# A Guide to HPWHsim

**May 4, 2016**

**Prepared by:**

Nicholas Kvaltine
Michael Logsdon
Ben Larson
Ecotope, Inc.

# Contents

HPWHsim General Overview ..... 3
Limitations ..... 4
HPWHsim Usage ..... 8
Interface ..... 8
Model Specification ..... 9
HPWHsim Tips and Tricks ..... 12
Failure ..... 13
Known Issues ..... 14
Sanden Model ..... 14
UA Losses ..... 14
HPWHsim Technical Explanation ..... 15
Heat Source Choice Logic ..... 17
Heating Logic ..... 19
Definitions ..... 21

# HPWHsim General Overview

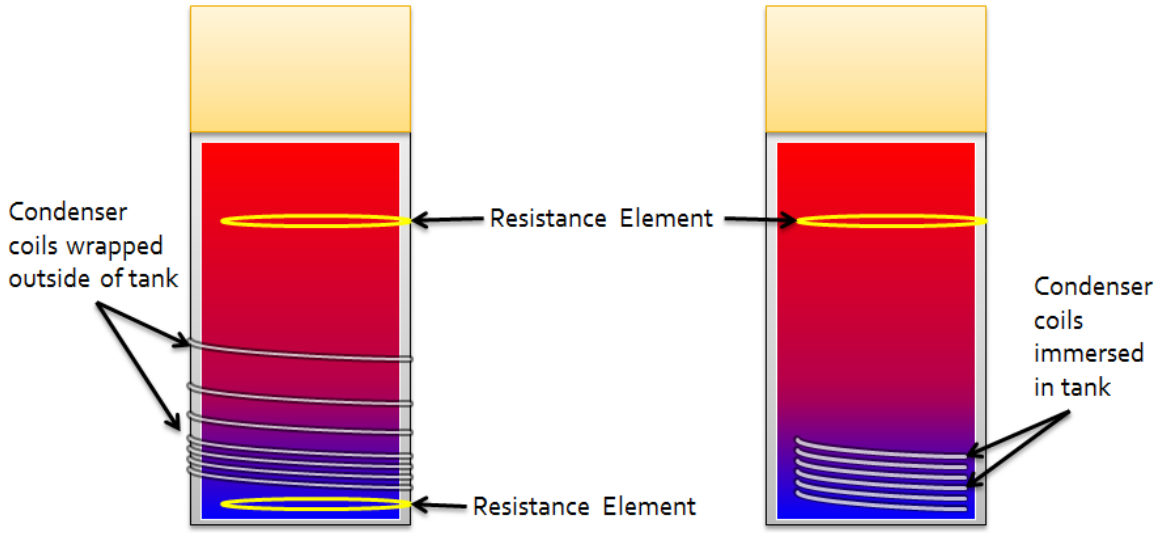
### *Motivations*

Heat Pump Water Heaters (HPWHs) provide an economical way to substantially reduce the amount of electrical energy used in domestic water heating which is the second largest use of electricity in residences.[1,2] It follows that the proper application of HPWHs can potentially have a large impact in reducing energy use. To help understand and estimate the amount of energy savings that can be achieved, a HPWH simulation was developed and calibrated. This HPWH simulation was developed with whole house simulation in mind; it is intended to be run independently of the overarching simulation's time steps, other parameters, and does not aggregate its own outputs. It was also designed to run quickly, as the typical use case would see many simulations run, each a year-long or more.

### *Flexibility*

The simulation was designed to model storage tank water heaters, specifically HPWHs. The configuration of a HPWH varies between different models so the simulation must accommodate these variations. Common variables are the number and position of electric resistance elements, the arrangement of the condensing coils, and the performance of the compressor system, among others. Figure 1 shows a schematic representation of two possible HPWHs. Electric resistance water heaters (ERWH) are a simpler case and can also be modeled with the simulation (imagine Figure 1 without a condenser).

Figure 1. Possible HPWH Configurations




[1] See Energy Information Agency (EIA) Residential Energy Consumption Survey (RECS): http://www.eia.gov/todayinenergy/detail.cfm?id=10271&src=%E2%80%B9%20Consumption%20%20%20%20%20%20Residential%20Energy%20Consumption%20Survey%20(RECS)-b1

[2] See analysis by the Regional Technical Forum showing cost-effective applications for HPWH: http://rtf.nwcouncil.org/measures/measure.asp?id=176
http://rtf.nwcouncil.org/meetings/2015/07/Res_SFandMH_%20HPWH_UES_v2.pptx

To allow for more flexibility in the specification of heat sources, the concept of "condensity" was developed. A portmanteau of "condenser" and "density", the condensity represents the section of the tank in which a heat source will add the heat it generates. A resistance element, for example, would have a condensity concentrated entirely in one node. The condensity is specified by 12 points which should sum to 1. A related factor, the "condentropy" is calculated from the condensity and used along with the condensity and the temperature of the water to calculate the distribution of heat from wrapped condenser coils. Submerged heat sources use a simpler technique, adding heat to all the nodes at or above their level which have the same temperature.

In addition to the physical properties of the HPWH and the topology of its heat sources, each model of HPWH has a unique set of criteria which direct it to engage or disengage its various heat sources. This logic is specifiable as well, by choosing from a set of standard decision criteria, such as the temperature of the top third of the tank, and supplying setpoints for those criteria. Although the simulation does not model interaction with the environment (that is left up to the overarching, calling program), the ability to set up a ducted HPWH is available due to the separate specification of the evaporator temperature and the tank ambient temperature.

## Limitations

As a general comment, the operation of heat pump water heaters, and especially those with resistance elements, is highly complex, interdependent, and non-linear. Subtle changes in seemingly benign inputs like inlet water temperature or ambient air temperature can lead to different heat sources firing at different times, which, especially in the case of resistance elements, can cause dramatically reduced efficiency. For a HPWH trigger happy with its resistance element, the difference between a daily average efficiency of 3 and 1.5 could be an additional gallon during the morning shower. Due to effects such as this, simulating the same draw profile over and over again for an entire year creates an estimate that is extremely fragile and sensitive to operating conditions, such as ambient temperature, inlet water temperature, and setpoint. The solution for overcoming the interdependent complexity of draws, control logic, and operating conditions, is to simulate draw patterns with as much randomness as possible. By varying draw patterns, the internal state of the HPWH can more effectively explore the configuration space it is likely to experience in field operations. It is preferable to either define a unique draw profile for the entire year based on real-life water usage data, or for a more compact representation, repeat a given daily/weekly draw pattern with added stochastic variations. Throw as much randomness as possible at the simulation to generate the most robust possible estimate of real life performance (this is also preferable because, as we know, real life provides an ample quota of randomness).

### *Simulation Limitations*

The requirement for speed and simplicity place several limitations on the simulation.

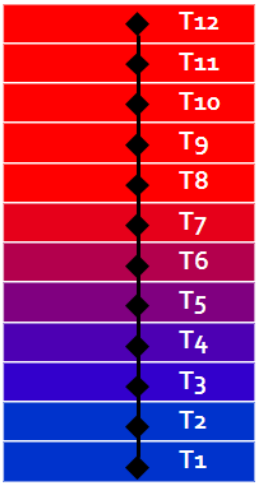
- There is no hydrodynamic simulation done for the water in the tank. Heat transport within the tank is not modeled directly, but is taken into account in the way heat sources add heat to the tank. Thus there is a possibility that the tank could be warmer on the

bottom than the top, an unphysical situation, though this would require unusual circumstances and has not been observed when simulating with realistic data.

- There is a single water supply which enters the tank at the bottom and leaves from the top. This limitation does not affect most water heaters, but could rule out certain combined-use water heaters.

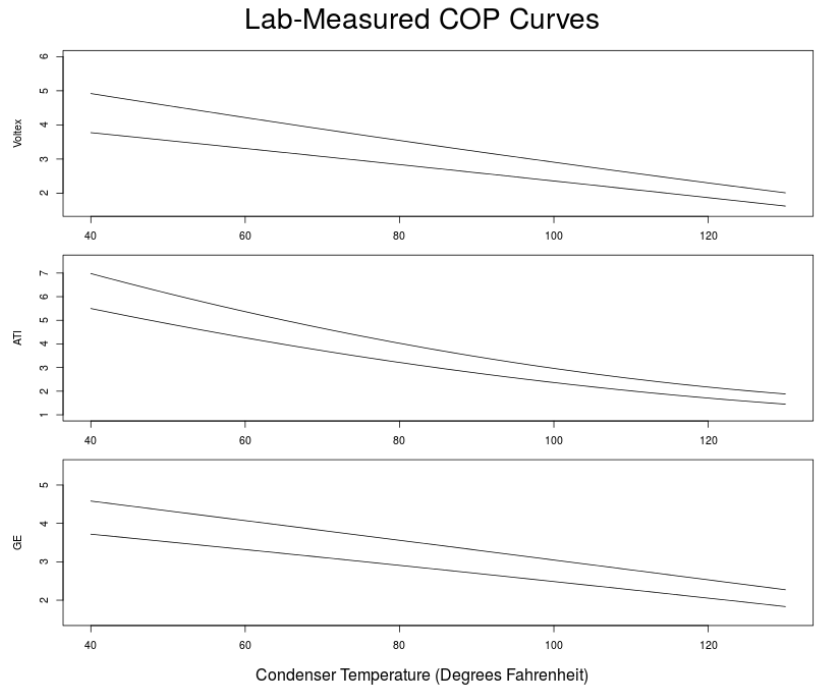
- To simplify water temperature profile calculations, the simulation uses a nodal approximation. That is to say, the tank of water is vertically divided into equal-volume sections, each of which has a temperature. Figure 2 shows an example with 12 nodes, where the color represents temperature. The number of nodes is a variable, however certain calculations within the code require that it be a multiple of 12. For most simulations, 12 nodes is sufficient, however certain models benefit from additional nodes. The reason for this is because any time a partial node volume is drawn from the tank, each node mixes with the node above it. For water heaters that are especially sensitive to tank temperature profile, additional nodes can improve accuracy at the cost of performance.

**Figure 2. The nodal approximation**




- Another approximation used to simplify the calculations are the coefficient of performance (COP) and input power curves. In lieu of simulating a vapor compression cycle from first principles, each model of HPWH is tested in a lab and the input power and COP are characterized as functions of the evaporator and condenser temperatures. It is assumed that these functions are linear in evaporator temperature and at most quadratic in condenser temperature. Example COP curves are shown in Figure 3.

**Figure 3. COP curves for three makes of water heater. Upper curve in each graph is measured at 67 F, lower curve at 47 F**




These linear approximations work quite well for the packaged units with a traditional, single speed, vapor-compression cycle. However, the variable speed Sanden $CO_2$-based system ramps up or down to maintain mostly constant output capacity, which, when combined with an approximately linear relationship between COP and air temperature, leads to a roughly hyperbolic shape of input power with varying air temperature. Allowing a hyperbola for the relationship between input power and air temperature was deemed outside the scope of this version of HPWHsim, although it is a desired upgrade in the future. As such, the Sanden model results are most trustworthy with the evaporator temperature within a range of approximately 30 °F to 80 °F.

- To simplify calculating the standby losses, the total losses are calculated using the average temperature of the tank as a whole. These losses are then equally split amongst all of the nodes. In cases where the bottom of the tank is cold and the top is hot, this can result in a node which is below ambient temperature losing heat. This is an uncommon occurrence though, and the procedure is consistent with the measurement of the UA, which is done for the entire tank.

### *Data Limitations*

There are several key notes as to the applicable ambient temperature ranges for the HPWH models defined in the simulation. These models were largely specified based on lab testing data that was motivated by determining performance in a northwest climate. This led to testing largely at two ambient conditions, 50 °F and 67.5 °F, as well as determining a low temperature compressor cutoff. The results from the simulation are likely accurate in a range of temperatures up to 85 °F, based on the observed linearity with respect to evaporator temperature of previously tested vapor compression cycle based heat pump systems. For more extreme ambient

temperatures, such as 95 °F, as could be found in a garage in a warmer climate, the simulation provides an approximation of the performance, however, as we simply do not currently have the data, we cannot assert how accurate it is or not.

# HPWHsim Usage

## Interface

Using the HPWH simulation in your own C++ code is quite simple. The HPWH.hh header contains all the necessary functions and definitions.

1. First, declare a HPWH object.
2. Use either of the member functions HPWHinit_preset, HPWHinit_file, or HPWHinit_resTank to intialize the hpwh object. Descriptions of these two functions are provided later in this document.
3. Run the simulation. There are two functions used for this. The primary function is runOneStep,
   which takes arguments for the water inlet temperature in Celsius, the volume of water to be withdrawn this step in liters, the ambient temperature in Celsius, the external (evaporator) temperature in Celsius, the Demand Response status, and the length of time in minutes of this step.
   RunNSteps takes an additional argument, the number of steps to run, and calls runOneStep that many times, aggregating the outputs as appropriate and reassigning them to the usual variables.

   Note: Although using runOneStep with 20 minutes per step and runNSteps with 20 steps at 1 minute per step both run 20 minute of simulation, the results will be far different. 1 minute is the recommended length of time per step.

   There is an alternate version of runOneStep that takes four inputs. This version uses stored variables for the inlet temperature and the minutes per step. The stored variables must be set using the setting functions setInletT and setMinutesPerStep; they do not change unless these functions are called.

4. Get the outputs. The hpwh object holds all of the outputs from the step (or steps, for runNsteps) that was run last. These are accessed through the following functions:

   getTankNodeTemp – returns the temp of the specified node
   getNthSimTcouple – the nodes are averaged to result in six equally spaced points, representing the 6 thermocouples used in lab measurements. Returns the Nth thermocouple.

   getNthHeatSourceEnergyInput – returns the energy (electrical) input into the Nth heat source
   getNthHeatSourceEnergyOutput – returns the energy output (heat) from the Nth heat source

   getNthHeatSourceRunTime – returns the length of time the Nth heat source ran last step. Should not be longer than minutes per step, although the sum of all heat source run times can be, due to concurrently running heat sources

isNthHeatSourceRunning – returns whether or not the Nth heat source is still running

getOutletTemp – returns the average temp of outlet water over the last step – 0 for no draws
getEnergyRemovedFromEnvironment – returns the energy removed from the environment
getStandbyLosses – returns the energy lost from the water through the tank walls

Many of these functions have two forms. The first, with fewer arguments (or no arguments), returns the value in the default units, which are metric. The other form takes an argument of type UNITS, which is an enum that specifies which units will be used for the output value.

## Model Specification

HPWHinit_presets allows for a fully specified water heater model to be selected and initialized. The choices are enumerated by the enum MODELS. Currently the available choices are:

These are used for testing and should not necessarily be expected to provide realistic results:
MODELS_restankNoUA, MODELS_restankHugeUA, MODELS_restankRealistic, MODELS_basicIntegrated, MODELS_externalTest

These models are based on real tanks and measured lab data:

// AO Smith models
MODELS_AOSmithPHPT60
MODELS_AOSmithPHPT80
MODELS_AOSmithHPTU50
MODELS_AOSmithHPTU66
MODELS_AOSmithHPTU80

// GE Models
MODELS_GE2012
MODELS_GE2014STDMode
MODELS_GE2014

// Sanden CO2 transcritical heat pump water heaters
MODELS_Sanden40
MODELS_Sanden80

// The new-ish Rheem
MODELS_RheemHB50

// The new-ish Stiebel
MODELS_Stiebel220E

// Generic water heaters, corresponding to the Northern Climate Specification tiers 1, 2, and 3
MODELS_Generic1
MODELS_Generic2
MODELS_Generic3

HPWHinit_file allows for a run time specification of all parameters and values used to run a HPWH. The function takes a filename argument and processes the text in this file to calculate parameters for the HPWH model. The specification is based on keywords and values. As described here, the quoted strings represent the keys, and %d, %f, etc. are the values.

%d – an integer
%f – a decimal number-of-showers
%s – a string for units - gal, L, F, C, kJperHrC
%b – the string “true” or “false”

“#” - if this starts a line, that line is a comment and will be skipped
“” - blank lines will also be skipped
"numNodes %d" – the number of nodes
"volume %d %s" – the tank volume, and its units
"UA %d %s" – the UA of the tank and its units, currently only specifiable as kJperHrC
"depressTemp %b" – whether or not the temperature depression ability should be used
"mixOnDraw %b" – whether or not the bottom of the tank should mix when draws occur
"setpoint %f %s" – the setpoint of the tank
"verbosity %s" – the verbosity; accepts "silent", "reluctant", "typical", or "emetic"
"numHeatSources %d" – the number of heat sources

The following commands apply specifically to heat sources and must be preceded (in the same line) by:
“heatsource %d" where the integer given refers to one of the heat sources.

### *Heat Source Parameters*

"isVIP %b" – if the heat source is a VIP or not
"isOn %b" – if the heat source is currently running
"type %s" – the type of the heat source: "resistor" or "compressor"
"coilConfig %s" – the configuration of the heat source's coils: "wrapped", "submerged", or "external"
"condensity %d %d %d %d %d %d %d %d %d %d %d %d" – the twelve values of the condensity
"T1 %d %s" – the first temperature for the heat source's COP and input power curves, and the units
"T2 %d %s" – the second temperature for the heat source's COP and input power curves, and the units

The following are the coefficients for the input power and COP curves – units are not specifiable
"inPowT1const %f"
"inPowT1lin %f"
"inPowT1quad %f"
"inPowT2const %f"
"inPowT2lin %f"
"inPowT2quad %f"
"copT1const %f"
"copT1lin %f"
"copT1quad %f"
"copT2const %f"
"copT2lin %f"
"copT2quad %f"

"hysteresis %f %s" – the hysteresis (see Tips and Tricks) and units
"backupSource %d" – the number of the heat source that is the backup to this heat source

"onlogic %s %f %s" – a logical specifier for heat source engaging conditions. The first string chooses the logic and the number chooses the setpoint, followed by its units. Currently available on-logics are: "topThird", "bottomThird", and "standby"
**Example**: "heatsource 1 onlogic bottomThird 40 F"

"offlogic %s %f %s" – a logical specified for heat source turn-off conditions. The first string chooses the logic and the number chooses the setpoint, followed by its units. Currently available off-logics are: "lowT", "lowTreheat", "bottomNodeMaxTemp", "bottomTwelthMaxTemp", "largeDraw"
**Example**: "heatsource 1 offlogic largeDraw 66 F"

HPWHinit_resTank is used to specify a fully resistive water heater. Since resistance tanks are so simple, they can be specified with only four variables: tank volume, Energy Factor, and the power of the upper and lower elements. Energy Factor is converted into UA internally, although an external setter for UA is also provided in case the Energy Factor is unknown.

Several assumptions regarding the tank configuration are assumed: the lower element is at the bottom, the upper element is at the top third. The logics are also set to standard settings, with upper as VIP activating when the top third is too cold. The performance of a resistance tank is not particularly sensitive to the logic values, so the standard set should simulate actual usage well.

The default resistance tank can be specified by calling HPWHinit_resTank with no parameters. This will instantiate a 47.5 gallon (50 gallon nominal) water heater with an Energy Factor of 0.95 and elements with 4500 Watts of power capacity.

## HPWHsim Tips and Tricks

### *Units*

Most variables used in HPWHsim have the units they are specified in appended with an underscore. For example, the tank temperatures use the variable named tankTemps_C, which implies they should be supplied in degrees Celsius. There are functions for converting between units, and many of the output or input functions have versions which take a UNITS input. The functions with this input will take the values specified and, if necessary, convert them to the appropriate units.

### *Interior space temperature depression, for single zone models*

The predecessor of HPWHsim was originally written for SEEM, Ecotope's single-zone building energy use simulation engine. Because HPWHs remove energy from the air, there is the possibility that they will depress the temperature of their local environment, thus decreasing their performance. This effect was measured in a field study, with the average temperature depression being approximately 4.5 F with a half life of 9.4 minutes. It works by exponentially decreasing the local (ambient and evaporator) temperature of the HPWH to a fixed level below the actual input ambient temperature when a heat source which has the "depressesTemperature" boolean set to true is running. When no qualifying heat source is running, the local temperature exponentially returns to ambient using the same time constant.

### *The verbosity and callback function*

The HPWHsim provides an error/debugging message facility, along with the ability to specify where the output should be sent, via a callback function. There are four levels of verbosity for the messages:
VRB_silent = 0, VRB_reluctant = 1, VRB_typical = 2, and VRB_emetic = 3
They are ordered by the amount of output they will write: "silent" prints no messages, "reluctant" prints messages only for fatal errors, "typical" prints basic debugging information, and "emetic" prints copious amounts of information. Each level prints all the messages from that below it, i.e. "typical" will print all of the "reluctant" messages and all of its messages. When initialized, the HPWH object defaults to silent, but can be set using the setVerbosity function.

The default behavior is for output to be written to stdout, using cout. If it is desired for the output to be sent to a different place, this can be done by providing a function pointer and a context pointer. These variables are stored as part of the HPWH object and set using the setMessageCallback function:

```
void setMessageCallback( void (*callbackFunc)(const std::string message, void* pContext),
void* pContext);
```

As can be seen, the callback function should return nothing, and take a string and a pointer as arguments.

### *Hysteresis*

One of the parameters for each HeatSource is a term labeled "hysteresis_dC". This represents

the different behavior that has been observed for heat sources which are running. The units, dC, are differential Celsius degrees, i.e. the conversion between C and F should not use the constant offset term (32 F). It is applied to:

OFFLOGIC_lowT - when the heat source is engaged, the hysteresis is subtracted from the decisionPoint. If the heat source is not engaged, the decisionPoint is used as is.

OFFLOGIC_lowTreheat – when the heat source is engaged, the hysteresis is added to the decisionPoint. The lowTreheat logic does nothing when the heat source is not engaged, as it is not intended to prevent a heat source from engaging.

#### *resetTankToSetpoint*

The resetTankToSetpoint function will set all nodes to the setpoint. The starting temperature of a HPWH is set to the setpoint when initialized. Any change in the setpoint will not alter the current temperature of the tank however, so if a non-default setpoint is desired, the resetTankToSetpoint function can be used.

## Failure

### *Fatal Errors*

There are a number of reasons that the HPWH simulation can fail:

- The draw size in one step is more than the volume of the tank.
- A HeatSource which has defined backupHeatSource or companionHeatSource pointers is copied or assigned.
- HPWHinit_file or HPWHinit_presets fails to properly initialize the HPWH.
- A value other than 1 for minutesPerStep is specified when attempting to run with tempDepression.

This kind of fatal error will write out an informative error message if the verbosity is not set to "silent" and will return HPWH_ABORT.
The simulation is unrecoverable from this kind of error, and must be reinitialized. Care should be taken not to invoke one of these failure conditions again.

### *Non-Fatal Errors*

Certain kinds of mistakes will cause errors but not prevent the simulation from continuing to run. In these cases, the function which has failed will return a failure code specified in the header as HPWH_ABORT. Examples of these failures are the functions which set or get from a particular node or heat source. Specifying a number, N, outside the range of nodes or heatsources (e.g. 13 for a HPWH with 12 nodes) will return HPWH_ABORT. Typically a failure of this kind will output a useful error message, provided the verbosity is set to at least "reluctant".

# Known Issues

## Sanden Model

The model for the Sanden HPWH will have accurate run times only within a limited range of ambient temperatures. The useful range is estimated to cover 35 to 80 °F, however, deviations from expected values were not tested. The cause of this problem is the assumption of linear input power response to ambient temperature. The Sanden unit has internal logic that attempts to maintain constant output capacity. COP, as measured in the lab, retains a linear response to evaporator temperature, which implies that the input power, which is the capacity divided by the COP, must be non-linear. This is observed in the lab, with input power rising dramatically as evaporator temperatures drop below 35 °F.

Since the COP response to ambient temperature is linear, our model still works and should give accurate COP estimates as well as total energy in most cases. The further conditions deviate from the optimal range the more likely it is that heating will not finish before the next draw cycle, which can have a larger impact in COP and energy use. The error in the input power model, however, will cause the duration of compressor operation to be inaccurate, and consequently errors relating to the time of day of operation will develop. It is recommended that the current Sanden model be used with caution if at all.

## UA Losses

The way the UA losses are calculated (see the Technical Explanation) can result in violation of entropy. Since the total UA loss is based on average tank temperature and divided equally, nodes that are colder than the ambient temperature will lose heat if the average tank temperature is above ambient. Although this is counterintuitive, it does not cause any serious discrepancy with measured data. Partially this is due to the relative rarity of this occurrence, and partially it is due to the fact that the total UA used for each model is measured. This means that the heat loss rate should be accurate, although the distribution of tank temperatures may be off.

# HPWHsim Technical Explanation

### *HPWHinit_resTank*

The UA is calculated from a given Energy Factor using the following equation[3]:

$$UA = \frac{\frac{1}{\mathrm{EF}} + \frac{1}{\mathrm{RE}}}{67.5 \cdot \left(\frac{24}{41094} - \frac{1}{\mathrm{RE} \cdot \mathrm{Power}}\right)} \quad (1)$$

Where EF is the Energy Factor, RE is the Recovery Efficiency, set to 0.98, and Power is the power of the lower element in BTU per Hour.

### *Update Tank Temperatures*

The temperatures of the nodes in each tank are updated at the beginning of each step. The temperature profile changes due to draws and UA losses are calculated in this step.

The fractional number of nodes drawn is calculated as shown below:

$$drawFraction = \frac{volumeofdraw}{volumeperNode} \quad (2)$$

The draw fraction is then divided into its whole part and its fractional part. In a typical HPWH, with a volume per node of approximately 4 gallons, most draws will be less than one node. The whole nodes are "moved upwards" in the tank by assigning the temperature of each node to the node above it and filling in the bottom node with water at the specified inlet temperature. Then the remaining partial node draw is calculated using the following:

$$tankTemps[i] = tankTemps[i] * (1 - fractionofnode) + tankTemps[i-1] * fractionofnode \quad (3)$$

Where *i* is the number of the node in the node array. The bottom node, where *i - 1* is negative, is mixed with water at the inlet temperature. The correctness of this calculation relies on the relative invariance of the specific heat of water with respect to temperature and the equality of node volumes.

The temperature and volume of any water shifted out of the tank is tracked and averaged to calculate the outlet temperature.

If the "mix on draw" feature is enabled for this HPWH, the bottom third of the tank experiences mixing. This is done by finding the average temperature of the bottom third. Then for each node, one third of the difference between the average and the current temperature is subtracted:

[3] Lutz, James D., et. al. WHAM: Simplified Tool for Calculating Water Heater Energy Use, 1999.

$$tankTemps[i] = tankTemps[i] + ((average - tankTemps[i])/3.0); \quad (4)$$

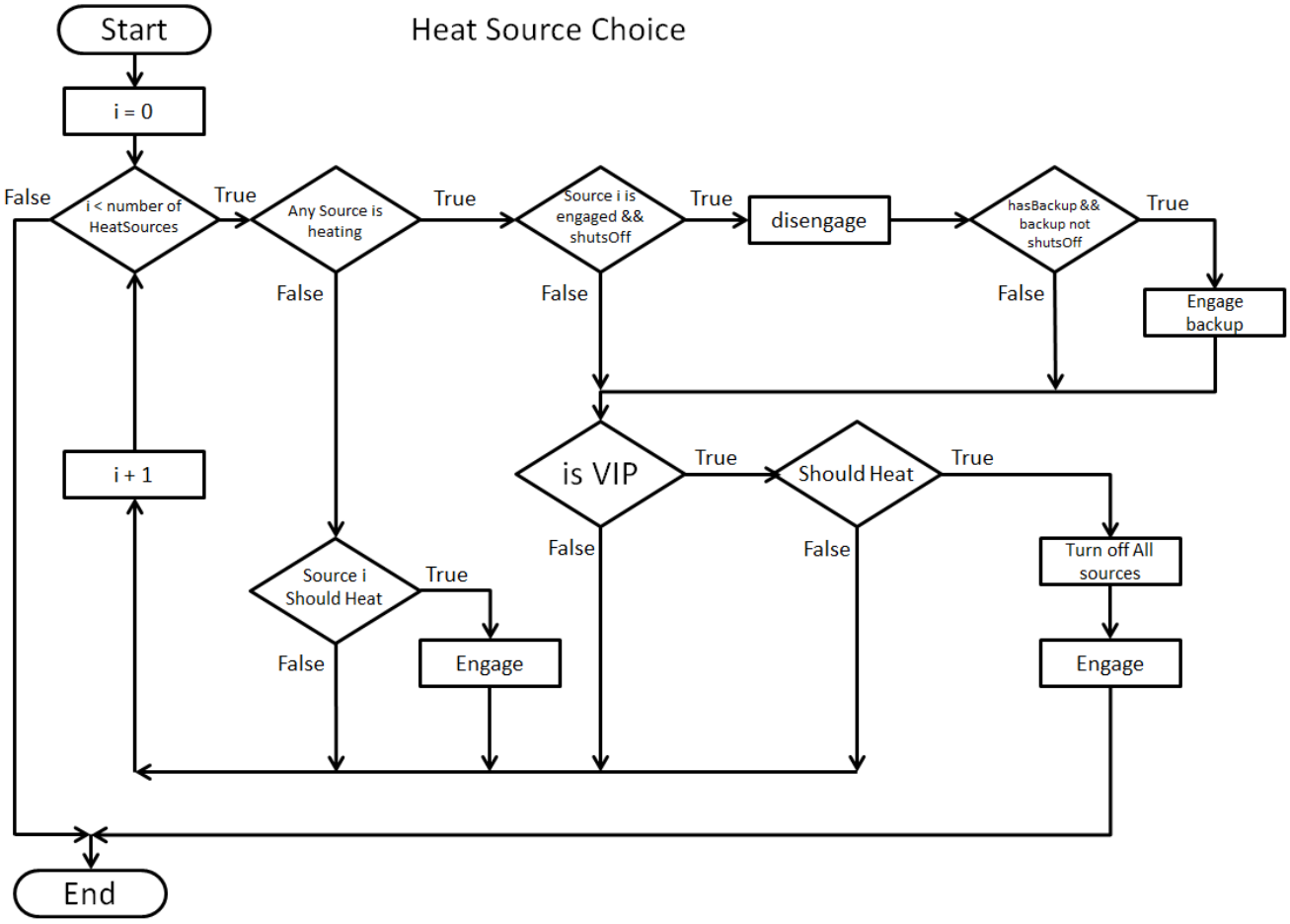
Next, the temperatures of the nodes are adjusted for standby losses. The average temperature of the tank is found, then the total energy lost by the tank is calculated by:

$$\begin{aligned}&\big(tankUA * (averagetanktemperature - ambienttemperature)\\&\qquad * (minutesper\,step/60.0)\big)\end{aligned} \quad (5)$$

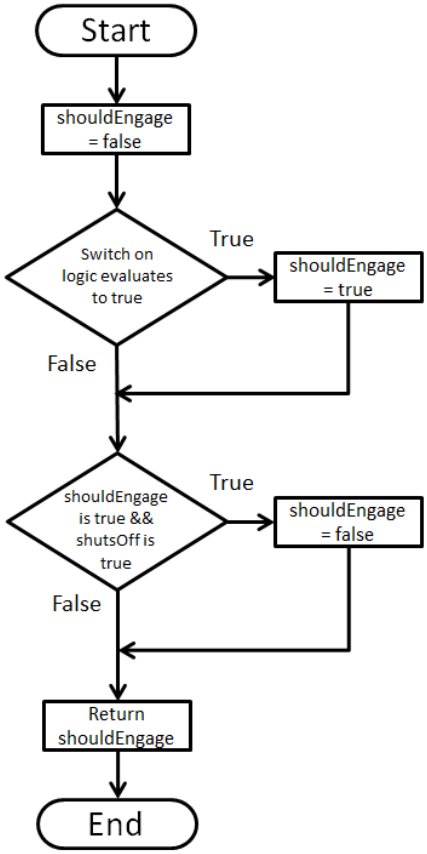
Where *tankUA* is measured kJ / hrC. The total losses are split equally amongst all nodes, and then the effect on temperature is calculated by:

$$nodetemperatureloss = \frac{nodeenergyloss}{\big((volumepernode * densityofwater) * specificheatofwater\big)} \quad (6)$$

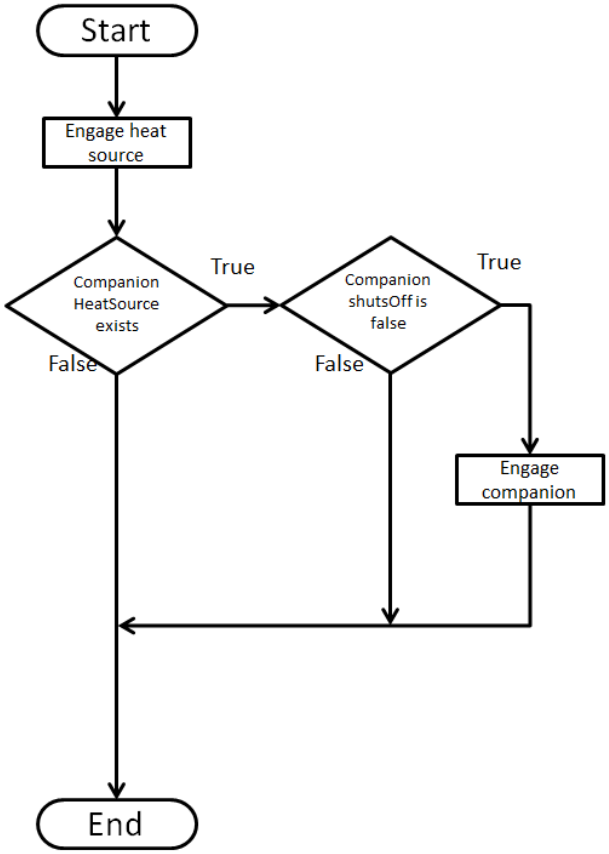
## Heat Source Choice Logic



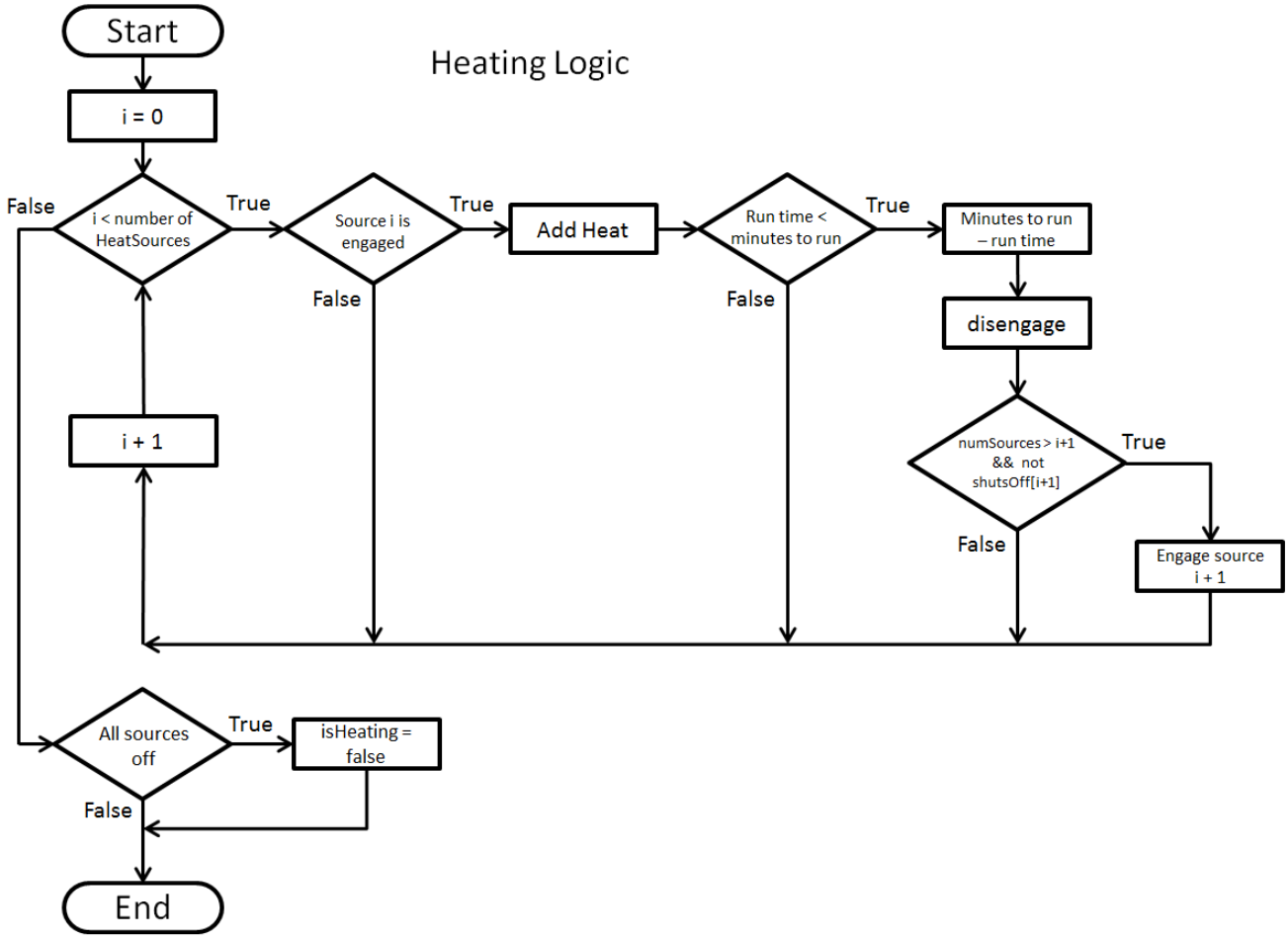
Should Heat




Engage Heat Source



## Heating Logic




The addHeat function begins by choosing between two paths to take, one for "submerged" and "wrapped" configurations, and the second for "external" configurations. The first path proceeds as follows:

Each node is assigned a weighting that will determine what fraction of heat will be assigned to that node. Weightings less than a threshold, currently set to 0.001, are set to zero in order to reduce the computational load.

The capacity of the heat source is calculated using different methods for "submerged" or "wrapped" configurations. In common for both, no heat is added below the lowest node where the heat source has non-zero condensity. For "submerged" configurations, the distribution is the same as the condensity (the number of nodes must be a multiple of 12 so that the condensity divides it evenly). The case for "wrapped" configurations is more complicated. The *i*th node heatDistribution is found by:

$$\frac{(setpoint - temp_i) * 1}{1 + e^{(((temp_i - temp_{lowestNode})/shrinkage) - offset)}} \quad (7)$$

Where *shrinkage* is a parameter derived from the condensity by Equation (8) and *offset* is 5/1.8.

$$alpha + beta * \sum -condensity_i * log(condensity_i) \quad (8)$$

Where *alpha* and *beta* are 1 and 2 respectively.

The heat is added, starting from the top node and moving down node by node. The amount of heat to be added to a node is defined by Equation (9).

$$capacity = (capacityperhour * (minutesToRun/60.0) * heatDistribution_i) \quad (9)$$

Where the *i* subscript denotes the *i*th node.

This heat is combined with any heat leftover from the last step and used as input to the addHeatAboveNode function. This function starts with the given node and searches upwards for a node that is not at the same temperature. The temperature at that node becomes the target temperature, and all nodes below it receive heat until all the heat is used, or until all the nodes are at the target temperature. Then the target temperature becomes the temperature of the next node up. If there is no node further up, the target temperature becomes the setpoint. If all nodes above the given node are at setpoint, any remaining heat is returned to be used for the next node down.

For the "external" configuration, the addition of heat is simpler. The capacity is determined and adjusted for the amount of time remaining in the step. The fraction of the bottom node that can be heated with the capacity is calculated. If it is greater than one all nodes are shifted down and a node at setpoint is added at the top. The amount of time run to create that amount of heat is calculated and subtracted from the available time, and then, if a shutOff logic is not achieved, the process is repeated. If the fraction of the bottom node that can be heated in the remaining time is less than one, all nodes are shifted down by that fraction, mixing in the same way as Equation (3).

## Definitions

**HPWH** (pronounced 'hup-wuh') – Heat Pump Water Heater

**ERWH** (pronounced 'err-wuh') **–** Electric Resistance Water Heater

**Heat Source –** A heat source is a component of a HPWH model that has the ability to add heat to a tank. A HPWH can have an arbitrary number of heat sources, though most have between 1 and 3, inclusive. An ERWH typically has 2. A heat source has its own set of properties including, but not limited to, condensity, turn on and turn off logic, and parameters that describe its energy performance.

**Type of Heat Source** – Currently allowing only "resistor" and "compressor", this setting is intended to allow for the aggregation of total compressor or resistance energy usage. This is a commonly sought quantity for comparison of HPWH's, particularly when simulating yearly timespans.
One caveat: on file input, heat sources of type "compressor" will set the "depressesTemperature" boolean to true. If this particular heat source is a compressor but should not depress temperature, depressesTemperature should be set to false after setting the type; however, typically doTempDepression would be set to False if no temperature depression was intended to be done at all.

**depressesTemperature –** This boolean specifies whether the heat source should cause the temperature to be depressed when it is running. There is no effect when the "doTempDepression" member of the parent HPWH is set to false.

**doTempDepression -** This boolean specifies whether the HPWH should utilize the temperature depression feature. This feature is intended for use with single zone house simulations. It is intended to model the effect of a HPWH cooling the space where it is located. It works by exponentially decreasing the local (ambient and evaporator) temperature of the HPWH to a fixed level below the actual input ambient temperature when a heat source, which has the "depressesTemperature" boolean set to true, is running. When no qualifying heat source is running, the local temperature exponentially returns to ambient using the same time constant. Currently, the total temperature drop is 4.5 F, and the half-life is 9.4 minutes.

**decisionPoint -** This is the value used for the heatSource logics. Each logic represents a condition which may occur based on some value, e.g. if the top third of the tank rises above some temperature. The decisionPoint is the value that is specified.

**Verbosity –** This is the amount of output that will be provided. It has several levels and is discussed in more depth in the "Tips and Tricks" document.

**HPWH_ABORT –** an error code specified in HPWH.hh which signifies that an error has occurred in the function which returned this value

**Condensity –** a portmanteau of "condenser" and "density", the condensity is a 12-node model

that represents how the condensing coils come in contact with the tank. It is explicitly limited to 12 nodes, and each value is expressed as a fraction (so that the condensity sums to 1). For example, a resistance element would have a condensity such as (0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0) whereas a compressor might be something like (0, 0, 0, 0, 0, 0, 0.2, 0.2, 0.2, 0.2, 0.2, 0).

**Condentropy –** a portmanteau of "condensity" and "entropy", the condentropy is a derived value based on the condensity. It is not specified separately, but instead represents in some way how spread out the condensity is. This effects the way heat is distributed when the heat source is running.