Seaglider Science Controller

Integrative Observational Platforms Group
Applied Physics Laboratory
University of Washington

Revised: June 2024

Contents

Chapter 1

Introduction

This document is an introduction to the use of the Seaglider science controller (scicon). Scicon is an electronic and software subsystem on Seaglider vehicles that allows for independent (of Seaglider flight operations), asynchronous sampling and onboard processing of science sensors. This guide assumes that the user is familiar with general glider piloting, basestation and software menu operations.

In the Seaglider software architecture, scicon is an autonomous logger (as opposed to a sensor). This means that the Seaglider provides power and the flight software communicates with scicon via a series of action messages at specific points of flight operations (dive start, apogee reached, dive finished, pre-Iridium call, post-Iridium call, depth changed, etc.), but otherwise the logger device operates independently of the glider flight control. During flight, scicon samples sensors according to its own configuration files (the glider *science* file has no effect on scicon operations). When the dive is complete, the glider queries scicon for all data collected during the dive and then passes those files to the basestation directly. Data collected by scicon is not reported in the regular glider log and engineering files. Basestation software is responsible for merging glider data and scicon data into the resultant netcdf product.

Chapter 2

Hardware and wiring

Scicon consists of a small ARM-based single-board computer usually mounted on the VBD cylinder in the aft of the vehicle. This arrangement provides the easiest wiring access to the bulkhead connectors on the aft endcap for sensors mounted in or on the aft fairing.

Scicon provides connections for five serial sensors (four RS232, one logic level) and two frequency output sensors (e.g., SBE CT, SBE 43). The logic level serial port and one frequency channel are shared. The layout of channel connectors on the board is shown in Fig. 2.1. Scicon is connected to the glider via the console serial port. To configure scicon on the glider, configure a logger device using the `param/config/logger` menu option. Scicon is usually connected to one of the standard glider serial ports USART2, USART4, USART5B or 5C.

Generally, sensors are connected either to scicon or to the glider "truck" platform. If they are connected to scicon, then sampling is done via scicon and control is via the scheme file, *scicon.sch* (below). If connected to the glider directly then sampling is controlled via the *science* file. With the latest generation of electronics, specifically the Rev E "combi" tailboards, it is possible to have one serial sensor and one frequency sensor (e.g., the SBE CT sail) electrically connected to both scicon and the glider truck. In this arrangement, the configuration can be changed mid-mission. At any one time, only one platform (glider truck or scicon) can be configured to sample a given sensor. Simultaneous access is not allowed. On the glider the shared ports are the CT frequency port and the 5D serial port. An example of typical aft endcap wiring is shown in Fig. 2.2.

It is possible, though not common, to have some sensors connected to the glider (to be controlled via *science* file and reported in the engineering file) and some sensors connected to scicon.

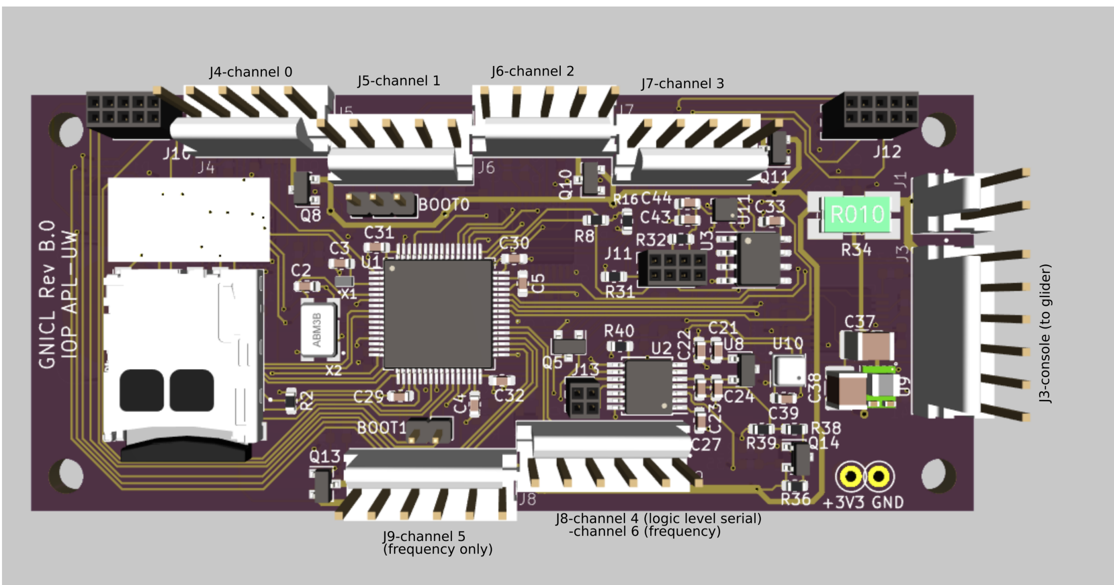

Figure 2.1: Science controller board layout with connector numbering and channel labels.

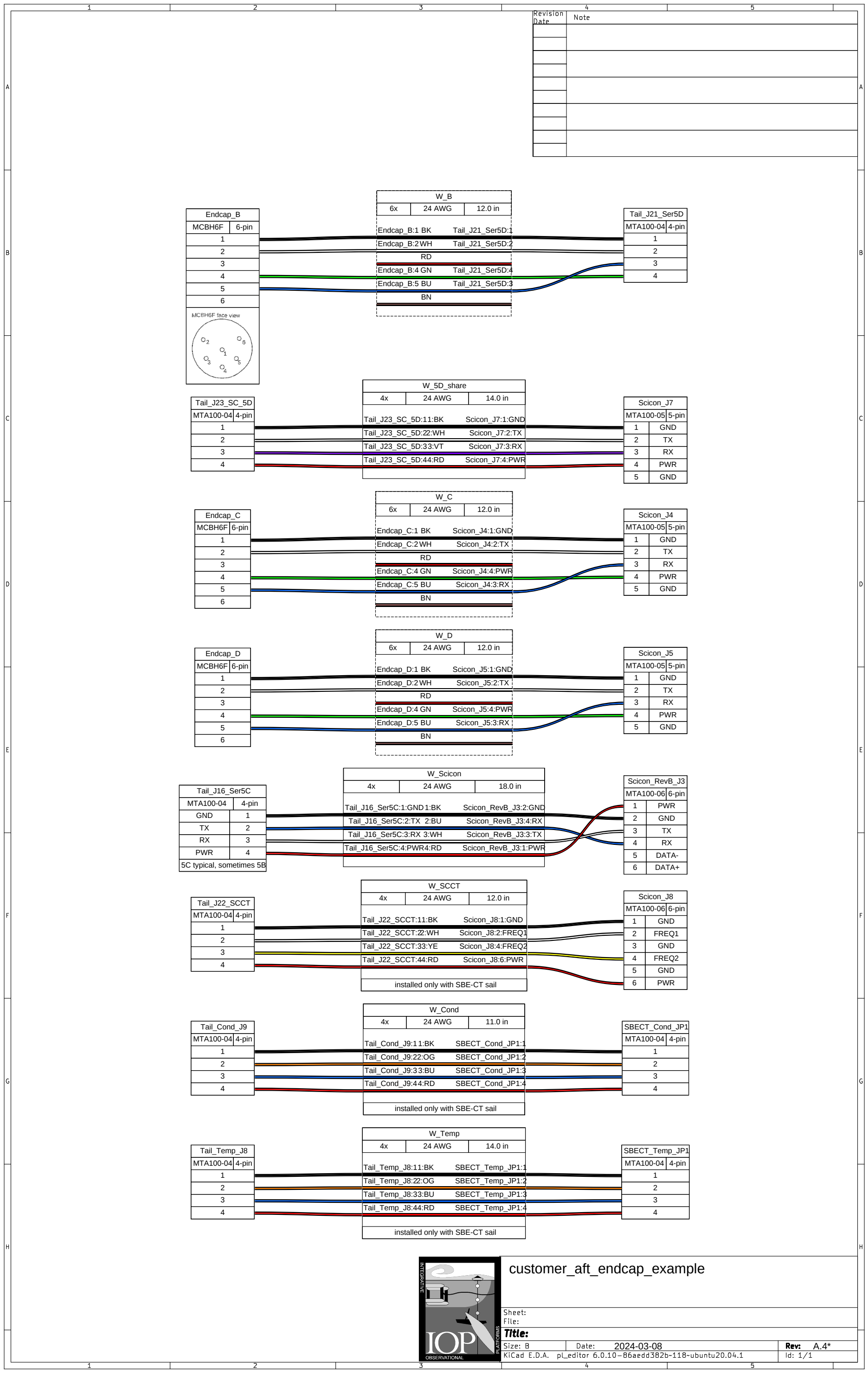

Figure 2.2: Example wiring for an aft endcap with scicon installed. Bulkhead assignments, tailboard USART assignments, and scicon channel assignments are somewhat arbitrary. Generally, wiring can be re-arranged as mission needs dictate.

Chapter 3

Configuration

Scicon operations are controlled by three configuration files: *scicon.ins* (describes instrument definitions), *scicon.att* (how sensors are attached), and *scicon.sch* (sampling scheme for the intervals, depth bins, dives and profiles of each attached sensor). Generally, *scicon.ins* and *scicon.att* do not change over the course of a mission, but they can be changed if needed.

The scicon configuration files are automatic, that is, they are handled automatically by the glider if they are present in the glider home directory on the basestation, similar to *targets*, *science*, etc. If present during an Iridium communication session, the glider will download the file to its own local file storage, and then after the call is complete will copy the files over to scicon's own filesystem. They are read by scicon at every power-up during the execution of the script file *startup.scr* so any changed configurations will take effect at the next operation.

Comments in all three files are denoted with a `#` as the first character on a line. Inline comments are not supported.

3.1 scicon.ins

The instrument properties file defines the classes of instruments that can be sampled, one definition per instrument type. This class definitions contains information such as baud rate, warmup times, timeouts, query strings, how data is parsed, and the types of data returned.

3.1.1 example

```
sbect = {
    prefix = ct
    cycles = 255
    warmup = 700
    timeout = 2000
    column = condFreq(1000,0)
    column = tempFreq(1000,0)
}
```

```
legatoPoll = {
   prefix = fb
   baud = 19200
   cycles = 0
   timeout = 1000
   warmup = 8000
   skip = 0
   terminator = 10
   query = %F%n%3%F%n%[Ready: ]fetch sleepafter=true%r%n
   format = %d-%d-%d %d:%d:%f, %00, %01, %02, %03
   meta = %F%n%1%F%n%[Ready: ]getall%r%n%[Ready:]
   start = %F%n%1%F%n%[Ready: ]disable%r%n%[Ready: ]
   stop = %F%n%1%F%n%[Ready: ]%9%9powerexternal used%r%n%[%n]
   column = conduc(10000,0)
   column = temp(10000,0)
   column = pressure(1000,0)
   column = conducTemp(10000,0)
}
aa4831 = {
   prefix = aa
   format = %s %d %00 %01 %02 %03 %04
   baud = 9600
   warmup = 1000
   terminator = 10
   timeout = 5000
   skip = 0
   column = O2(1000,0)
   column = airsat(1000,0)
   column = temp(1000,0)
   column = calphase(1000,0)
   column = tcphase(1000,0)
   meta = %[4831]%n%$11%9get all%r%n%[Reference]%[%$23]
}
```

3.1.2 syntax

```
instrumentClass = {
    prefix=     two character prefix for file and directory names
    driver=     standard | ad2cp | json - specifies which parser to use for
                handling data. standard is default and uses the format=
                definition to parse data. ad2cp handles averaging and
                decimation for Nortek AD2CP binary format. A single coeff=
                must be specified in the instance to set the expected
                record size (for example, (86 + 4x1xN_burst_cells)
                + (86 + 4x3xN_velocity_cells)).
```

```
    baud=        baud rate for serial communications (e.g., 19200)
    cycles=      number of cycles to measure for frequency channel
    timeout=     timeout in milliseconds
    warmup=      warmup time in milliseconds
    status=      status expression, evaluated and returned on "log sample"
    skip=        number of initial lines to skip after power-up before sampling
    terminator= terminating character of data output line (decimal byte value)
    query=       query string to send for each sample (can be empty)
    format=      format string for parsing serial data returned
    meta=        command string to retrieve instrument metadata
    start=       command string start sampling
    stop=        command string to stop sampling
    column=      name(scale,offset)
    column=      name(scale,offset)
    ...          additional columns
    conf=        configuration string to send before starting
    conf=        configuration string to send before starting
    ...          additional configuration strings
    post=        RBR-TS | TS - apply post-stop function (e.g., bin averaging)
                 If TS is specified for T-S profiles derived from SBE CT
                 then 8 coeff= values (C_G...C_J, T_G...T_J) must be provided
                 in the instrument instance
    comment=     comment string
    comment=     comment string
    ...          additional comment strings
}
```

`prefix` and one of `baud` or `cycles` are required. `format` and `terminator` are required for serial instruments if using the `standard` driver (which is the default if not otherwise specified). At least one `column` must be specified. All other values default to zero or empty.

3.1.2.1 format statements

For sensors that return data as a single line of ASCII serial text, the format= statement is used to define how that line will be parsed into individual columns. It can be any string consisting of regular text (that will be matched exactly) and the special characters:

```
  %00, %01, %02, ... = read a number from the response as channel 0 (1, 2, ...)
  %[n]00             = read n digits from the response as channel 0 (1, 2, ...)
  %f                 = read and skip over a floating point number
  %d                 = read and skip over an integer number
  %s                 = read and skip over string data up to a white space
```

Columns can be parsed in any order in the format statement, i.e., they do not need to appear in numerical order. If channel 00 is “temperature” and channel 01 is “salinity” based on the order of `column` statements and the instrument output format is

```
mm/dd/yyyy HH:MM:SS salinity voltage temperature
```

then the format might be one of

```
format=%s %s %01 %f %00
format=%d/%d/%d %d:%d:%f %01 %f %00
```

3.1.2.2 columns

There must be one `column= statement` in the definition for each datatype returned by the instrument. If columns %00 ... %05 are referenced in the format statement then there must be six column statements to define the name and packing information for those variables.

```
column = name(scale,offset) - scale and offset are applied for conversion
                              to long integers for the differenced tabular
                              ASCII data file representation
```

Columns must be listed in order (00, 01, 02...). The first column listed will define channel 00, the second channel 01, etc. Scale and offset values determine how scicon will pack the parsed data into files for transmission. Data are sent as long integers. If the data from the sensor is reported as integers then scale can be 1. Floating point values are generally scaled up by the order of magnitude that preserves the appropriate number of decimal places when the data is reconstructed on the basestation.

3.1.2.3 string commands

Commands that are sent to the instrument (start, query, meta, stop, conf) can contain interpolated text. Interpolated text are special strings, similar to print/printf format specifiers in some programming languages, which are evaluated each time the string is constructed and sent to the instrument.

```
%1 ... %9 = DelayMilliSecs(10) .. DelayMilliSecs(90)
%b = serial break
%r = CR (13)
%n = NL (10)
%e = esc (27)
%F = flush the receive queue
%$xx = send hex byte xx - must include both characters (00 - ff)
%[] = wait for an arbitrary sequence of characters. Characters between the
      brackets can include any ASCII character. To wait for % use %%. To wait
      for ] use %]. Use %r and %n to wait for CR and NL, respectively.
      Use %$xx to wait for an arbitrary hex byte. Several commands use the
      string gathered while waiting for this sequence as output. For example,
      the final captured output of the stop= command is reported on the finish
      line of the data file trailer.
%(n) = wait for an arbitrary number of milliseconds (n  = 1...99999)
%<> = interpret characters as a strftime specification for current time
%% = %
```

3.2 scicon.att

The attach file defines the specific instruments that are currently configured and how they are connected to scicon. There is one entry per instance of an instrument type, defining which class of instrument (this must be a type defined in *scicon.ins*), which hardware channel it is attached to, any instance specific configuration data (calibration coefficients for example), and directives to average or decimate samples, and drop or derive new columns for files that are telemetered. Raw data (all columns at the original sample rate) are always stored on the scicon filesystem.

3.2.1 example

```
ct = {
    type = legatoPoll
    hwchan = 2
}
optode = {
    type = aa4831
    hwchan = 1
}
ad2cp = {
    type = ad2cp
    hwchan = 3
    drop = attitude
    avg = 4
}
```

3.2.2 syntax

```
instrumentInstance = {
    type=       instrumentClass
    hwchan=     hardware channel number (0-4 are serial channels, 5-6 are freq)
    share=      another instrumentInstance that serves as the "parent"
                in lieu of hwchan when multiple classes (with different
                format statements) come from the same physical hardware
    avg=        number of samples to average into telemetered file
    dec=        decimation factor (integer) for samples into telemered file
    verbose=    verbosity level for including raw data into telemetered file
                bit 0: print raw line on short scan (default = off)
                bit 1: include short scan diagnostics (default = on)
                bit 2: always include raw data line (default = off)
    drop=       name of column to drop from telemetered file
    drop=       name of column to drop from telemetered file
    ...         additional dropped columns
    new=        columnName(scale,offset):expression
```

```
    new=        columnName(scale,offset):expression
    ...         additional derived columns
    conf=       configuration string to send before startup
    conf=       configuration string to send before startup
    ...         additional configuration strings
                configuration strings are cumulative. Strings specified
                here in the instance are sent after strings specified
                in the class.
    coeff=      calibration coefficient
    coeff=      calibration coefficient
    ...         additional calibration coefficients. Different drivers may
                use coeff values differently and expect specific numbers
                of values and that they be specified in a strict order.
    comment=    comment string
    comment=    comment string
    ...         additional comment strings
    convert=    columnName(slope,offset) - apply slope and offset to columnName
    convert=    columnName(slope,offset) - apply slope and offset to columnName
    ...         additional conversions
    binavg=     bin size (meters) for bin averaged profile products
}
```

`type=` and `hwchan=` are required and are often the only properties provided in a typical installation. Any class property (e.g., warmup, timeout) may also be specified to override the value in a given instance.

3.2.2.1 expressions

Columns derived with `new=` must include the mathematical expression used to calculate the derived value. Components of expressions are:

```
Variables referencing the latest data from any instrument:
    instanceName.columnName

Operators:
    standard operators: +,-,*,/,%,()
    binary operators:   <<,>>,|,&,~
    logical operators:  >,<,&&,||,==,!=,<=,>=,!
    if-then-else operator: a ? b : c

Math functions:
    sin, cos, tan, sinh, cosh, tanh
    pow, exp, log, log10,
    sqrt, hypot, ceil, fmod,
    fabs
```

```
Symbolic constants:
    pi

oceanographic functions:
    salinity(C, T, P) (psu)
    potentemp(S, T, P, RefP) (degC)
    soundspeed(S, T, P) (m/s)
    density(C, T, P) (kg/m^3)
    potendens(C, T, P, RefP) (kg/m^3)

    C=mMho/cm, T=degC, P=dbar, S=psu

Other functions:
    distance(lat0, lon0, lat1, lon1) (meters)
    time() (RTC time since epoch)
    epoch(yyyy, mm, dd, HH, MM, SS) (converts calendar time to epoch time)
```

For example, to reduce payload size, a CTD instrument could be configured to report density only, dropping the original temperature and salinity (conductivity) channels with the following in the instance definitions in the attach file:

```
drop = temperature
drop = conductivity
new = density(1000,0):potendens(ct.conduc, ct.temp, ct.pressure, 0)
```

3.3 scicon.sch

The scheme file defines how each instrument is sampled. Sampling intervals can be controlled as a function of depth bins, profile (dive, climb, loiter), and dive number. Per scheme configuration information can also be specified (for example to change ADCP parameters on dive and climb). An attached instrument can have multiple schemes defined. The active scheme will be chosen in order of decreasing specificity of profile and dive definitions.

3.3.1 example

```
ct = {
   50, 4.0
   200, 7.5
   1000, 14.0
}
optode = {
   dive = 2
   200, 5
   500, 12
   1000, 45
```

```
}
ad2cp = {
   profile = a
   conf = SETAVG,CH=124%r%n%[OK]
   conf = SAVE,ALL%r%n[OK]
   1000,15.000000
}
ad2cp = {
   profile = b
   conf = SETAVG,CH=234%r%n%[OK]
   conf = SAVE,ALL%r%n[OK]
   1000,15.000000
}
```

3.3.2 syntax

```
instrumentInstance = {
    profile=        which profile (a | b | c) this scheme applies to,
                    where a = dive, b = climb, c = loiter.
                    If unspecified this scheme becomes the default scheme
                    for this instance.
    dive=           dive modulo for applying this scheme. Default=1 (every).
    conf=           configuration string to send before startup
    conf=           configuration string to send before startup
    ...             additional configuration strings
                    Configuration strings are cumulative, in addition
                    to and sent after strings specified in the class
                    and instance.
    coeff=          override coeff value during this scheme
    coeff=          override coeff value during this scheme
    ...             additional coefficient values
                    Coefficient values specified here _replace_
                    values specified in the instance.
    avg=            Number of samples to average into telemetered file during this
                    scheme. Overrides any value given in the attach file.
    dec=            Decimation factor (integer) for samples into telemered file
                    during this scheme. Overrides any value in the attach file.
    depth, rate     deepest depth (meters) and sample rate (seconds)
    depth, rate     deepest depth (meters) and sample rate (seconds)
    ...             additional sampling bins
}
```

At least one `depth, rate` pair must be specified. Other values are optional. A value of zero for rate turns the instrument off in any bin. To turn an instrument completely off use `2000,0` or remove the instrument from the attach file.

Depth bins are specified as in the glider *science* file, in order of increasing depth. The specified depth values indicate the bottom of the bin. The top of the bin is implied by the depth of the preceding bin, with an implied zero at the beginning of the list. Sampling rates for all instruments are independent of each other.

3.4 Parameters

In addition to the configuration files there are five glider-level control parameters to affect the gross behavior of scicon. These are:

`$LOGGERS`	Bitmask for global on/off control. Set bit number `n` corresponding to the `$LOGGERDEVICEn` to turn scicon on (bit=1) or off (bit=0). If scicon is in slot 1 (`$LOGGERDEVICE1`) include the value 1 in `$LOGGERS` to turn scicon on. Remove 1 from `$LOGGERS` to turn scicon off.
`$SC_PROFILE`	Bitmask controlling which profiles to run scicon. 1=dive, 2=climb, 4=loiter. Set to 3 for example to run during dive and climb, but not during loiters.
`$SC_XMITPROFILE`	Bitmask controlling which profiles to run transmit to basestation. 1=dive, 2=climb, 4=loiter. Usually matches `$SC_PROFILE`, but can be different to conserve bandwidth, but collect data to be recovered with the glider.
`$SC_RECORDABOVE`	Depth limit above which scicon will be run. Usually set to some depth deeper than the deepest apogee depth, but can be shallower if all instruments will be off below a certain depth to conserve additional power.
`$SC_NDIVE`	Modulus controlling which dives to run scicon. Usually set to 1 to run scicon every dive. A value of 2 will run scicon every other dive, 3 every third dive, etc.

Chapter 4

Glider menu controls

With scicon configured on the glider, the `hw/loggers/sc` menu will be available. The following options (with optional arguments) are available:

```
------ SciCon ------
     1 [on      ] Turn on controller
     2 [off     ] Turn off controller
     3 [selftest] Selftest
     4 [sample  ] Report a sample
     5 [readclk ] Read controller clock
     6 [syncclk ] Synchronize controller clock to TT8
     7 [command ] Execute controller command
            string="command string to send"
            string="command string to send"
            ...
     8 [get     ] Get file from from controller
            name=filename to retrieve from scicon
     9 [put     ] Put file onto controller
            name=filename to send to scicon
    10 [firmware] Put firmware onto controller
            name=filename to send
    11 [action  ] Execute logger action
            action=number (see below)
    12 [edit    ] Edit configuration
            param=value
            param=value
            ...
    13 [config  ] Show configurable params
    14 [direct  ] Direct comms
            file=script
            string="string to send"
            string="string to send"
```

```
          binary=0|1 (default 0)
          addlf=0|1 (default 0)
          stroke=0|1 (default 0)
    15 [capture ] Capture comms
          file=script
          string="string to send"
          string="string to send"
          seconds=timeout
    16 [loader  ] Bootloader access
          timeout=seconds
    17 [stream  ] Stream sensor data
          sensor=name
    18 [ct      ] CT sensor data
```

These options can be used to verify basic scicon and sensor functionality after maintenance or assembly. They are also helpful when developing and debugging new sensor integrations. Note that the `selftest` option is not the same as the sequence of tests performed during the autonomous pre-launch (whole glider) selftest. Use `action action=16` to reproduce those tests.

4.0.1 action numbers

The Seaglider operating software communicates with logger devices through a series of action messages at various points of the operational cycles. For testing purposes, these messages can be sent through the `action` menu option with the argument `action=N` where N is one of the numbers from table 4.1

phase	**number**	**notes**
OFF	1	sent to force off
DIVE_START	2	
APOGEE_REACHED	3	
DIVE_FINISHED	4	
SYNC_CLOCK	5	clock synced after GPS1
SYNC_PPS	6	clock synced to PPS after GPS 1
SYNC_2_CLOCK	7	clock synced after GPS2
SYNC_2_PPS	8	clock synced after GPS2
XMIT_READY	9	before Iridium call - gathers collected data
XMIT_COMPLETE	10	after Iridium call - transfers control files
RECOVERY	11	every recovery cycle
PREDIVELOG	12	not used by scicon
POSTDIVELOG	13	collect data for log file
BOOT	14	
LAUNCH	15	cleans up filesystem (deletes old data)
SELFTEST	16	
PRE_SELFTEST	17	not used by scicon
SAMPLE_DEPTH	18	new depth data available, report CT data if available
SAMPLE_STATUS	19	check for change of profile or depth based controls
DIAGNOSE_POWER	20	
SEALEVEL	21	latch sealevel value for pressure sensors
MOTORS_ON	22	not used by scicon
MOTORS_OFF	23	not used by scicon
SURFACE	24	

Table 4.1: Available action points and their assigned number for messages that can be sent from Seaglider to scicon.

Chapter 5

Selftest results

The autonomous selftest function of the glider menu (`launch/autotest`) runs a series of diagnostic commands on scicon, results of which are reported in the selftest capture file. Many of these report on the basic functionality of the scicon hardware (real-time clock, oscillator speeds, fuel gauge configuration, firmware version). Selftest results also include the current instrument, attachment, and scheme files as parsed by scicon, a data sample from each instrument, e.g.,

```
--checking ct legatoPoll
ct: -0.001 21.928 9.876 21.945
avg mA=10.17, J=2.5
--checking wl wlbb2flvmt
wl: 54.000 60.000 70.000 537.000
avg mA=45.69, J=1.4
--checking optode aa4831
optode: 256.757 93.949 21.939 28.929 30.992
avg mA=40.53, J=0.7
```

and metadata from each instrument (if `meta=` is defined in *scicon.ins*). These latter two are particularly important to review to verify correct instrument operation and configuration.

These test results can be viewed either by direct reading of the selftest capture file (*ptGGGNNNN.cap*) or using either the command line (selftest.sh) or visualization server (/selftest/GGG) based selftest viewers that are part of the basestation. The latter provide more formatting and error highlighting for easier readability.

Chapter 6

File formats and basestation handling

Scicon creates a directory for every profile, with separate files for each sensor inside the directory. Directory names are in the form *scNNNNp* where NNNN is the dive number and p is the profile indicator (a=dive, b=climb). Loiter data is included as part of the climb. At the end of each dive, the glider transfers a gzipped tarball of each of these directories from scicon to its own filesystem and then during the Iridium call uploads these files to the basestation.

The format of individual data files within the directories is similar to the glider's own engineering file. Header and footer lines are marked with %. Data lines are represented as the sample-to-sample difference written in plan ASCII text. The first column is always the elapsed time in milliseconds from the start of sampling. Filenames are "instanceName.dat" for each instrument within the directory. If there is any data reduction or manipulation specified (averaging, decimation, dropping or adding of columns) then the raw data is written into "instanceName.dao" and the changed data for telemetry written into the .dat file.

6.0.1 .dat example

As an example, this is the beginning and ending of a file from a legatoPoll CTD attached with the instance name "ct" for a dive profile. The header lines indicate that the pressure sensor latched 10240 as the pressure at launch when the glider sent the "log sealevel" command, that this file belongs in the directory sc0030a (which gives both dive number and dive direction), column names and how they are scaled, and that sampling for this file started at 06-June-2023 at 16:39:58.346 UT.

```
% instrument: ct legatoPoll
% sealevel: 10240
% columns: elapsed_t(1,0) conduc(10000,0) temp(10000,0) pressure(1000,0) conducTemp(1000
% container: sc0030a
% comment: SG236
% start: 6 6 123 16 39 58 346
9367 381421 136429 10308 130109
```

```
1129 414 -107 197 0
5012 40 167 88 87
4994 57 43 137 130
...
20000 7 2 275 27
20000 -41 -49 289 -27
% stop: 6 6 123 19 46 35.437
% finish: powerexternal used = 4.135e+000
% samples: 830
% ontime: 5803704
```

Trailer lines indicate how many samples were recorded during this profile, the total powered on time in milliseconds, and for this instrument (because there is a stop command defined in the .ins file), the results of that command in the “finish” line. For a Legato CTD, the “powerexternal used” command returns the energy used in Joules.

Scicon does real-time fuel gauging at the whole board level, not per instrument. The glider uses the fuel gauge results in its power modeling for battery consumption, but there is no per instrument breakdown. The average current consumption over the entire dive is reported in `$SENSOR_MAMPS` in the log file. Per instrument power usage can be deduced by looking at the samples count and ontime in the trailers of individual files as above along with knowledge of the energy per sample of a given instrument. Some instruments (Legato CTD, Nortek AD2CP) can provide their own usage numbers (either measured or modeled). Selftest results include the results of the “log test” command which include average current and Joules used during the test. This can be a good indicator of per sample current consumption, but be aware that these single sample numbers may not be representative for instruments with a long warmup-time, that stay on between samples, or with a shutdown procedure, as with a a typically configured Legato CTD for example.

6.0.2 .eng example

After uploading and unpacking the tarballs, the basestation converts the .dat files into .eng files named pscGGGNNNN_class_instance.eng where GGG is the glider number, NNNN is dive number, p is profile (a or b) and class and instance are as defined in the .ins file (class) and .att file (instance). Header and trailer lines are copied through. Data lines are summed to remove differencing and have scale and offset applied. Column 1 is now absolute epoch time in seconds.

```
%instrument: ct legatoPoll
%sealevel: 10240
%columns: legatoPoll.time legatoPoll.conduc legatoPoll.temp legatoPoll.pressure legatoPo
%container: sc0030a
%comment: SG236
%start: 6 6 123 16 39 58 346
1686069607.713 38.142 13.643 0.068 13.011
1686069608.842 38.184 13.632 0.265 13.011
```

```
1686069613.854 38.188 13.649 0.353 13.020
1686069618.848 38.193 13.653 0.490 13.033
...
1686080757.271 32.197 3.621 994.448 3.747
1686080777.271 32.193 3.616 994.737 3.744
%stop: 6 6 123 19 46 35.437
%finish: powerexternal used = 4.135e+000
%samples: 830
%ontime: 5803704
%timeouts: 0
%errors: 0
```

Chapter 7

Firmware commands

Beyond use as the glider science controller, the scicon board is a powerful general purpose data logger. When in direct communications with scicon or via the "command" menu options, the following commands are available at the scicon > command prompt. Only a small subset of these are used by the glider during normal operations.

`attach [filename]`	show/read attachments
`scheme [filename]`	show/read sample scheme file
`prop [filename]`	show/read property file
`detach instrument`	remove instrument
`log start DEPTH dirname [comment]`	start logging
`log stop`	stop logging
`log finish`	stop loggging and tar files
`log inst`	report data from all attached - single line
`log fetch`	report data from all attached - verbose
`log test`	report data and power from all attached
`log sample NAME1 NAME2 ... [DEPTH]`	report latest from NAMEs
`log count NAME`	report count from NAME
`log power NAME`	report power draw from NAME
`log meta`	report metadata from attached
`log scheme PROFILE(a|b|c) DIVE`	change scheme
`log sealevel`	record sealevel from pressure sensors
`log single NAME`	sample only NAME while logging
`log all`	restore sampling of all attached
`script scriptfilename`	execute commands in script
`source scriptfilename`	execute commands in script without complaint if missing
`oneshot scriptfilename`	exec then delete script
`pdos YES`	reboot with suspended WD

`quit`	reboot

`time command args`	measure execution time of command
`repeat N command args`	loop repeat command N times
`print string1 string2...[>|» dest]`	echo string(s) to console or file

`ver [long] [set]`	program version information

`sysclk`	show clock speed
`sysclk N [hse]`	set system clock speed
`hscal [N]`	calibrate system clock
`rtcclk`	show RTC config

`baud N`	set console baud rate
`echo on|off`	turn console echo on or off

`fgwd calibrate`	calibrate FG slope/offset
`fgwd wd`	show wd value
`fgwd wd N`	set wd value to N
`fgwd volts`	read battery voltage
`fgwd current`	read instantaneous current draw
`fgwd read`	read accumulators
`fgwd clock`	reconfigure wd clocks
`fgwd save`	save wd value to NVRAM
`fgwd clear`	clears accumulators
`fgwd temp`	read MSP430 internal temp
`fgwd vlo`	calibrate vlo
`fgwd fire`	fire wd (reset)
`fgwd offset OFFSET SLOPE`	set FG slope and offset
`fgwd zero`	zero FG offset
`fgwd config`	read configuration
`fgwd ad`	read AD rate register
`fgwd ad N`	write N to AD rate register
`fgwd register X n`	read n-byte value at reg X
`fgwd write X n`	write n-byte value at reg X
`fgwd reboots`	report watchdog reboot count

`batt`	read RTC batt voltage

`vee clear`	re-init VEE header
`vee clear N`	re-init end pointer to N
`vee format`	zero out VEE
`vee read string|float|long|byte var`	read var from VEE
`vee write string|float|long|byte var value`	write var to VEE

<table>
<tr><td><code>flash status</code></td><td>read protection status</td></tr>
<tr><td><code>flash protect on|off iap|app</code></td><td>change protection status</td></tr>
<tr><td><code>flash boot iap.bin</code></td><td>flash write IAP region</td></tr>
</table>

<table>
<tr><td><code>xr filename</code></td><td>receive file via xmodem</td></tr>
<tr><td><code>xs filename [N]</code></td><td>xmodem send file</td></tr>
<tr><td><code>cat filespec [>|» dest]</code></td><td>cat file(s) to console or file</td></tr>
<tr><td><code>rr filename</code></td><td>raw receive file</td></tr>
<tr><td><code>rs filename</code></td><td>raw send file</td></tr>
<tr><td><code>firmware filename md5sig</code></td><td>xr receive file for firmware</td></tr>
<tr><td><code>ys filespec [path]</code></td><td>ymodem send file(s)</td></tr>
</table>

<table>
<tr><td><code>service N</code></td><td>service logging loop for N sec</td></tr>
</table>

<table>
<tr><td><code>compass init [calfile]</code></td><td>init auxp compass</td></tr>
<tr><td><code>compass coeff</code></td><td>report compass coefficients</td></tr>
<tr><td><code>compass raw</code></td><td>stream raw compass results</td></tr>
<tr><td><code>compass cal</code></td><td>stream cal compass results</td></tr>
<tr><td><code>compass calibrate</code></td><td>collect attitude cal data</td></tr>
</table>

<table>
<tr><td><code>dir</code></td><td>root directory contents</td></tr>
<tr><td><code>dir filespec</code></td><td>dir with globbing</td></tr>
<tr><td><code>dir filespec dirname</code></td><td>dir in dirname with globbing</td></tr>
<tr><td><code>dirfile capname</code></td><td>root dir saved to capname</td></tr>
<tr><td><code>dirfile capname filespec</code></td><td>glob results saved to capname</td></tr>
<tr><td><code>dirfile capname filespec dirname</code></td><td>glob results to capname</td></tr>
<tr><td><code>del filespec</code></td><td>delete in root dir (recurses)</td></tr>
<tr><td><code>del filespec dirname</code></td><td>delete in dir</td></tr>
<tr><td><code>gzip filename gzipname</code></td><td>compress file</td></tr>
<tr><td><code>gunzip gzipname filename</code></td><td>uncompress file</td></tr>
<tr><td><code>md5 filename</code></td><td>compute md5 checksum</td></tr>
<tr><td><code>cp src dest [size] [start]</code></td><td>copy file</td></tr>
<tr><td><code>ren src dest</code></td><td>rename file</td></tr>
<tr><td><code>mkdir dirname</code></td><td>make directory</td></tr>
<tr><td><code>filesdirs</code></td><td>report file and dir count</td></tr>
<tr><td><code>trunc filespec len</code></td><td>truncate file to len bytes</td></tr>
<tr><td><code>tar t|c|x[vz] tarname filespec dir</code></td><td>create tarball</td></tr>
<tr><td><code>vi filename</code></td><td>edit filename with vi</td></tr>
</table>

<table>
<tr><td><code>stream instrument</code></td><td>stream data direct from inst</td></tr>
<tr><td><code>term N [addlf] [stroke] [script file]</code></td><td>direct terminal port N</td></tr>
<tr><td><code>power N 0|1</code></td><td>power control port N</td></tr>
<tr><td><code>power</code></td><td>report power state of all ports</td></tr>
<tr><td><code>gpio read Xnn</code></td><td>read state gpio port X, pin nn</td></tr>
<tr><td><code>gpio set Xnn</code></td><td>set bit gpio port X, pin nn</td></tr>
</table>

`gpio reset Xnn`	clear bit gpio port X, pin nn
`freq N [1|2] [cyc] [samples]`	frequency count port N
`cardinfo [long]`	report SD card info
`mem`	report malloc stats
`sleep N`	low-power sleep N seconds
`sleep_ms N`	low-power sleep N millisec
`delay N`	busy wait N millisec
`clock`	read RTC
`clock HH:MM:SS`	set RTC time
`clock MM/DD/YYYY HH:MM:SS`	set RTC date and time
`epoch`	report epoch time
`epoch N`	set epoch time
`launch`	pre-mission setup and house cleaning