Data Reporter Sketch Notes:

The DataReporter application consists of a set of tasks and supporting source files. The tasks making up the DataReporter are:

* Monitoring the rainfall.
* Monitor ambient temperature.
* Manage Li-ion battery charging.
* Minimize the power requirement.
* Report data to a cloud based database via GSM modem.

Tasks are mechanized using state machines driven by an Executive Task Scheduler in the Arduino loop() function. Each task performs some essential process of the DataReporter application such as:

* Monitor
  + Sleep/Wakeup.
  + Rain gauge.
  + Monitor battery.
* LogData
  + Time tag data.
  + Record data points.
* ReportData
  + Send data to cloud database.
* ManageResources
  + Queue resource requests.
* LogMessages

**DataReporter Sketch Source Files:**

DataReporter\_REV\_1\_1.ino

Has two functions.

void setup()

Enter:

Debugging.

Any time power is applied.

Only a single call to the SetupTask()

All hardware and software setup is done in the SetupTask().

void loop()

Contains only the scheduler.

Definitions.h

Application wide global definitions only.

Global.ino

All global variables.

All enums.

All structure definitions.

Monitor.ino

This is the main DataReporter task.

Scheduled at all times.

Discovers and reports all datapoints.

Sleeps the processor at all possible times.

Setup.ino

All required hardware and software initialization is done in this task.

LogData.ino

Logs data points to an SD file.

Determines when to report data points to the database.

ReportData.ino

Reports the data points in the SD file to the database.

ResourceQueues.ino

Rtc.ino

SystemLogTask.ino

**Adding/Scheduling a Task**

A task is a state machine mechanized with a switch() statement that in turn is driven by the Executive Task Scheduler in the Arduino loop() function. A template of a task source file is shown in TaskTemplate.ino Every task is required to have an identifying 2 or 3 character prefix for all local definitions and local (.ino file) globals.

Add a task as follows:

* Add a MyNewTask.ino source file(Tab).
* Add a MyNewTask.ino file using TaskTemplate.ino template.
* Replace TEMPLATE\_TASK with YOUR\_TASK.
* Replace TaskTemTask with YourTask.
* Add an index that identifies the new task added i.e.
* #define YOUR\_TASK n in Definitions.h
* Initialize the new task entries in setup();
  + taskPointer[YOUR\_TASK] = NewTask;
  + taskScheduled[YOUR\_TASK] = false;
  + tasksState[YOUR\_TASK] = TASK\_INIT\_STATE;
* Add and code the states, for the new task, as required.

Schedule a task as follows:

* Check taskScheduled[YOUR\_TASK] with interrupts off.
* Set the entry state newTaskState[YOUR\_TASK].
* Set taskScheduled[YOUR\_TASK] true.
* Task is now running.

To end the task:

* Set taskScheduled[YOUR\_TASK] false.

**Shared resources:**

Shared resources, e.g. SD, system log messages, log data point, etc. are controlled and communicated with using FIFO queues. Each queue, e.g. “Xyz” has the following global variables:

* XyzQueueType XyzQueue[MAX\_XYZ\_QUEUE\_ITEMS];
  + Where XyzQueueType is the control structure used to control and communicate with the Xyz resource.
* int myRequestIndex XyzPush(XyzQueueType myRequestStructure);
* Copies the caller’s structure onto the Xyz’s FIFO queue.
* Schedules the Xyz’s task if it not currently scheduled.

XyzPop();

Removes the structure at the top of the queue.

XyzQueueCount

The number of items in the queue.

the top of the FIFO queue.

XyzQueueOutIdx XyzQueueInIdx

The index of the item at

The index of the next queue position available for queue insertion.

To use the the shared resource:

1. Declare the control structure for the resource.
2. Fill out the resource control structure.
3. Turn the interrupts off.
4. Call the push method for the resource, with a pointer to your control structure, and get an integer which indexes to your structure in the resource's queue.
5. Turn the interrupts on.
6. Monitor the status in the control structure to see if the resource has completed.
7. Process the results in the control structure.

**System Log**

The system log is intended to allow tasks to record diagnostic or error messages to the Syslog.txt

file on the SD card. The error/diagnostic messages, that are stored as character arrays in program memory, are prepended with a time stamp, up to three parameters appended and the composed message is written to the Syslog.txt file.

To create a system message:

* In the SystemLogTask.ino source file:
  + Add the char array, containing the text of the message at the end of the MESSAGE\_TABLE. Note that the char array uses “\_n” to insure uniqueness nd must end with a zero byte (“\0” ).
  + Add a pointer to the message, that you added above, as the next iten in the sysLogPointers [] array. Don't forget the comma on the preceding pointer.
* In the Definitions.h source file:
  + Define n, the index to your message pointer, in the sysLogPointers[].
  + #define (the task’s identifying 2 or 3 character prefix)…. n.

To report a diagnostic/error:

* Declare your sysLogControl control structure.
  + sysLogControl mySysLogCtlStruct…;
* Initialize mySysLogCtlStruct.
* Turn the interrupts off.
* Push the control structure onto the FIFO (sysLogQueue[] in Global.ino) and get the returned index to it.
  + mySysLogQueue Idx = PushSysLog(&mySysLogCtlStruct)
* Turn the interrupts on.
* Monitor the status in sysLogQueue[mySysLogCtlStructIdx] to see if the logging has completed.

**SD Read-Write**

The SD read-write task