ESP
MONTEREY BAY AQUARIUM RESEARCH INSTITUTE

The ESP Server
Main Application

5/17/22 Brent Roman   brent@mbari.org

ESP Server Startup

- Determines what Ruby code files to read
  - from ESP environment variables
- Reads state files to recall any machine state

  that cannot be directly sensed
  - e.g. puck placement
- Establishes comms with ESP gateway
- Does NOT power on other microcontrollers
  - until `ESP.configure`
- Does NOT change position of any actuators
  - until `ESP.ready!`

ESP Server

- Start with command:

```
esp   # interactive mode (attached to terminal)
esp mission   #run mission script (attached)
start esp mission   #run mission detached
start esp      #wait for espclient(s) to connect
```

  - Only “start esp ...” produces the *mode*.out log file
- ESP env variables must be initialized beforehand
  - per “Setting up the ESP Environment” slide
- Beware that network failures will cause crashes
  - if esp app is attached to a controlling terminal
  - use “start esp ...” to avoid this!

ESP Operating Modes

- Determine how time advances
  - Real-Time vs Simulated Time
  - Can’t access hardware in simulated time
- Determine what gets displayed on terminal
  - quiet or quick modes display little
  - normal modes display most useful events
  - debug modes display everything
- The binary log stores all events
  - regardless of operating mode!

Real-Time Operating Modes

- ESPmode=real
  - Normal operation in real-time with real hardware
  - Default mode when running on ESP hardware
- ESPmode=debug
  - copious output displayed
- ESPmode=brief
  - less then usual output displayed
- ESPmode=quiet
  - only errors displayed
- ESPmode=nolog
  - nothing displayed
- ESPmode=simreal
  - simulated hardware with normal output
- ESPmode=simdebug
  - simulated hardware with copious output

Simulated Time Operating Modes

- ESPmode=simfast
  - accelerated time, normal output
- ESPmode=simfaster
  - simfast for “long” mission mode
- ESPmode=simfastdebug
  - simfast with copious output
- ESPmode=quick
  - simfast with less than usual output
  - best for simulating missions before deployment
- ESPmode=quicker
  - quick mode for “long” missions
- ESPmode=simrapid
  - special mode for 3G ESP
  - accelerate sampling simulation

More About Operating Modes

- All modes are defined as Ruby files in the mode subdirectory
  - One may easily create their own custom modes.
  - Mode definition files are named:
    - $ESPhome/mode/*mode_name*.rb

Required ESP Environment Variables

- ESPmode=real #operating mode
- ESPhome=/home/esp/esp2 #top dir of ESP app
- ESPpath=/home/esp/esp2/mission:.

  #where to search for ESP mission scripts

- ESPconfigPath= #path to config files
- ESPlog=/var/log/esp #where to write files
- ESPname=bruce #name of ESP machine
- RUBYLIB=/home/esp/esp2/lib:/home/esp/esp2/utils
- PATH=*...:/opt/mbari/bin:$ESPhome/bin*

ESPenv script

- Sets required ESP environment variables
  - Must be ‘sourced’ [e.g. run with dot, as “. ESPenv”]
  - because it modifies the current shell’s environment!
  - Run automatically on login as part of shell startup
- All script’s parameters are optional

```
# 1st parameter is the platform type (eg. [shallow], 1km)
# 2nd parameter is unit name (eg. gigi, neo, etc.) = ESPname
#   The default for name derived from the system's hostname
# 3rd parameter is the ESPhome directory [~/esp2] = ESPhome
# 4th parameter is ESPpath
#   defaults to [$ESPhome/mission:.]
# 5th parameter is ESPconfigPath
#   defaults to
#[$ESPhome/type/$type/$unitName:$ESPhome/type/$type:
$ESPhome/type:$ESPhome/admin]
```

Optional
ESP Environment Variables

- `TZ=US/Pacific` #overrides time zone
- `ESPcheckpoints=0` #disables Thread.resume
- `ESPcmdPort=9999` #listen on TCP espclient port
- `ESPclient=host:8888` #connect to host on port
- `ESPaxisPort=3333` #listen on axis display port
- `ESPforget=true` #do not restore puck state
- `ESPmodules=/home/esp/esp2/lib/analytic`

  #path to drivers for analytical modules

Why Simulate?

- Simulate missions before deployment to catch
  - Syntax errors
  - Missing, wrong, or extra parameters
  - Configuration errors
    - Trying to pull a reagent that is not configured/defined
    - Insufficient volumes of reagent(s)
    - Waste container overflow
  - Scheduling errors
    - Not leaving enough time between mission phases
    - Scheduling recovery before last phase completes
- Simulate adaptive sampling triggers
  - With recorded or generated CTD data
  - Observe when sampling occurs
    - Adjust trigger conditions as needed
- Run simulations on ESP itself, or on a Linux desktop/laptop

Setting up to Simulate

- Real ESP’s automatically configure their env vars
  - Laptops and Desktops simulating ESPs do not
- Must set required env vars before simulating
  - or using ‘dumplog’ to display the binary log
- Typically all that is needed to simulate 2G ESPs is:

```
$  .  ESPenv  shallow  ESPname
```

  - where *ESPname* = name of the ESP to simulate
  - example of setting up to simulate ESPchris:

```
.  ESPenv  shallow  chris
ESPmode=simfast     #for simulated time
```

Simulated Time

- starts at 1/1/1970 UTC [i.e. the Unix Epoch]
- does not advance when idle
- advances instantly to any future time

  `-> delayUntil Time.now` #advances to now

- is restored when ESPserver restarts
- cannot reverse
  - to reset time to 1/1/1970, use:

    `$ forgetESPstate` #before starting esp

- `-> delay 600` #advances 10min instantly
- `-> sleep 600` #takes 10min, does not advance
  - (do not use sleep)

Time.now vs Thread.time

- Time.now = the current clock-on-the-wall time
- Thread.time = when all threads were last idle
  - In real-time modes
    - Thread.time always slightly before Time.now
    - Thread.time advances when ESP idle
  - In simulated time modes
    - Thread.time has no relation to Time.now
    - Thread.time advances only when ESP delays

Multithreading and Thread.time

- Computation delays do not advance Thread.time
- All threads must advance Thread.time in lock step
  - Thread.time advances only when all threads are idle
  - otherwise, it will be inconsistent between them
- These rules are required to ensure that:
  - processor speed does not impact results
  - simulated time results are the same as real-time
- However, threads can “unsync” from Thread.time
  - to allow it to advance while they remain “busy”
  - Useful for I/O
    - and simulating with multiple espclients

Testing ESP Operating Mode

- `ESP::Mode` = current operating mode (`$ESPmode`)
- `ESP::Home` = ESP install directory (`$ESPhome`)
- `ESP::ConfigPath` = configuration directories (`$ESPconfig`)
- `ESP::LogDir` = log directory (`$ESPlog`)
- `ESP::LogFn` = file path to binary log
- `ESP::MinVoltage` = minimum operating voltage

- `ESP.simulation?` true if this is a simulation
- `Thread.realtime?` true ESP running in real-time

Simulation Procedure

- ESPmode must be set before starting the ESP software
- Change the mode for all subsequent runs with:

  ESPmode=*newMode*

- Restore normal mode for all subsequent runs with:

  ESPmode=real

- Change mode w/o affecting subsequent runs with:

  ESPmode=*newMode* esp   *mission*

  - Omit *mission* to simulate interactively

- Most typical simulation command:

  ESPmode=quick esp *myNewMission*

Simulation Features and Limits

- Protocols are simulated in full detail
  - Every movement of the physical hardware is simulated
  - Every I2C message is simulated down to the byte level
  - Puck handling assumes that there are no stack height errors
  - Will not detect mechanical interference between axes
    - E.g. attempts to move the carousel with the Elevator up will ***succeed*** in sim
    - But, attempts to move the Elevator past its physical limits will ***fail*** in sim
  - One should test new protocols by simulating them first, before wasting reagents.
- Does simulate CTD
  - But not ISUS
- Tracks consumption of Time
  - Does not simulate energy consumption
  - Will track fluid and reagent use on 2G ESP
    - if script begins with: require ‘fluid’
- Simulation of whole missions is CPU intensive
  - Allow 90 minutes to simulate a full mission on the slow ESP processor
  - The same sim would take < 30 seconds on a fast server.
  - Figure on it taking 90 seconds for the typical laptop

Declaring Puck Stack Heights

- Puck stack height cannot be measured in simulation
  - Puck load must be prescribed in simulations
- Every new mission should define the number of pucks expected to be loaded in each tube!
  - Optional in “real” mode, but...
    - Isn't it better to fail early, if puck load is wrong?
- Excerpt of mission with 6 pucks in tubes 2, 3 and 4:

```
pucks 2=>6, 3=>6, 4=>6    # see next slides

mission startTube:  2, until:  “9AM  4/10/15”
 do

  <mission phases>

end
```

- Fails immediately if tube 2 did not start with exactly six pucks

Declaring Puck Stack Heights

- Commands to set and query the expected stack height:
  - `clear!  tubeList=1..7`
    - Clears each specified tube's stack height
  - `fill!  numPucks=22, tubeList=2..6`
    - Puts the specified number of pucks in each listed tube
  - `pucks  tubeHash={}`
    - Puts the specified number of pucks in specified tubes: eg. `pucks  3=>14, 7=>8`
    - If tubeHash omitted, just displays the # of pucks in each tube

Stack Height Setting Examples

```
-> fill! if ESP.simulation?
```

- Fills all tubes except #1 (for typical fully loaded carousel mission)
- But, only if running as a simulation

```
-> fill!;  clear!  2, 4..7
```

- Ends up with tube 3 containing 22 pucks, others empty

```
-> fill!  9
```

- Fills all tubes except #1 with 9 pucks (empties others)

```
-> fill!  9, 1, 3..5, 7
```

- Fills tube 1, 3, 4, 5 and 7 with 9 pucks (empties others)

```
-> pucks  2=>22, 6=>18
```

- Fills tube #2 with 22 pucks, tube #6 with only 18

```
-> pucks
```

- Changes nothing
- Just returns the hash of pucks in tubes.

Running Multiple ESPservers

- At most one ESP server may be run
  - in any mode that accesses real actuators

    ```
    Errno::EBUSY in MAIN -- Device or resource
    busy - /dev/I2Cgate -- Missing core Gateway!
    ```

  - in the same simulation ESPmode

    ```
    Log::Locked in trapHandler -- Another
    process is already writing to logFile
    ```

- One may run a simulation along side another in different ESPmode.
  - or while the ESP is running in `ESPmode=real`