Creative uses of the timer in HAL 2000 Home Automation Applications

Abstract: This document discusses the timer function and it's uses and limitations in the HAL programming environment. It uses the real world example of programming a bathroom motion detector and X10 light switch using sophisticated control logic.

Revision History:
Version 1.0 Released 10.07.2002 supports HAL Version 1.3.81 and earlier
Version 2.0 Released 04.02.2003, updated to take advantage of version 2.0.16 timer functions to streamline and simplify the logic.

The Goal

The goal of any automation is to cause the device or process being automated to function very much as if a human being were at the helm and operating the controls, but without requiring the attention of a human.

Timer Features

Hal incorporates a timer object into the tool kit of programmable sensors. There are four timer actions possible.

Set the Timer – The timer may be set to any value between 1 second and approximately 24 hours.

Stop the Timer – The timer may be stopped at any time. Stopping the timer avoids executing the actions tied to the expiration.

Allow the timer to Expire – Once set, the timer counts down until it expires. Upon expiration, it may be used as a trigger event for any rule.

Test the Timer – The status of the timer (whether it is running or not, but not the value) may be used as a Secondary Condition in any rule

Timer Limitations

This timer functionality is very useful, but there are limits to it's functionality. There are a couple of things we could wish to do with a timer, but which are not supported in the current HAL implementation.

Read the Timer – I can think of circumstances where one might wish to read a timer as a secondary condition, and take action before it expires. For example, if you have a timer designed to turn out a bathroom light on lack of motion, it might be nice to be able to recognize when it is a few seconds from expiration and blink the lights before turning them out. Creative programming can easily overcome this limitation, although it will require the use of multiple timers.

Force expiration of a timer – The opposite of stopping a timer is forcing it to expire immediately, thus triggering the action it is designed to invoke. Ideally, we should be able to set a timer to 0, and obtain immediate action. We can however load a timer with 1 second, which almost accomplishes the same thing, albeit with some very slight delay.

Minor limitations notwithstanding, the timer is nonetheless a very useful device, and a little creative thought will allow one to work within it's limits. The purpose of this document is to provide programming guidance in the use of timers, with real-world programming examples.

Why Use a Timer?

HAL has the capability of scheduling an event at some future point. We can easily decide to turn a light on or off five minutes, or five hours from now. At first thought, it may not be clear why a timer is desirable for this application.

Consider the simple case of turning on a light when someone enters a bathroom, and then turning it off a few minutes later. We could define this with a single rule, and no flags or timers. Create a rule structured like this.

Sensor Motion
IF *Sensor Motion* is ON (TE)
Then Turn *light* ON
In Five minutes Turn *light* OFF

At first glance, this seems sufficient. The light is turned on upon entering the room and turned off five minutes later by a Scheduled Event.

Unfortunately, it will be turned off regardless of the time the room actually remains occupied. Indeed, a scheduled turn-off event will occur five minutes after each time the motion sensor detects motion. The HAL schedule would likely be flooded with many repetitive scheduled events. This will result in a storm of Turn-on and Turn-off events, flooding the limited data bandwidth of the power line. Plus, the light will likely blink on and off a number of times, and HAL will most likely become too busy sending these commands to respond to other tasks.

Obviously then, this will not work. What we need is a solution whereby the light is turned on, and a timer is started, which timer is reset whenever it detects additional motion, and which then will expire and turn off the light once motion stops.

Timer Example, Take One

A more satisfactory approach makes use of three rules, and a timer. Consider this example of a HAL program.

On first detecting motion, we turn on the light and treat the light status as a flag that signifies the room is occupied. Then we load a default interval into a timer. Future incidences of motion only do one thing. As long as the light is on, then additional motion simply reloads that timer with the same default interval of five minutes. As long as motion is detected at least every five minutes, the timer can never expire.

RULE Sensor Motion Detected
IF *Sensor Motion* is ON (TE)
AND *Light* is OFF (SC)
THEN Turn light ON

RULE Sensor Motion Timer
IF *Sensor Motion* is ON (TE)
THEN Set *Timer Occupancy* to 5 Minutes

RULE Timer Motion Expired
IF *Timer Occupancy* is Expired (TE)
THEN Turn Light OFF

When the timer finally does expire, the room is presumed unoccupied and the light is turned off.

The light is turned on when the room is occupied, and remains on until five minutes after the last motion is detected. Using the status of the light as a filter, we avoid sending redundant X10 messages. Only one X10 ON and one X10 OFF command are actually sent.

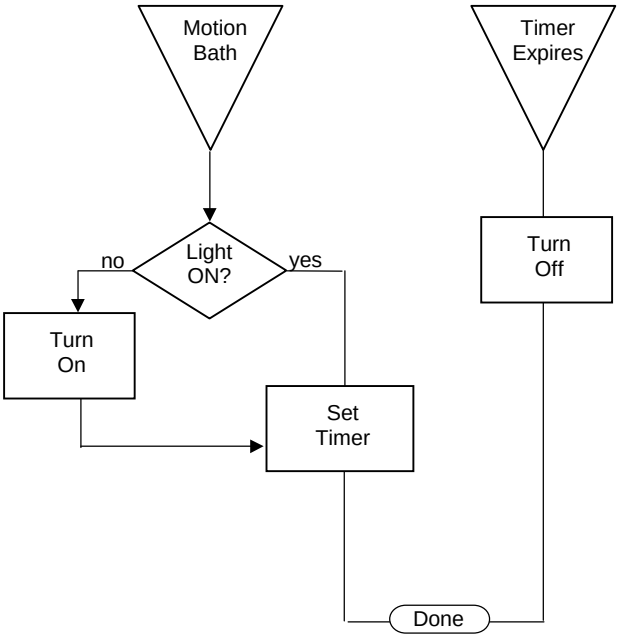
This works fairly well. Indeed, most motion detector controlled lights do no more than this. But it has two flaws.

First, it assumes the occupant will not remain motionless more than five minutes. A little practical experience will show that this is false. The occupant sometimes will not only remain motionless for more than five minutes, but also occasionally even when there is some motion the detector will miss it. We could solve that problem by making the interval longer than five minutes.

But that brings us to the second drawback. The light will stay on a full five minutes after the last detected motion.

Usually the last detected motion is the act of leaving the room. Hence it will always stay on for the interval defined, unnecessarily wasting power. Ideally, we want to use a relatively long interval when the room is occupied, but shorten the interval when the room is empty. That is the trick. Or we could simply warn the occupant.

We can document the logic flow, if not the exact instruction sequence with a small flowchart. This is a very good exercise as it will help later when you need to troubleshoot your work.



Timer Example, Take Two

One approach to improving our logic might be to use an additional timer, so that we can blink the lights before turning them out, thus giving the occupant fair warning to wave a hand or something.

RULE Sensor Motion Detected
IF *Sensor Motion* is ON (TE)
AND *Light* is OFF (SC)
THEN Turn light ON

RULE Sensor Motion Timer
IF *Sensor Motion* is ON (TE)
THEN Set *Timer Occupancy* to Five Minutes
Set *Timer Warning* to 4 minutes 30 seconds

RULE Timer Motion Expired
IF *Timer Occupancy* is Expired (TE)
AND *Light* is ON (SC)
THEN Turn Light OFF
Stop *Timer Warning*

RULE Timer Warning Expired
IF *Timer Warning* is Expired (TE)
AND *Light* is ON
THEN DIM Light 70%
In 2 seconds turn light ON

If we could read the timer and act when it is near expiring, we could avoid the use of a second timer. But since this is not a supported function we work around this limit by adding a second timer, set to 30 seconds shorter interval than the first, and as a result of this expiration we dim the light momentarily. Our occupant, thus forewarned that the lights are about to go out, can then wave a hand or otherwise trigger the motion detector to reset the timers.

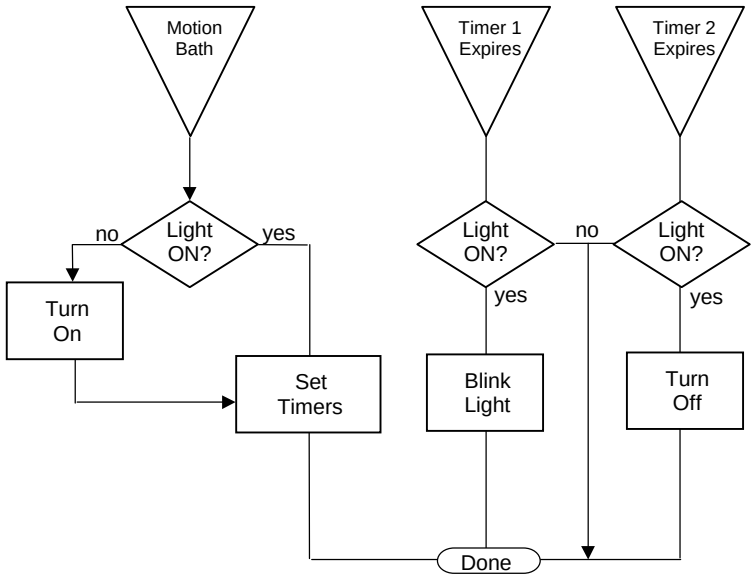
We use a Secondary Condition to filter the expiration of the new timer, else the timer might expire sometime when the light is not on and turn on the light unintentionally.

This very nearly solves the problem. Well, almost. The user still must respond to the change in light level, perhaps not practical if one is in the shower.

There is a subtle point here that should be pointed out. Some actions in HAL are very resource intensive, tying up the machine, and other actions are relatively benign. Various factors affect the level of resources a given action requires. The least demanding action is setting or clearing a flag. If the flag in question is one that is not saved to the hard disk, and logging is turned off, it uses the least resources possible. The price of not logging is that if the machine crashes, or is restarted, the state of the flag will not be remembered. Think carefully how each flag is to be used and if there is no need for it to be remembered through system shutdowns, then don't save it.

Sending X10 signals is very resource intensive. Therefore we want to send as few X10 signals as possible. Presumably the motion detector is already sending X10 signals every time it detects motion. Let's not add to the traffic.

This logic works pretty well. The light will come on as soon as the occupant is detected, remain on as long as motion is detected at least once every five minutes, and finally warn any occupant before extinguishing the light.



This is still not a completely satisfactory solution though. The test for acceptability of any automation is whether the result behaves very similarly to the way a human manually operating the system would behave.

The problem of leaving the light on after the room is vacated is still with us, requiring a fairly short timer interval, and the occupant still must respond to the warning signal and re-trigger the motion detector if the occupant is quiet longer than four minutes and 30 seconds.

RULE Sensor Motion Detected
IF *Sensor Motion* is ON (TE)
AND *Light* is OFF (SC)
THEN Turn light ON
Set flag *Possible_Exit* to TRUE

RULE Sensor Motion Timer
IF *Sensor Motion* is ON (TE)
THEN Set *Timer Occupancy* to 20 Minutes
Set *Timer Warning* to 19 minutes 30 seconds
Set Timer *Possible_Exit* to 15 sec.

RULE Timer Motion Expired
IF *Timer Occupancy* is Expired (TE)
AND *Light* is ON (SC)
THEN Turn Light OFF
Stop *Timer Warning*

RULE Timer Warning Expired
IF *Timer Warning* is Expired (TE)
AND *Light* is ON
THEN DIM Light 70%
In 2 seconds turn light ON

RULE Timer Possible_Exit Expired
IF *Timer Possible_Exit* is Expired (TE)
THEN Set Flag *Possible_Exit* to FALSE

RULE Sensor Motion Hallway
IF Sensor Motion_Hallway is ON (TE)
AND flag *Possible_Exit* is TRUE (SC)
THEN Set *Timer Occupancy* to 3 Minutes
Set *Timer Warning* to 2.5 minutes

In short, the function we have now is pretty good, but does not quite emulate the action a human would take if strictly manual light control were being used.

What we need is a more reliable method of detecting the vacancy of the room. Then we could use a much longer interval on the timers, perhaps even 30 minutes or longer, but reliably switch off the light immediately when the occupant leaves.

Timer Solutions, Case Three

There are various hardware solutions to detecting occupancy that can reliably determine a room is empty, but this article is about using creative logic to solve problems in HAL, not about home automation technology. For the purpose of this exercise we are assumed to be working within the constraints of common inexpensive motion detectors.

So how can we more reliably detect the exit of the occupant? One solution might be to look at other, nearby motion detectors. When installing home automation systems, we almost never install only one motion detector. So let's assume there is another motion detector in the hallway outside the bath. Let's explore adding another timer and flag to our logic, and then look at the logic associated with the hypothetical second motion detector.

Now we have three timers, two motion sensors one flag, and six rules encompassing twenty-two statements. The logic has become rather complex, and we've only defined the logic for controlling one light. But the function is now fairly complete. The occupancy timer defaults to 20 minutes, an interval far better suited to weathering "quiet time" without expiring prematurely. The upper limit to this timer really is how long we are willing to leave the light on if the exit signal is missed.
Each time motion is detected we test the hypothesis that this was the motion of the occupant exiting the room, and look for motion immediately thereafter in the hallway outside. If we see hallway motion within a 15-second interval, we conclude the room has been vacated and shorten the timer to three minutes.

It Works, But Can we Simplify It?

Actually it is possible to eliminate a flag and a rule and retain the completeness of the logic.

The Flag *Possible_Exit* is redundant because we can actually test for the operation of the timer itself as a Secondary Condition (SC).

By doing this, we also eliminate the need to have a rule to clear the flag on expiration.

This reduces us to only five rules, no flags, and 19 statements.

RULE Sensor Motion Detected
IF *Sensor Motion* is ON (TE)
AND *Light* is OFF (SC)
THEN Turn light ON

RULE Sensor Motion Timer
IF *Sensor Motion* is ON (TE)
THEN Set *Timer Occupancy* to 20 Minutes
Set *Timer Warning* to 19 minutes 30 seconds
Set Timer *Possible_Exit* to 15 sec.

RULE Timer Motion Expired
IF *Timer Occupancy* is Expired (TE)
AND *Light* is ON (SC)
THEN Turn Light OFF
Stop *Timer Warning*

RULE Timer Warning Expired
IF *Timer Warning* is Expired (TE)
AND *Light* is ON
THEN DIM Light 70%
In 2 seconds turn light ON

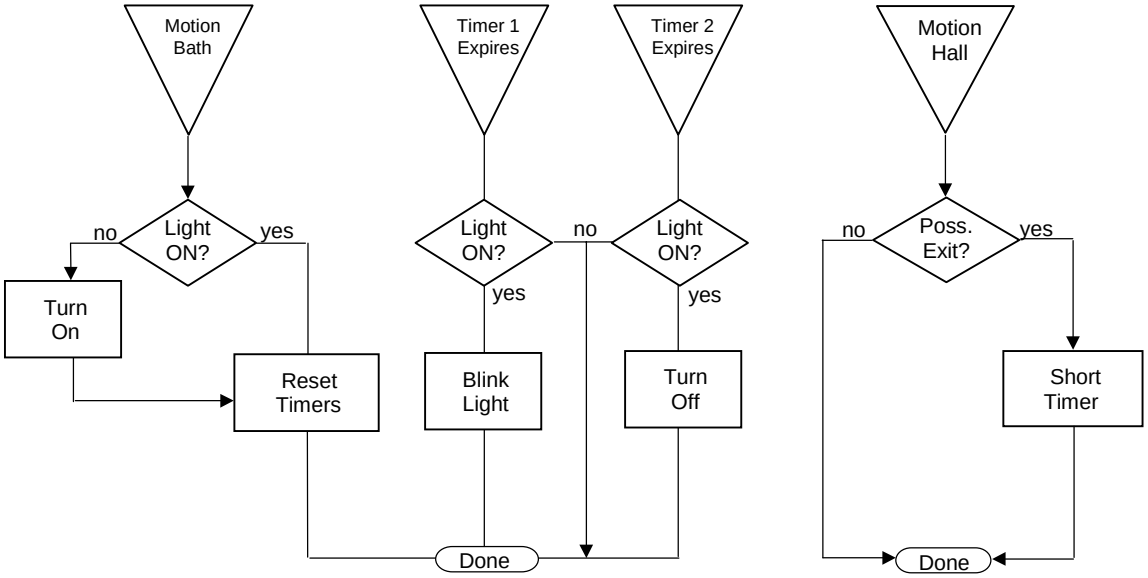
RULE Sensor Motion Hallway
IF Sensor Motion_Hallway is ON (TE)
AND *Possible_Exit* Timer is Running (SC)
THEN Set *Timer Occupancy* to 3 Minutes
Set *Timer Warning* to 2.5 minutes

Shortening the timer instead of shutting the light off immediately allows for the possibility that another person has tripped the hallway detector. In this hopefully rare case, the occupant still has a two and a half minute interval to again trip the motion detector, and worst case sees the dim/bright signal, and can respond.

In the opposite case, if the exit signal is missed, the light will still stay on the full twenty-minute interval. We could probably add more logic to handle this case, but we have probably reached the point of diminishing returns in that we could add more logic complexity, but get very little improvement in function. We have changed from a near certainty that the light would stay on five minutes to a rare possibility that the light MIGHT stay on 20 minutes.

We can of course fine-tune the timer values to optimize the performance. In a quiet home with minimal motion, this should be extremely reliable, but in a very busy home, it might leave the lights on for the 20 minutes occasionally.

This logic now more nearly imitates the process a human might follow, including perhaps even the occasional



forgetfulness in turning the light off. The difference is that the system will still turn the light off eventually.

Are We Done Yet?

In any engineering exercise, one must spend time not only looking at how a process works, but how it fails. Quality design is reflected as much in the failure modes as in the functions the design delivers.

We are using the X10 protocol for sending the ON and OFF commands. X10 is notorious for being unreliable. It is unreliable both in that sometimes phantoms turn-on things unintentionally and in that sometimes real commands are missed.

In the case of the phantom turn-on, there is usually not much we can do except wait for a human to notice and turn the light off. This is a problem if no human is present. In the case of a missed turn-on signal, the occupant is right there and can turn it on manually, but in the case of a missed turn-off signal, presumably the occupant has left and not noticed it. So once again there is a problem.

The power line has a very limited bandwidth for data, and we do not want to be sending redundant or repetitive signals to devices unnecessarily.

However, we do need to take some steps to enforce a "safe" failure mode to our lights. We must ensure that no matter what happens, our lights are eventually turned off, so we must balance the use of the limited resource (the power line bandwidth) against the need to ensure a fail-safe operation.

Since failures are, hopefully, rare, we can make an intelligent decision about what sort of acceptable time period a light may be allowed to be erroneously on. Since one of the purposes of using home automation is to save energy, this process possibly should be related to the power consumed by the light.

Perhaps a 50 watt light might be left on up to four hours, and a 200 watt light up to one hour. Also, we need to set a practical minimum as to how soon a light should be turned off. So we need yet another timer to periodically ensure a light that's supposed to be is really off.

RULE Timer Light Set
IF *Light* is ON (TE)
THEN set *Timer Light_Watchdog* to 3 hours

RULE Timer Light Expired
IF Timer *Light_Watchdog* is EXPIRED (TE)
THEN Turn *Light* OFF
set *Timer Light_Watchdog* to 3 hours

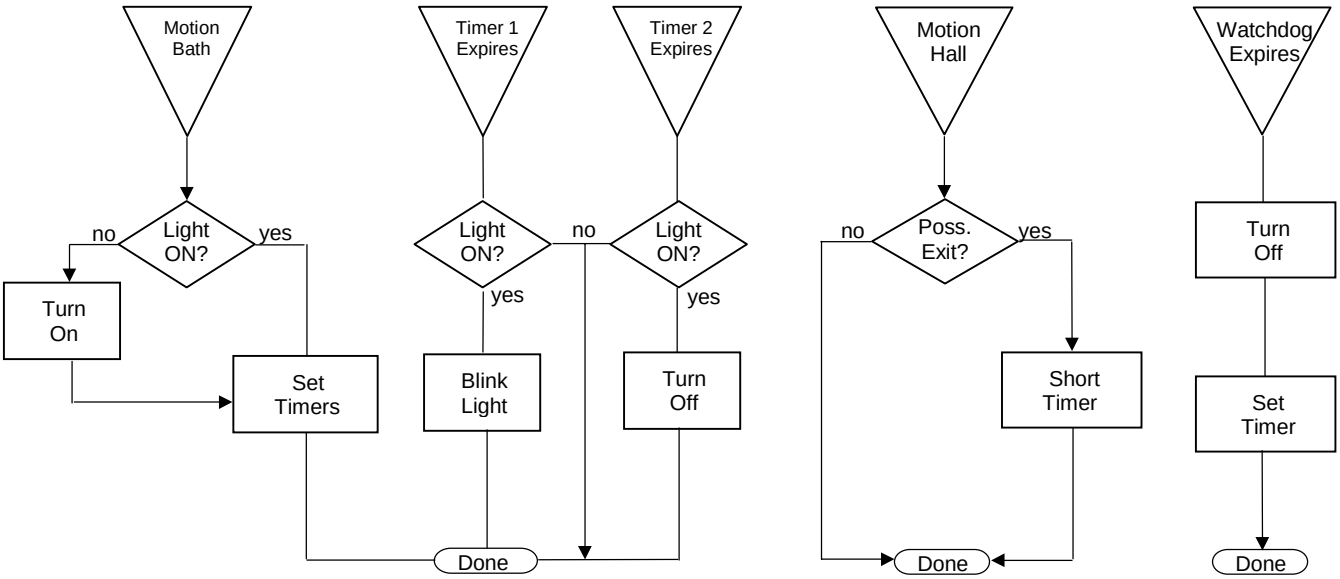
RULE Sensor Motion Timer
IF *Sensor Motion* is ON (TE)
THEN Set *Timer Occupancy* to 20 Minutes
Set *Timer Warning* to 19 minutes 30 seconds
Set Timer *Possible_Exit* to 15 sec.
Set Timer *Light_WatchDog* to 30 minutes

This "fail-safe" logic requires two more rules, encompassing five statements and another timer. We also add another statement to our original logic. The result is that a few minutes after the "exit signal" turns off the light, the fail-safe sends another turn-off command unless reset by another occupant entering the room and turning the light on again. Thereafter at three hour intervals, as long as the light is supposed to be off, redundant X10 OFF commands are sent to ensure the light is indeed off.

Notice there is a "race condition" the first time the motion detector is activated. The Sensor Motion Timer Rule sets it to 30 minutes, but the action of turning the light on simultaneously sets it to 3 hours. This shouldn't cause problems however as the next motion detection event will again set it to 30 minutes.

We thus ensure that if the light is turned on by a phantom, it will stay on a maximum of 3 hours. If the turn off signal is missed, it will be turned off again in about 20 minutes or so.

This makes for very reliable operation of the X10 motion controlled light.



A Final Word

That word is *Documentation***.** Documentation takes at least three forms.

At minimum, a text listing of the rules and macros in the program should be created, along with some explanatory comments. It's a shame that HAL does not allow at least minimal comment fields in the rules and macros.

Detailed documentation of the logic flow, complete with assumptions and a listing of flags and other variables used is the next step. This allows capture of not only the basic logic, but also the thinking behind that logic.

Creating a flowchart of the logic flow is extremely valuable although it is considerably more time consuming.

Flowcharting is a useful tool to examine the relationships between our rules. Flowcharting is a tedious and time-consuming process, and should be used judiciously. It's not usually worthwhile to take the time to diagram every logic step.

Documentation serves at least three purposes.

First, as an aid to the thought process for design and troubleshooting, it can be extremely useful. Every hour spent diagramming complex logic flow will pay back at least two hours in troubleshooting work saved at the end of the programming exercise by forcing the programmer to think more carefully about what we are asking the computer to do. Sometimes the work of flowcharting and documenting the logic is an essential part of the design process itself.

A second reason to consider doing at least a partial or high level flow chart of your logic is after the fact support.

Sooner or later you will have to revisit your programming gem. Perhaps you need to tweak it, enhance it, or copy it for another room. Maybe Automated Living will release a new version of HAL that changes some of your basic assumptions. In any case, carefully documenting your work, both with text and flowcharts, will make that effort much easier.

Finally, If you decide your logical construction is something you want to share with others, the effort of documentation and flowcharting pays off yet again. Having done the up front work of careful design, flowcharting and documentation means those valuable thoughts are now all on paper. Turning them into a proper document that others can study and learn from becomes almost trivial.