ESP

MONTEREY BAY AQUARIUM RESEARCH INSTITUTE

Environmental Sample Processor Mission Scripting

5/22/14 Brent Roman   brent@mbari.org

MBARI

Mission Scripts and Phases

- Top Level Commands for a deployment
  - Often omitted for lab work
- Usually contains a mission method
  - Specifies the starting tube number
  - Optionally specifies Mission End Time
  - Contains any number of mission phases
    - Each having a start time
      - with optional trigger conditons
    - One or more protocols run per phase
  - The ESP sleeps between phases
    - Contextual sensors continue being polled

Protocols

- Protocol scripts do the real work of microbiological assays
  - Many canned scripts available:
    - HAB = Harmful Agal Bloom
    - BAC = Bacterial Assay
    - LARV = Larval Assay
    - WCR = Whole Cell Archival
    - DA = Domoic Acid Assay
    - HABDA = combined HAB and DA assay
    - STX = Saxitoxin Assay
  - All have parameters you may modify to suit your needs
    - With default values so you needn't specify everything
  - You may also create new protocols using the existing protocols as a guide:
    - STX was created just last year as a variant of DA

Example “3peat” QC Mission

```
mission  startTube: 2,  until:  “6AM 12/18/12”  do

  at "12:40:00 12/14/12" do
   habda {noKill}
  end

  at "12:40:00 12/15/12" do
   habda {noKill}
  end

  at "04:00:00 12/17/12" do
   habda
  end

end
```

It's Ruby all the way down

- Commands, Missions, Scripts, Protocols, Configuration Files
  - All are written in version 1.8 of the Ruby scripting language
- *Learn a little Ruby*
  * Rote memorization fails when something goes wrong
  - Standard on Mac OS, easily installed everywhere else.
- A gentle tutorial:
  - https://pine.fm/LearnToProgram/
- The bible:
  - http://pragprog.com/book/ruby/programming-ruby
- More (free) choices to suit your learning style:
  - http://ruby.about.com/od/tutorialsontheweb/tp/10waysfree.htm

ESP

MONTEREY BAY AQUARIUM RESEARCH INSTITUTE

Environmental Sample Processor
Contextual Sensors

5/22/14 Brent Roman brent@mbari.org

MBARI

Supported Instruments

- Can => internal environmental sensors within ESP core's housing
  - Temperature, humidity, pressure, battery voltage, amperage
  - Updates every 10 minutes as long as ESP application runs
- CTD => Seabird SBE 16plus V2 interfaced via RS-232 sensor 1
  - Temperature, pressure, conductivity, plus *optional*...
  - Fluorometer, Transmissometer, Oxygen Sensor (1 of 2 types)
- ISUS => one of two types interfaced via RS-232 sensor 2
  - Concentration of nitrate and, optionally, bisulfide
  - Support for all manufactured at MBARI
    - Some later models from Satlantic (in use at WHOI)
- TBD = Something new can yet be interfaced as RS-232 sensor 3
  - Note: this port is not currently wired to lid of the can

Polling Contextual Sensors

- Trickier than it would first seem
  - ISUS must synchronize with CTD to receive timely updates
  - Sample rate optionally quickens during sampling
  - Multiple threads may not access instruments simultaneously
  - The Can's internal sensor polling is controlled independently
- ESP explicitly triggers every CTD sample!

- Code is in Polling object in mission/skeleton.rb
  - Polling.start #starts SensorPolling with new parameters
  - Polling.stop #stops polling and properly closes instrument files
  - Polling.pause #stops until resumed
  - Polling.resume #resumes previous polling schedule if paused

- Instrument shows last sampled state of all Instruments
  - CTD, ISUS, Can show last sampled state of each Instrument

Internal Environmental sensors

- can is short for Sleepy.queryCan --> forces immediate sampling
  - can.temperature => internal temp. at top of can in degrees C
  - can.humidity => humidity in % of saturation
  - can.pressure => internal pressure in psia
  - can.voltage => instantaneous battery voltage
  - can.current => instantaneous battery load in amps
  - can.avgCurrent => averaged battery load in amps
  - can.waterAlarm => percent “wet” (0..100) usually < 1
  - Wattage is merely can.current * can.voltage

- Sleepy.can accesses most recent sample
  - Typically updated every 10 minutes
  - Recorded in binary 'real.log' file

Seabird CTD

- Seabird 16plus V2 CTD with
  - support for fluorometer, transmissometer, oxygen sensor, ...
  - Generates file CTD-*.hex of raw samples
- CTD.status   # shows instrument status
- CTD.pumpmode = *mode*, where *mode* is either:
  - :off, :beforeSample, or :duringSample
- s = CTD.sample => returns sample object, assigns it to variable s
  - s.temperature => sea temperature in degrees C
  - s.conductivity => conductivity in S/m
  - s.pressure => pressure in decibars
  - s.transmissometer => % optical transmission
  - s.beamAttenuation => extinction coefficient in 1/m
  - s.sampleTime => time at which this sample was started
  - s.dataTime => time at which this sample was finished
  - s.depth => depth in meters (derived from pressure)
  - s.salinity => salinity in mythical PSUs
- More documentation in lib/instrument/ctd.rb

ISUS

- ISUS = In-Situ Ultraviolet Spectrometer
  - Stores raw spectra in ISUS-*.dat (MBARI's ISUS only!)
  - Logs errors in ISUS-*.err
  - ***Requires temp., salinity & depth from the CTD !!***
- ISUS.status   # shows instrument status
- ISUS.species = 2 (or 3)   #three to include bisufide
- ISUS.fit = 217..240 #spectral fit window in nm (tweak for species)
- ISUS.fromCTD temp, salinity, depth #update ISUS from CTD
- s=ISUS.sample => sample with most recent values fromCTD
  - s.no3 => Nitrate concentration in uM/L
  - s.br => Bromide in uM/L
  - s.hs => Bisulfide in uM/L (only valid if species>2 and fit tweaked)
  - s.sampleTime => when sample was requested
  - s.dataTime => when sample was recorded
- More documentation in lib/instrument/isus.rb

Parameters controlling Contextual Sensor Polling

- *$global* variables determine instruments' configuration/polling rates
- These may be assigned anytime before Polling.start
  - But, usually they get set once in mission/phasecfg.rb
  - Missions with :until=>*time* automatically invoke Polling.start
- CTD
  - $ctdPumpMode=:duringSample #may be :beforeSample or :off
  - $ctdInterval=Delay.new “5:00” #sample CTD every 5 minutes
  - $ctdPeriod=Delay.new “1:00:00” #upload CTD data every hour
  - $samplingCTDinterval=Delay.new “2:30” #2x faster ...
  - $samplingCTDperiod=Delay.new “30:00” # while sampling
- ISUS
  - $isusSpecies = 2 #ignore sulfides by default (3 to include them)
  - $isusFit =217.240 #because Luke says it should be so :-)
- ISUS polling rate is CTD sampling rate + 10 minutes
  - ISUS auto-sampling cannot be disabled

ESP

MONTEREY BAY AQUARIUM RESEARCH INSTITUTE

Adaptive Sampling With Trigger Conditions

5/22/14 Brent Roman brent@mbari.org

MBARI

Traditional ESP Missions

- A sequence of “phases”, each with a prescribed start time
  - Actions predetermined by puck load

- ESP sleeps between phases. While “asleep”:
  - Still monitors contextual sensors
  - Still maintains radio context with shore

- All phases began at times prescribed in the mission script
  - Start times specified may be absolute or relative
    - Relative times specify the “sleep time” between phases

- No adaptive sampling was possible without hand coding it

Trigger Condition Overview

- Each start time is augmented by a list of trigger conditions
  - A phase starts when any of its trigger conditions is true
  - The start time can be thought of as the one required trigger condition
    - It determines the latest possible starting time for the phase
    - Triggers start phases before their scheduled times
      - Triggers cannot delay phases beyond their “start times”
      - Triggers ***cannot*** change the sequence of actions performed
        - » *Processing sequence is determined by puck load.*
- Each trigger condition is reevaluated whenever contextual sensors read
  - Sensible, as trigger conditions almost always evaluate sensor data
  - This is a convention
    (but, not difficult to circumvent if necessary)
- Each trigger condition runs in its own Ruby thread
  - Failure (e.g. exceptions raised) in any trigger will not affect the others
    - You can even patch the code and restart failed trigger conditions
    - Or, kill the trigger thread to ensure it does not trigger the phase

Basic Trigger Conditions

- Basic Trigger Conditions contain arbitrary true/false expressions
  - A threshold value is associated with each
    - CTD.temp < threshold
    - ISUS.no3 > threshold
    - CTD.depth > threshold[0] and CTD.fluor > threshold[1]
  - Thresholds need not be scalar values
  - Trigger expressions are reevaluated just after each time contextual sensors are read while the mission is awaiting conditions
- May be assigned names like Cold, Hot, Fresh, Salty
- Threshold values can be modified at any time
  - Via the script itself or the interactively via espclient
  - All modifications to thresholds are logged
- Very flexible, but also painfully verbose for complex triggers

Trigger Thresholds

- Each trigger optionally has an associated threshold value
  - Usually used to parametrize conditional expressions
    - But you may choose to compare to constants instead
  - Need not be scalar, only the expression interprets it
  - Not usually applicable to box or range conditions
    - Such thresholds would be vectors of ranges if used
- If your conditional expressions reference a threshold:
  - You must set it before the trigger is used
    - Cold.threshold = 4.3   #it's that easy!
  - The default threshold value is nil
    - CTD.fluor > nil #will generate an exception!

Composite Trigger Conditions

- Two types
  - Trigger “all” means when all subordinate conditions are true
    - Trigger all: [Cold, DCM, HighNitrate]
    - Equivalent to: Cold[] and DCM[] and HighNitrate[]
    - Trigger all: []
      - is always true
  - Trigger “any” means when any subordinate condition is true
    - Trigger any: [Cold, DCM, HighNitrate]
    - Equivalent to: Cold[] or DCM[] or HighNitrate[]
    - Trigger any: []
      - is always false
- All subordinate conditions run in the same thread as the parent

Trigger Box Conditions

- True if each listed measurement is within *the same* associated box of interest
  - Represented as the same Ruby hash mapping used for Trigger Ranges
  - Trigger box:
    ```
    {CTD%:temperature      => [-3.3..2.1, 5..7.21],
     CTD%:salinity          => [ 33..33.4, 35..35.5]}
    ```
  - Read the boxes off the columns of the resulting matrix.
  - If temperature is in one column and salinity is in the other, the trigger condition is *false*
- Columns geometrically define a set of boxes in the space of sensor measurements

Trigger Box Corner Cases

- If measurements do not specify the same number of ranges:
  - Those that are missing ranges will be ignored

```
Trigger  box:
{CTD%:temperature => [-3.3..2.1, 5..7.21],
 CTD%:salinity      => [ 33..33.4 ]}
```

  - If the temperature is between 5..7.21, the trigger condition is true, regardless of salinity

- If a measurement specifies a single range (not an Array)
  - That range will be applied to all others
  - As though it had been repeated in an Array

```
Trigger  box:
{CTD%:temperature => [-3.3..2.1, 5..7.21],
 CTD%:salinity      => 33..33.4}
```

  - The salinity must always be in 33..33.4, regardless of temperature

Trigger.now!

- Not really a trigger condition, rather an action!
  - Causes the current mission phase to start immediately
  - Raises an exception if mission is not waiting
    - Exception is raised in caller's thread
    - The mission's processing is unaffected

- There need not be any trigger conditions associated with the waiting phase for Trigger.now! to work.
  - The phase may be just awaiting its start time

Trigger.replace or Trigger.restart

- Replace current phase's start time and/or trigger conditions
  - Affects only for the phase currently waiting to start
  - Raises an exception if mission is not waiting
- All arguments are optional
- First argument is the replacement phase start time
  - Specify nil to leave start time unchanged
- Other arguments are replacement trigger conditions
  - Omit other args to leave existing triggers in place
- Trigger.replace “+1.5 days”, Cold, Deep
  - Mission will continue waiting up to 36 more hours for the Cold ***or*** Deep condition to be satisfied

Trigger Holdoffs

- Trigger holdoffs are a simple way to avoid false triggers
  - A form of glitch filtering
  - ESP logs show countdown when awaiting holdoffs
- All triggers have an associated holdoff in samples
  - condition must be true for at least holdoff+1 samples
  - nil is the default holdoff value
    - holdoff=nil, equivalent of holdoff=0
      - But holdoff nil is not displayed, whereas 0 is
  - holdoff of false disables that particular trigger condition
  - holdoff of true forces trigger on its next evaluation

Trigger enable and disable

- Enable trigger monitoring with:
  - Trigger.enable
- Disable trigger monitoring with:
  - Trigger.disable
- Trigger monitoring is initially disabled
  - Use Trigger.enable as soon as contextual data starts making sense and all relevant thresholds are defined
- Triggers may be enabled/disabled at any time
  - Even while awaiting them
- Triggers are initially enabled during simulation!

Automatic Trigger Rearm

- Trigger monitoring may be disabled whenever a trigger condition causes a phase to start
  - If triggers remain enabled, rearm is said to be true
  - If triggers disable once one has fired, rearm is said to be false
- Set the rearm flag with:
  - Trigger.rearm = true
- Clear the rearm flag with:
  - Trigger.rearm = false
- Real missions start with rearm=false
  - You may change the Trigger.rearm flag at any time
  - You may want to combine it with Trigger.enable or Trigger.disable
- Simulation missions start with Trigger.rearm=true