

HPWHsim

Generated by Doxygen 1.8.6

Fri Feb 12 2016 17:14:41

Contents

1	Class Index	1
1.1	Class List	1
2	Class Documentation	3
2.1	HPWH::HeatSource Class Reference	3
2.1.1	Constructor & Destructor Documentation	3
2.1.1.1	HeatSource	3
2.1.2	Member Function Documentation	3
2.1.2.1	addHeat	3
2.1.2.2	disengageHeatSource	4
2.1.2.3	engageHeatSource	4
2.1.2.4	isEngaged	4
2.1.2.5	setCondensity	4
2.1.2.6	setupAsResistiveElement	4
2.1.2.7	shouldHeat	4
2.1.2.8	shutsOff	4
2.2	HPWH Class Reference	4
2.2.1	Member Enumeration Documentation	6
2.2.1.1	DRMODES	6
2.2.1.2	HEATSOURCE_TYPE	6
2.2.1.3	MODELS	6
2.2.1.4	UNITS	6
2.2.1.5	VERBOSITY	7
2.2.2	Constructor & Destructor Documentation	7
2.2.2.1	HPWH	7
2.2.2.2	HPWH	7
2.2.2.3	~HPWH	7
2.2.3	Member Function Documentation	7
2.2.3.1	getEnergyRemovedFromEnvironment	7
2.2.3.2	getNthHeatSourceEnergyInput	7
2.2.3.3	getNthHeatSourceEnergyInput	7

2.2.3.4	getNthHeatSourceEnergyOutput	7
2.2.3.5	getNthHeatSourceEnergyOutput	8
2.2.3.6	getNthHeatSourceRunTime	8
2.2.3.7	getNthHeatSourceType	8
2.2.3.8	getNthSimTcouple	8
2.2.3.9	getNthSimTcouple	8
2.2.3.10	getNumHeatSources	8
2.2.3.11	getNumNodes	8
2.2.3.12	getOutletTemp	8
2.2.3.13	getStandbyLosses	8
2.2.3.14	getTankNodeTemp	8
2.2.3.15	getTankNodeTemp	9
2.2.3.16	HPWHinit_file	9
2.2.3.17	HPWHinit_presets	9
2.2.3.18	isNthHeatSourceRunning	9
2.2.3.19	operator=	9
2.2.3.20	printHeatSourceInfo	9
2.2.3.21	printTankTemps	9
2.2.3.22	resetTankToSetpoint	9
2.2.3.23	runNSteps	10
2.2.3.24	runOneStep	10
2.2.3.25	runOneStep	10
2.2.3.26	setInletT	10
2.2.3.27	setMessageCallback	10
2.2.3.28	setSetpoint	10
2.2.3.29	setSetpoint	10
2.2.3.30	setVerbosity	10
2.2.3.31	WriteCSVRow	10
2.2.4	Member Data Documentation	11
2.2.4.1	HPWH_ABORT	11

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

HPWH::HeatSource	3
HPWH	4

Chapter 2

Class Documentation

2.1 HPWH::HeatSource Class Reference

Public Member Functions

- [HeatSource](#) ([HPWH](#) *parentHPWH)
- **HeatSource** (const [HeatSource](#) &hSource)
- [HeatSource](#) & **operator=** (const [HeatSource](#) &hSource)
copy constructor
- void [setupAsResistiveElement](#) (int node, double Watts)
assignment operator
- bool [isEngaged](#) () const
- void [engageHeatSource](#) (double heatSourceAmbientT_C)
- void [disengageHeatSource](#) ()
- bool [shouldHeat](#) (double heatSourceAmbientT_C) const
- bool [shutsOff](#) (double heatSourceAmbientT_C) const
- void **addHeat_temp** (double externalT_C, double minutesPerStep)
- void [addHeat](#) (double externalT_C, double minutesToRun)
- void [setCondensity](#) (double cnd1, double cnd2, double cnd3, double cnd4, double cnd5, double cnd6, double cnd7, double cnd8, double cnd9, double cnd10, double cnd11, double cnd12)

Friends

- class **HPWH**

2.1.1 Constructor & Destructor Documentation

2.1.1.1 HPWH::HeatSource::HeatSource ([HPWH](#) * *parentHPWH*)

default constructor, does not create a useful [HeatSource](#) constructor assigns a pointer to the hpwh that owns this heat source

2.1.2 Member Function Documentation

2.1.2.1 void HPWH::HeatSource::addHeat (double *externalT_C*, double *minutesToRun*)

adds heat to the hpwh - this is the function that interprets the various configurations (internal/external, resistance/heat pump) to add heat

2.1.2.2 void HPWH::HeatSource::disengageHeatSource ()

turn heat source off, i.e. set isEngaged to FALSE

2.1.2.3 void HPWH::HeatSource::engageHeatSource (double *heatSourceAmbientT_C*)

turn heat source on, i.e. set isEngaged to TRUE

2.1.2.4 bool HPWH::HeatSource::isEngaged () const

return whether or not the heat source is engaged

2.1.2.5 void HPWH::HeatSource::setCondensity (double *cnd1*, double *cnd2*, double *cnd3*, double *cnd4*, double *cnd5*, double *cnd6*, double *cnd7*, double *cnd8*, double *cnd9*, double *cnd10*, double *cnd11*, double *cnd12*)

a function to set the condensity values, it pretties up the init funcs.

2.1.2.6 void HPWH::HeatSource::setupAsResistiveElement (int *node*, double *Watts*)

assignment operator

< the copy constructor and assignment operator basically just checks if there are backup/companion pointers - these can't be copied configure the heat source to be a resistive element, positioned at the specified node, with the specified power in watts

2.1.2.7 bool HPWH::HeatSource::shouldHeat (double *heatSourceAmbientT_C*) const

queries the heat source as to whether or not it should turn on

2.1.2.8 bool HPWH::HeatSource::shutsOff (double *heatSourceAmbientT_C*) const

queries the heat source whether should shut off (typically lowT shutoff)

The documentation for this class was generated from the following files:

- /home/nkvaltine/nkvaltine/Projects/HPWHsim/dev/HPWH.hh
- /home/nkvaltine/nkvaltine/Projects/HPWHsim/dev/HPWH.cc

2.2 HPWH Class Reference

Classes

- class [HeatSource](#)

Public Types

- enum [DRMODES](#) { [DR_BLOCK](#) = 0, [DR_ALLOW](#) = 1, [DR_ENGAGE](#) = 2 }
- enum [MODELS](#) { [MODELS_restankNoUA](#) = 1, [MODELS_restankHugeUA](#) = 2, [MODELS_restankRealistic](#) = 3, [MODELS_basicIntegrated](#) = 4, [MODELS_externalTest](#) = 5, [MODELS_Voltex60](#) = 102, [MODELS_Voltex80](#) = 103, [MODELS_GEGeospring](#) =

- ```
104,
MODELS_SandenGES = 110, MODELS_SandenGAU = 111 }
• enum VERBOSITY { VRB_silent = 0, VRB_reluctant = 1, VRB_typical = 2, VRB_emetic = 3 }
• enum UNITS {
 UNITS_C, UNITS_F, UNITS_KWH, UNITS_BTU,
 UNITS_KJ }
• enum HEATSOURCE_TYPE { TYPE_none, TYPE_resistance, TYPE_compressor }
```

## Public Member Functions

- [HPWH](#) ()
- [HPWH](#) (const [HPWH](#) &hpwh)
- [HPWH](#) & [operator=](#) (const [HPWH](#) &hpwh)
- [~HPWH](#) ()
- int [HPWHinit\\_presets](#) ([MODELS](#) presetNum)
- int [HPWHinit\\_file](#) (std::string configFile)
- int [runOneStep](#) (double inletT\_C, double drawVolume\_L, double ambientT\_C, double externalT\_C, [DRMODES](#) DRstatus, double minutesPerStep)
- int [runNSteps](#) (int N, double \*inletT\_C, double \*drawVolume\_L, double \*tankAmbientT\_C, double \*heatSourceAmbientT\_C, [DRMODES](#) \*DRstatus, double minutesPerStep)
- void [setVerbosity](#) ([VERBOSITY](#) hpwhVrb)
- void [setMessageCallback](#) (void(\*callbackFunc)(const std::string message, void \*pContext), void \*pContext)
- void [printHeatSourceInfo](#) ()
- void [printTankTemps](#) ()
- int [WriteCSVHeading](#) (FILE \*outFILE, const char \*preamble="") const
- int [WriteCSVRow](#) (FILE \*outFILE, const char \*preamble="") const
- int [setSetpoint](#) (double newSetpoint)
- int [setSetpoint](#) (double newSetpoint, [UNITS](#) units)
- int [resetTankToSetpoint](#) ()
- int [getNumNodes](#) () const
- double [getTankNodeTemp](#) (int nodeNum) const
- double [getTankNodeTemp](#) (int nodeNum, [UNITS](#) units) const
- double [getNthSimTcouple](#) (int N) const
- double [getNthSimTcouple](#) (int N, [UNITS](#) units) const
- int [getNumHeatSources](#) () const
- double [getNthHeatSourceEnergyInput](#) (int N) const
- double [getNthHeatSourceEnergyInput](#) (int N, [UNITS](#) units) const
- double [getNthHeatSourceEnergyOutput](#) (int N) const
- double [getNthHeatSourceEnergyOutput](#) (int N, [UNITS](#) units) const
- double [getNthHeatSourceRunTime](#) (int N) const
- int [isNthHeatSourceRunning](#) (int N) const
- [HEATSOURCE\\_TYPE](#) [getNthHeatSourceType](#) (int N) const
- double [getOutletTemp](#) () const
- double [getOutletTemp](#) ([UNITS](#) units) const
- double [getEnergyRemovedFromEnvironment](#) () const
- double [getEnergyRemovedFromEnvironment](#) ([UNITS](#) units) const
- double [getStandbyLosses](#) () const
- double [getStandbyLosses](#) ([UNITS](#) units) const
- int [runOneStep](#) (double drawVolume\_L, double ambientT\_C, double externalT\_C, [DRMODES](#) DRstatus)
- void [setInletT](#) (double newInletT\_C)
- void [setMinutesPerStep](#) (double newMinutesPerStep)

## Static Public Attributes

- static const int [HPWH\\_ABORT](#) = -274000

## 2.2.1 Member Enumeration Documentation

### 2.2.1.1 enum HPWH::DRMODES

specifies the various modes for the Demand Response (DR) abilities values may vary - names should be used

Enumerator

**DR\_BLOCK** this mode prohibits the elements from engaging and turns off any currently running

**DR\_ALLOW** this mode allows the water heater to run normally

**DR\_ENGAGE** this mode forces an element to turn on

### 2.2.1.2 enum HPWH::HEATSOURCE\_TYPE

specifies the type of heat source

Enumerator

**TYPE\_none** a default to check to make sure it's been set

**TYPE\_resistance** a resistance element

**TYPE\_compressor** a vapor cycle compressor

### 2.2.1.3 enum HPWH::MODELS

specifies the allowable preset [HPWH](#) models values may vary - names should be used

Enumerator

**MODELS\_restankNoUA** a simple resistance tank, but with no tank losses

**MODELS\_restankHugeUA** a simple resistance tank, but with very large tank losses

**MODELS\_restankRealistic** a more-or-less realistic resistance tank

**MODELS\_basicIntegrated** a standard integrated [HPWH](#)

**MODELS\_externalTest** a single compressor tank, using "external" topology

**MODELS\_Voltex60** this is the Ecotope model for the 60 gallon Voltex [HPWH](#)

**MODELS\_Voltex80** Voltex 80 gallon tank

**MODELS\_GEGeospring** Original GE

**MODELS\_SandenGES** Sanden 40 gallon CO2 external heat pump

**MODELS\_SandenGAU** Sanden 80 gallon CO2 external heat pump

### 2.2.1.4 enum HPWH::UNITS

Enumerator

**UNITS\_C** celsius

**UNITS\_F** fahrenheit

**UNITS\_KWH** kilowatt hours

**UNITS\_BTU** british thermal units

**UNITS\_KJ** kilojoules

## 2.2.1.5 enum HPWH::VERBOSITY

specifies the modes for writing output the specified values are used for  $\geq$  comparisons, so the numerical order is relevant

Enumerator

**VRB\_silent** print no outputs

**VRB\_reluctant** print only outputs for fatal errors

**VRB\_typical** print some basic debugging info

**VRB\_emetic** print all the things

## 2.2.2 Constructor &amp; Destructor Documentation

## 2.2.2.1 HPWH::HPWH ( )

default constructor

## 2.2.2.2 HPWH::HPWH ( const HPWH &amp; hpwh )

copy constructor

## 2.2.2.3 HPWH::~~HPWH ( )

destructor just a couple dynamic arrays to destroy - could be replaced by vectors eventually?

## 2.2.3 Member Function Documentation

## 2.2.3.1 double HPWH::getEnergyRemovedFromEnvironment ( UNITS units ) const

get the total energy removed from the environment by all heat sources in specified units (not net energy - does not include standby) moving heat from the space to the water is the positive direction returns HPWH\_ABORT for incorrect units

## 2.2.3.2 double HPWH::getNthHeatSourceEnergyInput ( int N ) const

Parameters

|          |                   |
|----------|-------------------|
| <i>N</i> | default units kWh |
|----------|-------------------|

## 2.2.3.3 double HPWH::getNthHeatSourceEnergyInput ( int N, UNITS units ) const

returns the energy input to the Nth heat source, with the specified units energy used by the heat source is positive - should always be positive returns HPWH\_ABORT for N out of bounds or incorrect units

## 2.2.3.4 double HPWH::getNthHeatSourceEnergyOutput ( int N ) const

## Parameters

|          |                   |
|----------|-------------------|
| <i>N</i> | default units kWh |
|----------|-------------------|

**2.2.3.5 double HPWH::getNthHeatSourceEnergyOutput ( int *N*, UNITS *units* ) const**

returns the energy output from the Nth heat source, with the specified units energy put into the water is positive - should always be positive returns HPWH\_ABORT for N out of bounds or incorrect units

**2.2.3.6 double HPWH::getNthHeatSourceRunTime ( int *N* ) const**

returns the run time for the Nth heat source, in minutes note: they may sum to more than 1 time step for concurrently running heat sources returns HPWH\_ABORT for N out of bounds

**2.2.3.7 HPWH::HEATSOURCE\_TYPE HPWH::getNthHeatSourceType ( int *N* ) const**

returns the enum value for what type of heat source the Nth heat source is

**2.2.3.8 double HPWH::getNthSimTcouple ( int *N* ) const**

## Parameters

|          |                 |
|----------|-----------------|
| <i>N</i> | default units C |
|----------|-----------------|

**2.2.3.9 double HPWH::getNthSimTcouple ( int *N*, UNITS *units* ) const**

returns the temperature from a set of 6 virtual "thermocouples", which are constructed from the node temperature array. Specify t-couple from 1-6, 1 at the bottom using specified units returns HPWH\_ABORT for N < 0, > 6, or incorrect units

**2.2.3.10 int HPWH::getNumHeatSources ( ) const**

returns the number of heat sources

**2.2.3.11 int HPWH::getNumNodes ( ) const**

returns the number of nodes

**2.2.3.12 double HPWH::getOutletTemp ( UNITS *units* ) const**

returns the outlet temperature in the specified units returns 0 when no draw occurs, or HPWH\_ABORT for incorrect unit specifier

**2.2.3.13 double HPWH::getStandbyLosses ( UNITS *units* ) const**

get the amount of heat lost through the tank in specified units moving heat from the water to the space is the positive direction negative should occur seldom returns HPWH\_ABORT for incorrect units

**2.2.3.14 double HPWH::getTankNodeTemp ( int *nodeNum* ) const**

## Parameters

|                |                 |
|----------------|-----------------|
| <i>nodeNum</i> | default units C |
|----------------|-----------------|

**2.2.3.15** `double HPWH::getTankNodeTemp ( int nodeNum, UNITS units ) const`

returns the temperature of the water at the specified node - with specified units or HPWH\_ABORT for incorrect node number or unit failure

**2.2.3.16** `int HPWH::HPWHinit_file ( std::string configFile )`

This function will load in a set of parameters from a file The file name is the input - there should be at most one set of parameters per file This is useful for testing new variations, and for the sort of variability that we typically do when creating SEEM runs Appropriate use of this function can be found in the documentation

The return value is 0 for successful initialization, HPWH\_ABORT otherwise

**2.2.3.17** `int HPWH::HPWHinit_presets ( MODELS presetNum )`

This function will load in a set of parameters that are hardcoded in this function - which particular set of parameters is selected by presetNum. This is similar to the way the HPWHsim currently operates, as used in SEEM, but not quite as versatile. My impression is that this could be a useful input paradigm for CSE

The return value is 0 for successful initialization, HPWH\_ABORT otherwise

**2.2.3.18** `int HPWH::isNthHeatSourceRunning ( int N ) const`

returns 1 if the Nth heat source is currently engaged, 0 if it is not, and returns HPWH\_ABORT for N out of bounds

**2.2.3.19** `HPWH & HPWH::operator= ( const HPWH & hpwh )`

assignment operator

**2.2.3.20** `void HPWH::printHeatSourceInfo ( )`

this prints out the heat source info, nicely formatted specifically input/output energy/power, and runtime will print to cout if messageCallback pointer is unspecified does not use verbosity, as it is public and expected to be called only when needed

**2.2.3.21** `void HPWH::printTankTemps ( )`

this prints out all the node temps, kind of nicely formatted does not use verbosity, as it is public and expected to be called only when needed

**2.2.3.22** `int HPWH::resetTankToSetpoint ( )`

this function resets the tank temperature profile to be completely at setpoint The return value is 0 for successful completion

**2.2.3.23** `int HPWH::runNSteps ( int N, double * inletT_C, double * drawVolume_L, double * tankAmbientT_C, double * heatSourceAmbientT_C, DRMODES * DRstatus, double minutesPerStep )`

This function will progress the simulation forward in time by N (equal) steps The calculated values will be summed or averaged, as appropriate, and then stored in the usual variables to be accessed through functions

The return value is 0 for successful simulation run, HPWH\_ABORT otherwise

**2.2.3.24** `int HPWH::runOneStep ( double inletT_C, double drawVolume_L, double ambientT_C, double externalT_C, DRMODES DRstatus, double minutesPerStep )`

This function will progress the simulation forward in time by one step all calculated outputs are stored in private variables and accessed through functions

The return value is 0 for successful simulation run, HPWH\_ABORT otherwise

**2.2.3.25** `int HPWH::runOneStep ( double drawVolume_L, double ambientT_C, double externalT_C, DRMODES DRstatus )`  
[inline]

An overloaded function that uses some member variables, instead of taking them as inputs

**2.2.3.26** `void HPWH::setInletT ( double newInletT_C )` [inline]

Setters for the what are typically input variables

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

sets the function to be used for message passing

**2.2.3.28** `int HPWH::setSetpoint ( double newSetpoint )`

Parameters

|                    |                 |
|--------------------|-----------------|
| <i>newSetpoint</i> | default units C |
|--------------------|-----------------|

**2.2.3.29** `int HPWH::setSetpoint ( double newSetpoint, UNITS units )`

a function to change the setpoint - useful for dynamically setting it The return value is 0 for successful setting, HPWH\_ABORT for units failure

**2.2.3.30** `void HPWH::setVerbosity ( VERBOSITY hpwhVrb )`

sets the verbosity to the specified level

**2.2.3.31** `int HPWH::WriteCSVRow ( FILE * outFILE, const char * preamble = " " ) const`

a couple of function to write the outputs to a file they both will return 0 for success the preamble should be supplied with a trailing comma, as these functions do not add one. Additionally, a newline is written with each call.

## 2.2.4 Member Data Documentation

### 2.2.4.1 `const int HPWH::HPWH_ABORT = -274000` `[static]`

this is the value that the public functions will return in case of a simulation destroying error

The documentation for this class was generated from the following files:

- `/home/nkvaltine/nkvaltine/Projects/HPWHsim/dev/HPWH.hh`
- `/home/nkvaltine/nkvaltine/Projects/HPWHsim/dev/HPWH.cc`

# Index

- ~HPWH
  - HPWH, [7](#)
- addHeat
  - HPWH::HeatSource, [3](#)
- DR\_ALLOW
  - HPWH, [6](#)
- DR\_BLOCK
  - HPWH, [6](#)
- DR\_ENGAGE
  - HPWH, [6](#)
- DRMODES
  - HPWH, [6](#)
- disengageHeatSource
  - HPWH::HeatSource, [3](#)
- engageHeatSource
  - HPWH::HeatSource, [4](#)
- getEnergyRemovedFromEnvironment
  - HPWH, [7](#)
- getNthHeatSourceEnergyInput
  - HPWH, [7](#)
- getNthHeatSourceEnergyOutput
  - HPWH, [7, 8](#)
- getNthHeatSourceRunTime
  - HPWH, [8](#)
- getNthHeatSourceType
  - HPWH, [8](#)
- getNthSimTcouple
  - HPWH, [8](#)
- getNumHeatSources
  - HPWH, [8](#)
- getNumNodes
  - HPWH, [8](#)
- getOutletTemp
  - HPWH, [8](#)
- getStandbyLosses
  - HPWH, [8](#)
- getTankNodeTemp
  - HPWH, [8, 9](#)
- HPWH
  - DR\_ALLOW, [6](#)
  - DR\_BLOCK, [6](#)
  - DR\_ENGAGE, [6](#)
  - MODELS\_GEGeospring, [6](#)
  - MODELS\_SandenGAU, [6](#)
  - MODELS\_SandenGES, [6](#)
  - MODELS\_Voltex60, [6](#)
  - MODELS\_Voltex80, [6](#)
  - MODELS\_basicIntegrated, [6](#)
  - MODELS\_externalTest, [6](#)
  - MODELS\_restantkHugeUA, [6](#)
  - MODELS\_restantkNoUA, [6](#)
  - MODELS\_restantkRealistic, [6](#)
  - TYPE\_compressor, [6](#)
  - TYPE\_none, [6](#)
  - TYPE\_resistance, [6](#)
  - UNITS\_BTU, [6](#)
  - UNITS\_C, [6](#)
  - UNITS\_F, [6](#)
  - UNITS\_KJ, [6](#)
  - UNITS\_KWH, [6](#)
  - VRB\_emetic, [7](#)
  - VRB\_reluctant, [7](#)
  - VRB\_silent, [7](#)
  - VRB\_typical, [7](#)
- HEATSOURCE\_TYPE
  - HPWH, [6](#)
- HPWH, [4](#)
  - ~HPWH, [7](#)
  - DRMODES, [6](#)
  - getEnergyRemovedFromEnvironment, [7](#)
  - getNthHeatSourceEnergyInput, [7](#)
  - getNthHeatSourceEnergyOutput, [7, 8](#)
  - getNthHeatSourceRunTime, [8](#)
  - getNthHeatSourceType, [8](#)
  - getNthSimTcouple, [8](#)
  - getNumHeatSources, [8](#)
  - getNumNodes, [8](#)
  - getOutletTemp, [8](#)
  - getStandbyLosses, [8](#)
  - getTankNodeTemp, [8, 9](#)
  - HEATSOURCE\_TYPE, [6](#)
  - HPWH, [7](#)
  - HPWH\_ABORT, [11](#)
  - HPWHinit\_file, [9](#)
  - HPWHinit\_presets, [9](#)
  - HPWH, [7](#)
  - isNthHeatSourceRunning, [9](#)
  - MODELS, [6](#)
  - operator=, [9](#)
  - printHeatSourceInfo, [9](#)
  - printTankTemps, [9](#)
  - resetTankToSetpoint, [9](#)
  - runNSteps, [9](#)
  - runOneStep, [10](#)
  - setInletT, [10](#)



- setMessageCallback, 10
- setSetpoint, 10
- setVerbosity, 10
- UNITS, 6
- VERBOSITY, 6
- WriteCSVRow, 10
- HPWH::HeatSource, 3
  - addHeat, 3
  - disengageHeatSource, 3
  - engageHeatSource, 4
  - HeatSource, 3
  - isEngaged, 4
  - setCondensity, 4
  - setupAsResistiveElement, 4
  - shouldHeat, 4
  - shutsOff, 4
- HPWH\_ABORT
  - HPWH, 11
- HPWHinit\_file
  - HPWH, 9
- HPWHinit\_presets
  - HPWH, 9
- HeatSource
  - HPWH::HeatSource, 3
- isEngaged
  - HPWH::HeatSource, 4
- isNthHeatSourceRunning
  - HPWH, 9
- MODELS\_GEGeospring
  - HPWH, 6
- MODELS\_SandenGAU
  - HPWH, 6
- MODELS\_SandenGES
  - HPWH, 6
- MODELS\_Voltex60
  - HPWH, 6
- MODELS\_Voltex80
  - HPWH, 6
- MODELS\_basicIntegrated
  - HPWH, 6
- MODELS\_externalTest
  - HPWH, 6
- MODELS\_restankHugeUA
  - HPWH, 6
- MODELS\_restankNoUA
  - HPWH, 6
- MODELS\_restankRealistic
  - HPWH, 6
- MODELS
  - HPWH, 6
- operator=
  - HPWH, 9
- printHeatSourceInfo
  - HPWH, 9
- printTankTemps
  - HPWH, 9
- resetTankToSetpoint
  - HPWH, 9
- runNSteps
  - HPWH, 9
- runOneStep
  - HPWH, 10
- setCondensity
  - HPWH::HeatSource, 4
- setInletT
  - HPWH, 10
- setMessageCallback
  - HPWH, 10
- setSetpoint
  - HPWH, 10
- setVerbosity
  - HPWH, 10
- setupAsResistiveElement
  - HPWH::HeatSource, 4
- shouldHeat
  - HPWH::HeatSource, 4
- shutsOff
  - HPWH::HeatSource, 4
- TYPE\_compressor
  - HPWH, 6
- TYPE\_none
  - HPWH, 6
- TYPE\_resistance
  - HPWH, 6
- UNITS\_BTU
  - HPWH, 6
- UNITS\_C
  - HPWH, 6
- UNITS\_F
  - HPWH, 6
- UNITS\_KJ
  - HPWH, 6
- UNITS\_KWH
  - HPWH, 6
- UNITS
  - HPWH, 6
- VRB\_emetic
  - HPWH, 7
- VRB\_reluctant
  - HPWH, 7
- VRB\_silent
  - HPWH, 7
- VRB\_typical
  - HPWH, 7
- VERBOSITY
  - HPWH, 6
- WriteCSVRow
  - HPWH, 10