referred to as the **science** processor, is used to control the science package instruments and log all scientific data sets.

The glider stores all sensor data in a set of 6 files encoded using the dinkum binary data format. This format consists of an ascii header followed by binary encoded data.

- **dbd**: **dbd** files contain the full-resolution sensor data gathered by the **flight** controller.
- **ebd**: **ebd** files contain the full-resolution sensor data gathered by the **science** controller.
  sbd: **sbd** files contain a subset of the **dbd** file contents, as configured by the sbdlist.dat.
- **tbd**: **tbd** files contains a subset of the **ebd** file contents, as configured by the tbdlist.dat.
  mbd: **mbd** files contain a subset of the **dbd** file contents, as configured by the mbdlist.dat.
- **nbd**: **nbd** files contain a subset of the **ebd** file contents, as configured by the nbdlist.dat.

Glider **missions** are composed of **segments**. The glider uses an **8.3 file** naming scheme to store data files.

Individual data **segments** are named according to the **mission** number (first 4 digits) and **segment** number (last 4 digits) of the **mission**, and are zero-based.

> For example, the following file: *04940000.sbd* is the first segment (0000) of the 495th (zero-based) mission.

The mission number ranges from 0000 – 9999 and is assigned on the day that the mission begins. This number is not incremented until a new mission is started. The segment number also ranges from 0000 – 9999 and is incremented each time a new segment begins.

Tools and executables are provided by TWRC for renaming, decoding, filtering, and merging binary data files to their ascii equivalents:

datahost.webbresearch.com   Glider/production/windoze-bin

1. **rename_dbd_files**: renames to 8.3 filename to it's more informative format
2. **dbd2asc**: decodes the 8.3 binary filename (requires the correct ASCII sensor list header) to it's ascii equivalent. Result is printed to STDOUT.
3. **dba_sensor_filter**: Takes the output of **dbd2asc** and filters out any sensors not contained in the specified sensor list. The sensor list is a list of whitespace delimited sensors to include.
4. **dba_merge**: Merges 2 segment file pairs (ie: **dbd** & **ebd** or **sbd** & **tbd**) by timestamp (**m_present_time** and **sci_m_present_time**).
5. **dba2_orig_matlab**: Takes the output of either **dbd2asc** and **dbd_sensor_filter** and writes a pair of files for importing into the Matlab programming environment.

Example of Data Decoding:

- Using a command line terminal, commands you need to type are in blue below:
- We have 2 files, 00500000.EBD and 00500000.DBD.

1. Rename the 8.3 binary files are renamed to their full filename equivalents:

```
Rename_dbd_files 00500000.[DE]BD
```

```
ru28-2013-197-5-0.dbd
ru28-2013-197-5-0.edb
```

The file is renamed according to the following conventions:

ru28: name of the glider
2013: year the mission was started
197: julian day (zero-based) on which the mission was started
5: sequential mission number (zero-based) started on day 197
0: sequential segment number (zero-based) of the current mission number

2. Binary to ASCII Decoding

Decode each binary file to ASCII and, if desired, filter out unwanted sensors. For dbd & ebd pairs, sensor filtering is often used as the majority of sensors are of no scientific analysis. This must be done on a per-file basis, and is done differently depending on whether the full file ascii header, used in decoding, is contained in the file.

The presence or absence of this header is controlled by:

u_dbd_sensor_list_xmit_control (dbd files) and u_sci_dbd_sensor_list_xmit_control

a. When the full ascii header is contained in the files:

```
Dbd2asc ru28-2013-197-5-0.dbd > ru28-2013-197-5-0.dba
```

An error will be returned if the full ASCII header was not contained in the file:

```
Dbd2asc ru28-2013-197-5-0.dbd
Error, ignoring: file:ru28-2013-197-5-0.dbd Can’t open cache file
./cache/868d75a7.cac
Nothing to process!
```

b. In this case, you can use the **-c** option with **dbd2asc** and specify the location of the header file:

```
Dbd2asc –c /path/glider_cache_folder_location ru28-2013-197-5-0.dbd > ru28-2013-
197-5-0.dba
```

If the full header is in the original binary file, use of this option creates and write the header, for future use, in the specified directory.

The resulting ASCII file in a. and b., **ru28-2013-197-5-0.dba**, contains all of the glider controller sensors.

c. We can remove the majority of the useless (at least for scientific analysis purposes) sensors by piping the output of **dbd2asc** to **dba_sensor_filter**, along with the list of sensors to include, before redirecting to the filename.

```
Dbd2asc –c /path/glider_cache_folder_location ru28-2013-197-5-0.dbd |
dba_sensor_filter –f /path/standard_sensors.txt > ru28-2013-197-5-0.dba
```

where path/standard_sensors.txt contains the following (example) list of sensors (comments begin with #):

```
# Always include this!
m_present_time

# Tells us if the glider thinks it's at the surface
m_appear_to_be_at_surface

# Steering parameters
m_fin
c_fin
m_pitch
c_pitch
m_roll
c_roll
m_heading
c_heading

# Glider health
m_coulomb_amphr
m_battery
m_vacuum

# Iridium parameters
m_iridium_call_num
m_iridium_signal_strength
m_iridium_redials

# Depth parameters
m_depth
m_pressure
m_altitude
m_water_depth

# Depth-averaged currents
m_final_water_vx
m_final_water_vy
m_water_vx
m_water_vy

# GPS sensors
c_wpt_lat
c_wpt_lon
m_gps_lat
m_gps_lon
m_lat
m_lon
m_gps_status
m_gps_full_status

# Science sensors
sci_ctd41cp_timestamp
sci_m_present_time
m_science_clothesline_lag

# CTD sensors
sci_water_pressure
sci_water_cond
sci_water_temp

# GliderDOS version
x_software_ver
```

This results in a file containing all of the sensors listed above (non-existent sensors are Ignored).

Convert the corresponding ebd file in the same way:

```
Dbd2asc –c /path/glider_cache_folder_location ru28-2013-197-5-0.ebd |
dba_sensor_filter –f /path/standard_sensors.txt > ru28-2013-197-5-0.eba
```

3. Merge Glider and Science File Pairs

Now merge the **dba** & **eba** pair into a single ASCII file using dba_merge.

```
Dba_merge ru28-2013-197-5-0.dba ru28-2013-197-5-0.eba > ru28-2013-197-5-0.deba
```

This file is one of the 2 file types accepted by the Dbd or DbdGroup class.

```
Dbd = Dbd(‘ru28-2013-197-5-0.deba’)
```

Create a Matlab file pair using dba2_orig_mat

```
Dba_merge ru28-2013-197-5-0.dba ru28-2013-197-5-0.eba > | dba2_orig_matlab ru28-2013-
197-5-0_sf_dbd.m
```

2 files are produced: **ru28_2013_197_5_0_sf_dbd.m** and **ru28_2013_197_5_0_sf_dbd.dat**. This is the other file type accepted by the Dbd or DbdGroup class (but remember to specify the .m file, not the .dat sibling)

We recommend using the **ru28-2013-197-5-0.deba file when creating instances of the **Dbd** or **DbdGroup** classes, not the Matlab "equivalents" as the **ru28-2013-197-5-0.deba** contains the default masterdata sensor units, while the Matlab files do not contain these units, so a default of **nodim** is used.

** John Kerfoot, Rutgers, has put together a Matlab toolbox to do some of these routines called Slocum Power Tools, available on GitHub. More information at https://github.com/kerfoot/spt/wiki.

7. G3s Glider Processor

Beginning in 2020, new slocum gliders transitioned from the Persistor-based processor found in G1, G2, and early G3 gliders to a new non-persistor based processor (STM32).

One main difference between the persistor and non-persistor gliders is that ‘PicoDos’ is now called ‘GliderShell’.

Some of the common commands and features have also changed. This table shows the key differences. (Table from TWR)

Persistor	STM32
exit pico	exit shell
boot pico	boot shell
app	app
boot app	boot app (*will not boot the app, only sets the glider to boot app)
consci	consci
exit	exit
send/s/zs *.*	send/s/zs *.* (just * will also work)
burnapp	flash-flight or flash-science
u4stalk	uart (now uses the actual comms port ID from the science bay motherboard, i.e. J0, J1, J2)
talk	talk (*some device names have changed)
date	date (time command not supported)
adtest	adtest
\ (backslash used to separate directories in file paths)	/ (forward-slash used to separate directories in file paths)