CSR

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
CENTER FOR SPACE RESEARCH
CAMBRIDGE, MASSACHUSETTS 02139

REVISION LOG

TITLE: Software Detailed Design
Software Housekeeper

DOC. NO.
36-53220 Rev. A

Revision	Date (mm/dd/yy)	ECO No.	Page(s) Affected	Reason	Approval
01	5/11/95	36-252	all	Initial Release	DH
A	4/29/96	36-610	all	Updated to reflect current telemetry design. Added LED ping-pong feature.	5/15/96

28.0 Software Housekeeper (36-53220 A)

28.1 Purpose

The Software Housekeeper provides the statistical repository for the various active tasks. Periodically it delivers the database to be telemetered, and begins a fresh accumulation of software data. This process is initiated during start-up and persists until CPU reset.

The frequency with which the housekeeper attempts to deliver packets is one per minute (patchable).

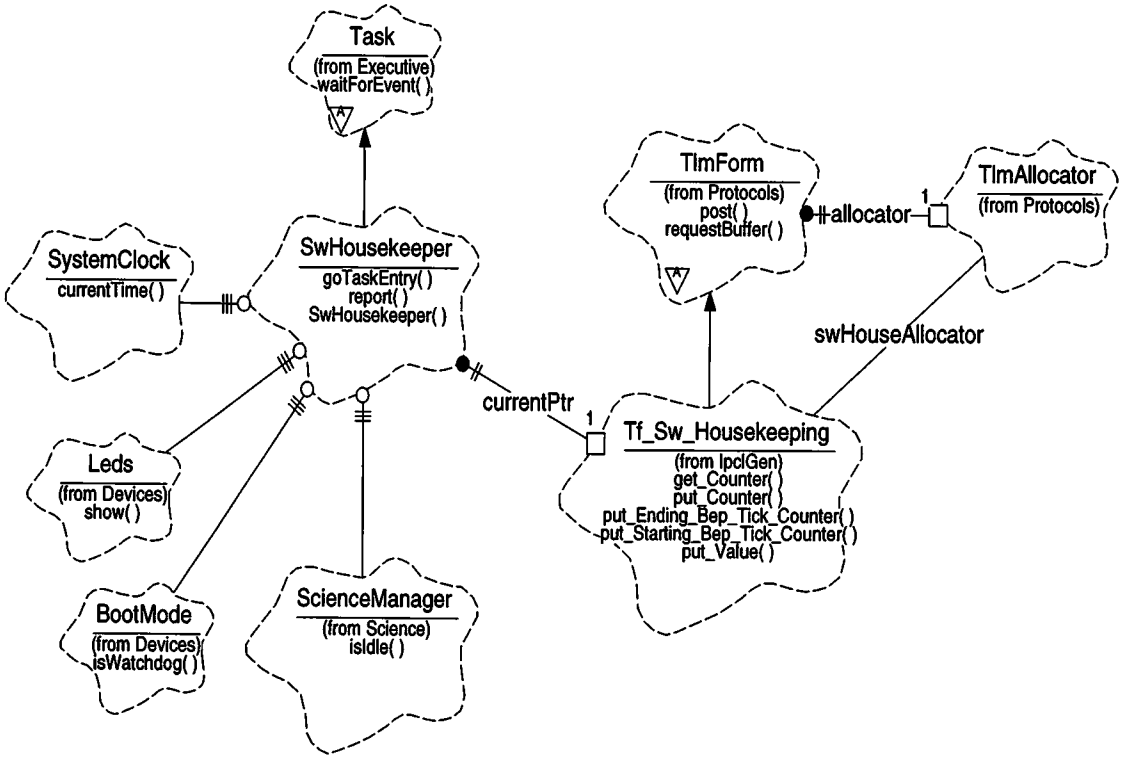
28.2 Uses

Use 1:: A method of acquiring reported statistical values
Use 2:: A system for periodically delivering the accumulated statistics.
Use 3:: Periodically indicate instrument state using software discrete telemetry (LEDs)

28.3 Organization

Figure 126 illustrates the relationship between the classes used by the Software Housekeeper.

FIGURE 126. Software Housekeeper Class Relationships



The Software Housekeeper uses the ***Executive***, ***Protocols***, ***Science*** and ***Devices*** class categories.

SwHousekeeper - This class is a subclass of ***Executive::Task***. It is responsible for accumulating and delivering software housekeeping statistics reported by other processes. It provides functions which clients use to report statistics (`report`), and includes a main task function (`goTaskEntry`).

Tf_Sw_Housekeeping - This class encapsulates the representation of a telemetry packet. It is a subclass of ***Protocols::TlmForm*** and is generated by the IP&CL code generator. The **SwHousekeeper** contains two instances of this class, which are used to format and post software housekeeping telemetry packet buffers. It provides functions which write the starting and ending integration times (`put_Starting_Bep_Tick_Counter`, `put_Ending_Bep_Tick_Counter`), read and write the current statistic counter (`get_Counter`, `put_Counter`), and set the value field associated with the statistic (`put_Value`).

SystemClock - This class provides the Back End Processor tick count (`currentTime`) which is included with the Software Housekeeping Database. It is provided by the ***Executive*** class category.

Leds - This class is provided by the ***Devices*** class category and is responsible for setting the current software discrete telemetry value (`show`). The **SwHousekeeper** class uses this class to indicate the current state of the instrument software.

BootMode - This class is provided by the ***Devices*** class category and is responsible for providing the cause of the most recent reset of the Back End Processor. The **SwHousekeeper** uses this class to determine if the Watchdog timer caused the most recent reset of the Back End (`isWatchdog`). This information is used to indicate the current state of the instrument in the software discrete telemetry.

ScienceManager - This class is provided by the ***Science*** class category and is responsible for managing science runs. The **SwHousekeeper** uses this class to determine whether or not a science run is active (`isIdle`) This information is used to indicate the current state of the instrument in the software discrete telemetry.

TlmAllocator - This class is provided by the ***Protocols*** class category and is responsible for managing pools of telemetry packet buffers. All software housekeeping telemetry packets are allocated from a single pool, managed by the *`swHouseAllocator`* instance of this class.

TaskMonitor - This class (not shown) is a subclass of ***Executive::Task***. It is responsible for insuring that active tasks are responsive, (not subverted by an SEU, etc.).

28.4 Scenarios

28.4.1 Operational Overview

The Software Housekeeper task has a quiescent period during which statistics are accumulated. When activated, it initiates an attempt to obtain a fresh database; failing, it continues accumulating into the current telemetry packet data array; succeeding, a fresh new array becomes available and the former packet is delivered to be telemetered. Since statistics are accumulated directly into telemetry buffers, they are subject to single-event upsets (SEUs). The expected SEU rate into the 1Mbyte telemetry buffer memory is from 1 to 100 hits per day, and the maximum size of a single telemetry buffer is 4092 bytes. A housekeeping data point may be corrupted about once every two or three days.

On each attempt, the housekeeper sets the software discrete telemetry bits (LEDs) to indicate the current state of the instrument (see Section 4.3.2).

These states include:

Science is idle

Science is running

Instrument was reset by the watchdog timer and Science is idle

Instrument was reset by the watchdog timer and Science is running

Each of these states use two different discrete telemetry codes. The housekeeper switches between each code on each iteration to indicate to the ground that the housekeeper is active (see the ACIS Software IP&CL, MIT 36-5302.0204, for the formal list of code assignments).

The software discrete bi-levels are periodically sampled by the spacecraft via the RCTU, and are included in the engineering portion of telemetry. The sample rate is TBD, but is expected to be once per Major Frame (i.e. about twice a minute). The final location of the bi-level signals within the telemetry frame, and their sample rate will be specified in the final version of the AXAF-I Instrument Program and Command List (IP&CL), provided by TRW (part # TBD).

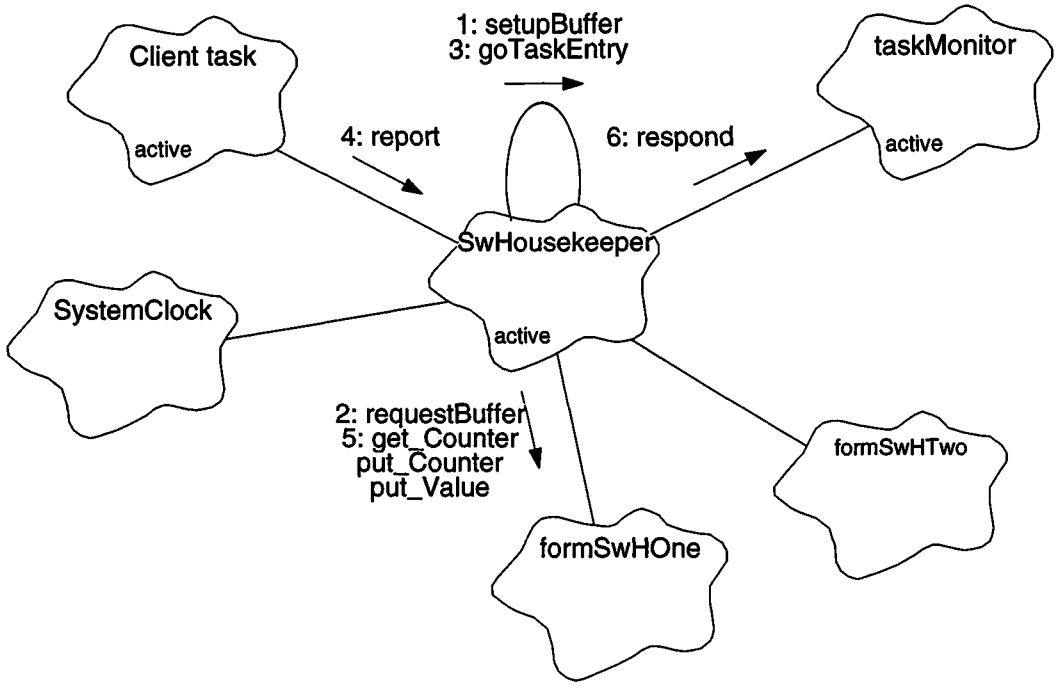
Each of the housekeeping statistics are comprised of two items; the number of times each entry was reported during the period, and an associated value. An enumerated list is used to specify the data items reported. The accumulating process will increment the count of occurrences in the first item of the two item data structure installed at the enumerated database location as each incident is deposited. The associated values' meaning and representation is entirely application dependent. It is stored as an unsigned integer.

Any process may be a client of the software housekeeper. It is the clients responsibility to deliver whichever statistical data are desired. The housekeeper does not solicit items to be included in the statistics.

28.4.2 Use 1: A method of acquiring reported statistical values

Figure 127 illustrates the operation of the Software Housekeeper data accumulation process.

FIGURE 127. Software Housekeeper Accumulation



1. During the system initialization, the constructor, `SwHousekeeper()`, passes a pointer to *`formSwHOne`* to **`SwHousekeeper::`**`setupBuffer()` to initialize the first packet buffer. Once the buffer is setup, it sets the form pointer, *`currentPtr`*, to point to *`formSwHOne`*. Once running, the housekeeper switches this pointer between *`formSwHOne`* and *`formSwHTwo`* to allow statistic accumulation while the other's telemetry packet buffer is being sent.
2. `setupBuffer()` uses *`formSwHOne`*`.requestBuffer()` to obtain the initial telemetry packet buffer, and initializes the contents of the buffer (see Section 28.4.3).
3. Once the system starts multi-tasking, the housekeeper's task function, `goTaskEntry()` is invoked. This function contains an infinite loop during which it waits in `intervalWait()` (not shown) while statistical data accumulates, then delivery of the housekeeping database packet is initiated.
4. The client task, any active software process, may deposit information whenever necessary, or desirable using **`SwHousekeeper::`**`report()`.
5. The software housekeeper `report()` function disables interrupts by declaring an instance of **`IntrGuard`** (not shown). It then reads the current statistic value using *`currentPtr`*`->get_Counter()`, increments the returned value and writes it back

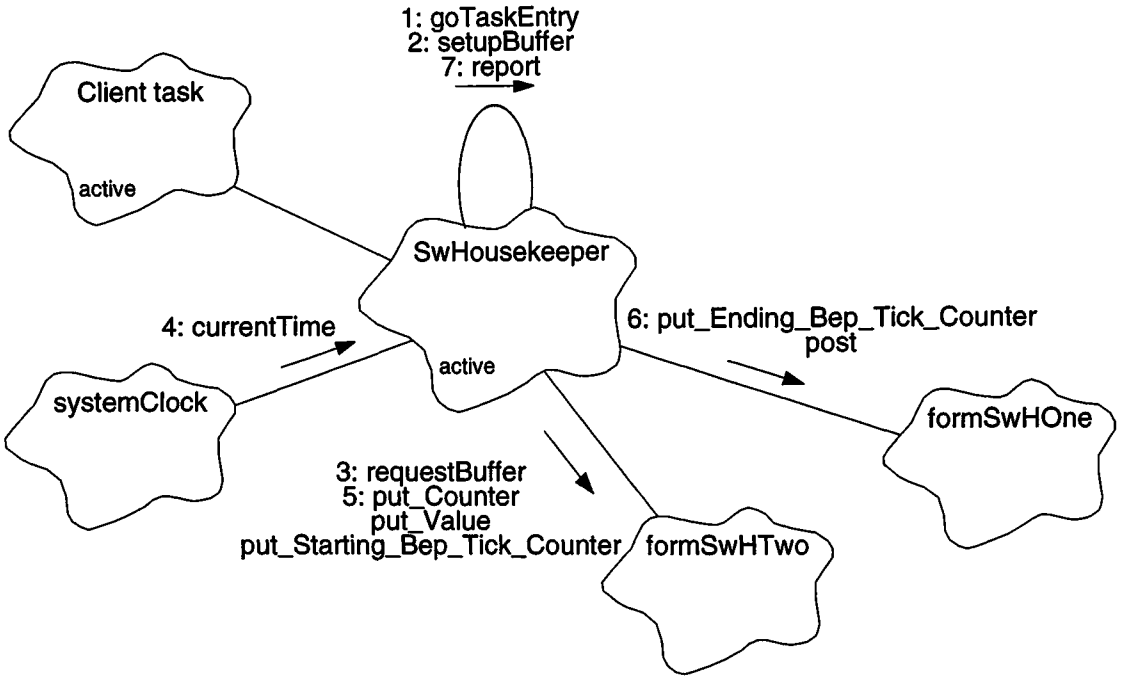
into the value using *currentPtr*->put_Counter(). It then writes the report value using *currentPtr*->put_Value(). Upon returning, the **IntrGuard** instance is destroyed, and the previous interrupt state is restored.

6. Meanwhile, the **TaskMonitor**, on its own schedule, will query the software housekeeper to determine if it is viable. During its accumulation period the housekeeper would reply using **TaskMonitor::**respond().

28.4.3 Use 2: A system for periodically delivering the accumulated statistics

Figure 128 illustrates the operation of the Software Housekeeper packet delivery process.

FIGURE 128. Software Housekeeper Delivery of Statistics



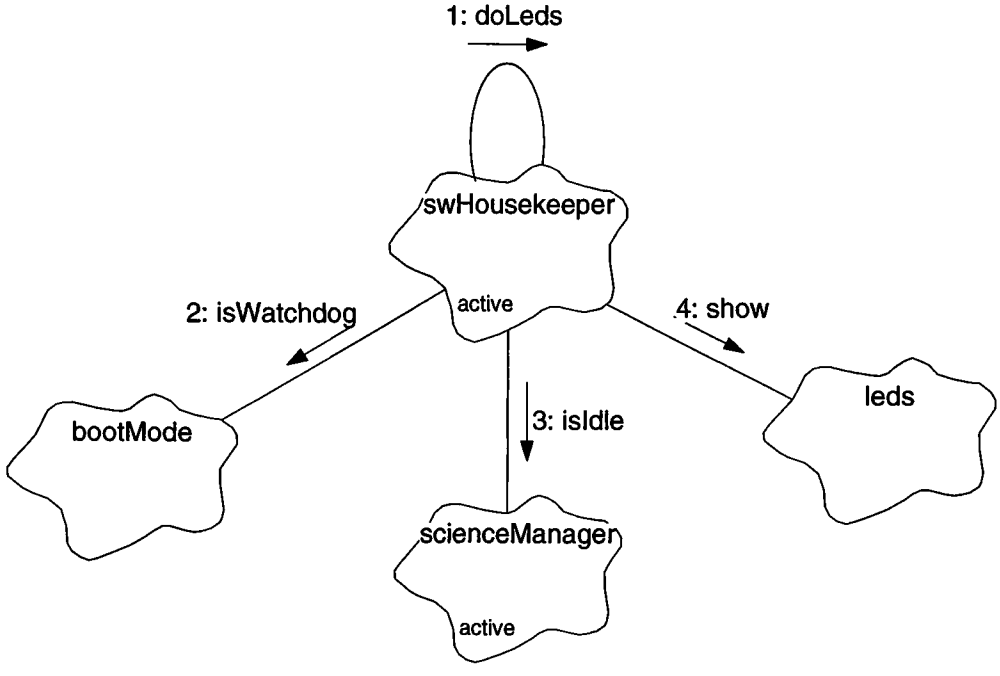
1. The software housekeeper thread, `goTaskEntry()`, associates the local pointer, *`nextPtr`* with the packet instance-in-waiting. During operation, the software housekeeper will alternate between this packet instance and the instance *`currentPtr`* associated in the constructor. The software housekeeper then enters its main task loop during which data accumulates and delivery of the housekeeping database packet is initiated.
2. At the end of the accumulation period, the housekeeper passes *`nextPtr`* to `setupBuffer()` to attempt to stage the next accumulation telemetry buffer.
3. `setupBuffer()` calls *`form`*`->requestBuffer()` to attempt to obtain a second telemetry packet buffer.
4. If the request succeeds, `setupBuffer()` calls *`systemClock`*`.currentTime()` to obtain the current BEP tick counter value (NOTE: This 1/10th second operating system timer is not the same as the science timestamp counter).
5. `setupBuffer()` then zeros the buffer's counter and value entries using *`form`*`->put_Counter()` and *`form`*`->put_Value()`. It then stores the starting integration time using *`form`*`->put_Starting_Bep_Tick_Counter()`. `setupBuffer()` supplies its caller with the counter value to delimit the end of the previous buffer's integration.

6. Once the second buffer has been obtained and initialized, `goTaskEntry()` swaps telemetry packet buffers by switching the values of *`currentPtr`* and *`nextPtr`*, through the temporary pointer, *`priorPtr`*. It then stores the ending integration BEP tick counter into the old buffer using *`priorPtr`*`->put_Ending_Bep_Tick_Counter()`. It then posts the old buffer to telemetry using *`priorPtr`*`->post()`.
7. Should another buffer not be available, software statistics will continue to accumulate in the current packet buffer. The housekeeper will record the incident in the database using its own `report()` function.

28.4.4 Use 3: Indicate instrument state to software discrete telemetry (LEDs)

Figure 129 illustrates the method used by the Software Housekeeper to periodically update the software discrete telemetry levels.

FIGURE 129. Update Instrument State Indicators



1. Once per iteration of its infinite loop, the Software Housekeeper's main task function, `goTaskEntry()`, calls `doLeds()` to update the software discrete telemetry state.
2. `doLeds()` determine if the most recent reset was due to the watchdog timer using *`bootMode`*`.isWatchdog()`.
3. `doLeds()` determine if a science run is underway using *`scienceManager`*`.isIdle()`.
4. `doLeds()` uses the acquired boot and science state information, and the current state of the *`aPhase`* instance variable to select the appropriate LED code, and writes the code to the software discrete telemetry bits using *`leds`*`.show()`.

28.5 Class SwHousekeeper

Documentation

The `SwHousekeeper` periodically initiates telemetering of accumulated statistical data provided by various software elements. During the normal course of their operation, the software tasks, which are intended to provide statistics, will call the software housekeeper and deliver the pertinent data which will be installed into the software housekeeping data structure. Periodically, the housekeeper will attempt to obtain a packet buffer for the replacement database. Failing, it will report the incident, and continue to accumulate in the current database. Succeeding, it will swap databases, initiate delivery of the acquired data to the telemetry service, and begin a new accumulation period.

Export Control: Public

Cardinality: 1

Hierarchy

Superclasses: **`Task`**

Public Interface

Operations:

```
SwHousekeeper()
goTaskEntry()
report()
```

Private Interface

Operations

```
doLeds()
intervalWait ()
setupBuffer()
```

Has-A Relationships:

`Tf_Sw_Housekeeping` *`formSwHOne`*: This is an instance of a software housekeeping packet form. A packet buffer in bulk memory will be associated with this instance.

`Tf_Sw_Housekeeping` *`formSwHTwo`*: This is an instance of a software housekeeping packet form. A packet buffer in bulk memory will be associated with this instance.

Tf_Sw_Housekeeping *****`currentPtr`: This pointer is used to indicate which software housekeeping packet is currently available to be filled with data

Boolean `aPhase`: This indicates the current software discrete telemetry (LED) ping-pong state. This code toggles between `BoolTrue` and `BoolFalse` as the LEDs are periodically updated.

const unsigned `accumInterval`: This variable contains time, in BEP timer ticks (10 per second)), over which to accumulate software housekeeping statistics. If telemetry resources are available, one housekeeping packet will be sent after each interval. The interval duration is approximately 1 minute.

const unsigned `statCount`: This is the largest statistic code used by the housekeeper (NOTE: The class constructor initializes this value to SWH_MAX_STAT = 64. To modify this value, patch the constructor.).

Concurrency: Active

Persistence: Persistent

28.5.1 SwHousekeeper()

Public member of: **SwHousekeeper**

Arguments

unsigned *taskId*: Identifies the task being constructed.

Documentation

The SwHousekeeper() constructor initiates acquisition of the first software housekeeping statistics database packet.

Preconditions

It is expected that functions which provide a telemetry packet will be in place when this function is constructed.

Semantics

SwHousekeeper() uses setupBuffer(), which acquires the packet buffer. The Buffer MUST have been obtained, or a fatal error will be generated!

Concurrency: Guarded

28.5.2 doLeds()

Private member of: **SwHousekeeper**

Return Class: **void**

Documentation

This function blinks the software discrete telemetry bits (LEDs) to indicate the current state of the instrument. The state of the LEDs are changed on each call to this function.

Semantics

This list of housekeeping LED codes is as follows, where their values are defined in the ACIS Software IP&CL, MIT 36-53204.0204:

LED_WD_SCIENCE_A
LED_WD_SCIENCE_B
LED_WD_IDLE_A
LED_WD_IDLE_B
LED_RUN_SCIENCE_A
LED_RUN_SCIENCE_B
LED_RUN_IDLE_A
LED_RUN_IDLE_B

Use *bootMode*.isWatchdog() to determine if watchdog cause the most recent reset. If so, select the LED_WD_* set of LED codes, otherwise, select the LED_RUN_*.

Use *scienceManager*.isIdle() to determine if science is performing a run. If so, select the LED_*_SCIENCE_* set of codes, otherwise, select the LED_*_IDLE_* set.

Toggle between LED_*_*_A and LED_*_*_B on each call to the function. Use the *aPhase* instance variable to determine which blink state to use. If *aPhase* is *BoolFalse*, select the LED_*_*_B set of LEDs, and set *aPhase* to *BoolTrue*. If *aPhase* is already *BoolTrue*, use the LED_*_*_A set of codes, and set *aPhase* to *BoolFalse*.

Use the selection code to lookup the corresponding LED code, and write the code using **Leds**::show().

.

Concurrency: Guarded

28.5.3 goTaskEntry()

Public member of: **SwHousekeeper**

Return Class: **void**

Documentation

goTaskEntry() is the initiation point for the software housekeeper. On a regular basis, it initiates dispatch of the accumulated housekeeping statistics being telemetered.

Preconditions

The constructor for SwHousekeeper() has initiated the first packet.

Semantics

On start-up, goTaskEntry() associates a pointer with the second **Tf_Sw_Housekeeping** software housekeeping packet. It then enters a FOREVER loop, and remains inactive in intervalWait() for a fixed time period (*accumInterval*) during which statistics are accumulated into the telemetry buffer. When activated, it initiates an attempt to obtain a replacement empty telemetry packet buffer using setupBuffer(). If it is successful, it will switch packet buffer pointers and begin accumulating into the fresh packet buffer database, store the ending integration time into the prior buffer, and post it to telemetry using **TlmForm::**post(). Otherwise, it will record the condition using **SwHousekeeper::**report() to record a skipped software housekeeping packet delivery (statistic), before continuing accumulation in the same packet during the ensuing interval. At the end of each iteration, goTaskEntry() calls doLeds() to update the software discrete telemetry codes.

Post Conditions

This procedure never returns.

Concurrency: Guarded

28.5.4 intervalWait()

Private member of: **SwHousekeeper**

Return Class: **void**

Documentation

`intervalWait()` idles the process for a predefined period. During this interval, housekeeping reports are filed by various entities. Also, during this accumulation period, `intervalWait()` will respond to `taskMonitor()` interrogations.

Semantics

`intervalWait()` idles in `waitForEvent()` while it waits for the monitor (are you alive) interrogation events or for *`accumInterval`* number of Science Frame pulse timing tick events. It will respond to the interrogation and will count the requisite number of pulses before returning.

Concurrency: Guarded

28.5.5 report()

Public member of: **SwHousekeeper**

Return Class: **void**

Arguments

SwStatistic *statisticId*:: Identifies the statistic being recorded.

unsigned *value*:: Contains a (possibly irrelevant) associated value.

Documentation

report() is the vehicle provided to accumulate the referenced statistics which the software housekeeper will deliver.

Semantics

First, the function tests *currentPtr* and returns if it is NULL. If *currentPtr* is not NULL, it verifies that the *statisticId* is within range (i.e. be less than *statCount*). If not, it sets the statistic id to SWSTAT_SWHOUSE_RANGE. It then disables by declaring an **IntrGuard** instance. It then uses *currentPtr*->get_Counter() and *currentPtr*->put_Counter() to read, increment and write the counter indexed by *statisticId*. It then uses *currentPtr*->put_Value() to store *value*. Upon returning, the **IntrGuard** destructor restores the original interrupt enable state.

Concurrency: Synchronous

28.5.6 setupBuffer()

Private member of: **SwHousekeeper**

Return Class: **Boolean**

Arguments

Tf_Sw_Housekeeping* *form*: : Points to telemetry formatter

unsigned& *starttick*: : Used to return integration start time

Documentation

This function sets up a telemetry packet buffer for the telemetry form, pointed to by *form*. If a buffer is successfully obtained, the function sets *starttick* to the starting integration BEP timer-tick value and returns *BoolTrue*. If a telemetry packet buffer is not available at the time of the call, the function returns *BoolFalse*.

Semantics

Call *form*->requestBuffer(). If successful, call *systemClock*.currentTime() to obtain the current timestamp, zero the statistics counters and values using *form*->put_Counter() and *form*->putValue(). Store the starting timer tick using *form*->put_Starting_Bep_Tick_Counter() and return *BoolTrue*. If *form*->requestBuffer() fails to obtain a packet buffer, return *BoolFalse*.

Concurrency: Guarded