

# SMART HOME HACKS™

*Tips & Tools for Automating Your House*



O'REILLY®

*Gordon Meyer*

**HACK**  
**#61****Track Fuel Consumption**

Collecting information about fuel oil usage patterns can be straightforward, and when you know how much oil you're using, you can plan ahead and buy fuel when rates are lowest.

My house has a 1,000-gallon underground tank to store fuel oil for the heating season. Instead of filling the tank on a scheduled basis, which means the amount you pay varies as the price of fuel oil changes, it's better to purchase oil during the summer when the demand is lowest and fuel oil is less expensive. During the more expensive winter months, it's best to purchase only the amount of additional fuel you need, if any. This system has saved me about \$700 a year in fuel oil costs, which makes it well worth the effort.

To buy fuel *just in time*, I need to know how much fuel I have left and the rate at which I consume it. Unfortunately, due to the way the filler lines for my tank are installed, measuring the fuel level is unreliable. Instead, to figure out how much is remaining in the tank, I must keep track of how much fuel I've used since it was last filled.

The furnace's oil gun draws fuel oil from the tank and burns it to produce heat. The oil gun has a 1.25 tip on it, which means that it burns 1.25 gallons of fuel oil per hour. Said another way, it takes 22.5 seconds to burn one ounce of oil. To track the consumption of fuel oil, I need to track how long the gun is burning fuel on a daily basis. I hooked up a relay switch in parallel with the oil gun motor. The relay is connected to a [Powerflash module \[Hack #10\]](#). When the oil gun turns on or off, the relay causes the module to send X10 On and Off commands to the power line. I use [XTension \[Hack #17\]](#) to measure the time between each on and off signal, calculate the accumulated time, and calculate the fuel consumed each day, week, month, and season.

Table 5-2 lists the units that are necessary for this hack.

Table 5-2. Required units

Unit	Meaning
Line Pump	Universal module monitoring pump
Wireless Pump	Altered motion detector monitoring pump
The Oil Pump	Reflects state of the oil pump
PavgOn	Average pump on time
PmissdOff	Count of undetected Off states
PmissdOn	Count of undetected On states
PmaxOn	Longest running time so far

Table 5-2. Required units (continued)

Unit	Meaning
PtimesOff	Count of Off commands
PtimesOn	Count of On commands

To ensure I don't miss any signals from the universal module, as can happen occasionally with X10, I also use a hacked motion detector to send a wireless signal *directly to my computer* [Hack #83] when the fuel gun starts and stops. The motion detector's dusk sensor is wired to the same relay that's connected to the universal module. When the relay is activated, XTension receives a wireless signal from the motion detector and a power-line signal from the universal module.



Visit Tom Laureanno's web site (<http://www.laureanno.com/x10-mod1.html>) for information about converting a motion detector's dusk sensor to a wired trigger.

Each module is a separate device in XTension. The motion detector is Wireless Pump, and the universal module is Line Pump. The On and Off scripts for each unit turn on a pseudo unit named The Oil Pump and use some logic to determine if an event was missed. Let's look at the On script for the Line Pump unit.

```

if (time delta of "The Oil Pump") > 10 then
  if (status of "The Oil Pump") is true then
    --we missed the last off...
    write log "Missed pump off event, assuming average."
    turnon "The Oil Pump" with no script --keep the wireless event from
doing another missed event.
    set value of "PtimesOff" to (value of "PtimesOff") + 1
    if (time delta of "PmissedOff") > 20 then
      set value of "PmissedOff" to (value of "PmissedOff") + 1
      --If we're going to fudge for the missed event, we have to
      -- fudge the oil totals also. 22.5 seconds per oz...
      set pOil to (value of "PavgOn") / 22.5
    end if
  else
    turnon "The Oil Pump"
  end if
else
  turnon "The Oil Pump"
end if
end if

```

The Wireless Pump unit receives its signal faster because it avoids the transmission delays of power line X10, so this script for Line Pump checks to see if the pump is already on and was turned on in the last 10 seconds. If both

## Track Fuel Consumption

aren't true, the other unit missed the command. Pseudo units that track how many commands are missed are updated, as is `PavgOn`, which tracks consumption, to adjust for the missed event.

A similar method in the `Off` script adjusts for missed `On` commands:

```
if (time delta of "The Oil Pump") > 10 then
  if (status of "The Oil Pump") is true then
    turnoff "The Oil Pump"
  else
    --we missed the last on...assume average
    if (time delta of "PmissedOn") > 20 then --don't duplicate wireless
      device
        write log "Missed pump on event"
        set value of "PtimesOn" to (value of "PtimesOn") + 1
        set value of "PmissedOn" to (value of "PmissedOn") + 1
        set secondsOn to value of "PavgOn"
      end if
    end if
  else
    if (status of "The Oil Pump") is true then
      turnoff "The Oil Pump"
    end if
  end if
end if
```



The scripts that identify and adjust for missed commands are used in both the `Line Pump` and `Wireless Pump` units. Either could miss a command, and having the scripts in both places helps ensure accuracy.

The pseudo unit called `The Oil Pump` does all the calculation of elapsed time and fuel consumed. This is its `On` script:

```
write log "Pump on"
write log "the time delta of the pump is " & (time delta of (thisUnit))
set value of "PtimesOn" to (value of "PtimesOn") + 1
turnon "Attic fan" in 8 * minutes--this fan is turned on when the main
furnace air handling fan comes on.
write log "Attic fan schedules" color green
save database for unit thisUnit
write log "END OF ON SCRIPT"
```

This script increments the `PtimesOn` pseudo unit that's used for statistical tracking. It also schedules a fan in the attic to come on, thanks to an [appliance module \[Hack #3\]](#), to help circulate the furnace's hot air.

The `Off` script for `The Oil Pump` does all the heavy lifting of calculations:

```
set secondsOn to time delta of (thisUnit)
write log "the time delta of the pump is " & (secondsOn)
set value of "PtimesOff" to (value of "PtimesOff") + 1
set tenthsOn to secondsOn / 6
set value of "PavgOn" to (((value of "PdayOn") * 6) / (value of "PtimesOn"))
```

The elapsed time since the unit was turned on, calculated using the time delta provided by XTension, is converted from seconds to minutes. The daily average time turned on (*PavgOn*) is updated using this information.

If the current elapsed time is the highest today so far, *PmaxOn* is updated to set a new high score:

```
if (value of "PmaxOn") < secondsOn then
    set value of "PmaxOn" to secondsOn
end if
```

Likewise, if this is the shorted elapsed time, *PminOn* is updated to reflect the new value:

```
if (value of "PminOn") > secondsOn then
    set value of "PminOn" to secondsOn
end if
```

Total usage for the season is stored as a *unit property*:

```
set t to (Get Unit Property "TimeOn")
Set Unit Property "TimeOn" to (t + secondsOn) -- in seconds
set S to (Get Unit Property "SeasonUsage")
Set Unit Property "SeasonUsage" to round ((S + pOil) * 100) / 100 -- in
ounces of oil
```

This calculation determines the per-hour usage and updates a unit property:

```
set h to (Get Unit Property "HourUsage")
set h to round (h * 100) / 100
Set Unit Property "HourUsage" to round ((h + pOil) * 100) / 100
```

Daily, weekly, and monthly totals are also calculated and stored:

```
set D to (Get Unit Property "DayUsage")
Set Unit Property "DayUsage" to round ((D + pOil) * 100) / 100
set w to (Get Unit Property "WeekUsage")
Set Unit Property "WeekUsage" to round ((w + pOil) * 100) / 100
set M to (Get Unit Property "MonthUsage")
Set Unit Property "MonthUsage" to round ((M + pOil) * 100) / 100
```

Finally, save database for unit causes XTension to write all the unit properties to its database file:

```
save database for unit thisUnit
```

Usually, XTension saves to disk hourly; this statement ensures the latest info is saved in case a problem occurs.

XTension's unit properties provide a handy way to store the calculated values. Table 5-3 provides a comprehensive list, but you don't need to create these in advance. XTension will create them when you store a value.

Table 5-3. XTension's unit properties

Property name	Value (example)
SeasonUsage	94580
DayUsage	0
MonthUsage	657
WeekUsage	0
TimeOn	542221

You might wonder why these values are kept in unit properties, but some (PavgOn, for example) are stored using pseudo units. All values could be stored as unit properties, but properties aren't visible in the Master List and aren't as easily accessible when you access XTension using a web browser [Hack #99]. I like to keep some of the values more accessible so that I can check on them easily and ensure that the system is not missing too many signals or that the average consumptions aren't inordinately high.

Now that the data is being gathered, let's do some analysis. A scheduled script that executes hourly appends the information to a comma-delimited text file:

```

set MyDate to ((month of (current date)) & (day of (current date)) & ":" &
(round (time of (current date)) / hours) as string) as text
set t to (Get Unit Property "TimeOn" from unit "The Oil Pump")
set h to (Get Unit Property "HourUsage" from unit "The Oil Pump")
set D to (Get Unit Property "DayUsage" from unit "The Oil Pump")
set w to (Get Unit Property "WeekUsage" from unit "The Oil Pump")
set M to (Get Unit Property "MonthUsage" from unit "The Oil Pump")
set S to (Get Unit Property "SeasonUsage" from unit "The Oil Pump")
try
--in case the file was left open
  close access alias "G4 X HD:Applications:XTensionFolder:OilUsageGraph.
txt"
on error the error_message number the error_number
end try
set MyFileRef to open for access alias "G4 X HD:Applications:XTensionFolder:
OilUsageGraph.txt" with write permission
set MyData to read MyFileRef
set eof of MyFileRef to 0 - append to end of file
set MyData to ((MyData) & (MyDate & "," & h & "," & D & "," & w & "," & M &
"," & (t / 6) & "," & S) as text) & return
write MyData to MyFileRef
close access MyFileRef

```

You can import this file into a spreadsheet for further analysis or graphing. I use DeltaGraph (<http://www.redrocksw.com/deltagraph/mac/>; \$300). Figure 5-15 shows a typical chart for slightly more than a week's usage.

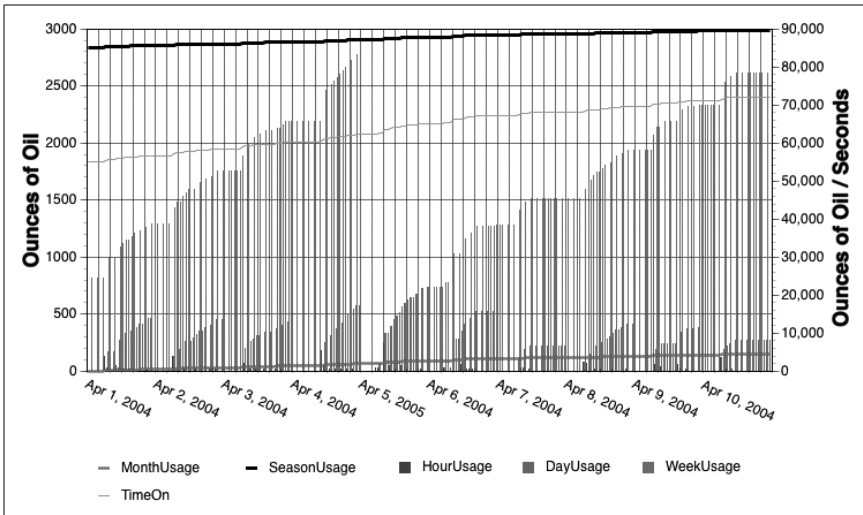


Figure 5-15. Cumulative fuel usage chart

If your system also tracks the current outside temperature [Hack #64], plotting that value against daily usage would be an interesting addition. For more on plotting tools and methods, see “Chart Home Automation Data” [Hack #92].

—Henk van Eeden