

What is ShedSolar?

ShedSolar is a program that provides several services related to the Outback solar system that I have installed in my shed:

- Controls the temperature of the batteries. My system has three Discover AES 7.4KWH LiFePO4 batteries. These batteries cannot be charged if their temperature is below 0C (32F), and they are not in a heated building. Here in our high northern Utah valley we routinely see temperatures as low as -15C (0F), so my system's batteries are inside an insulated box with a 200W heater. *ShedSolar* will sense the battery temperature and turn the heater on as required to keep the battery temperature at an appropriate level. During hours when the sun is shining (and the batteries can therefore be charged), the battery temperature will be kept within the range of 15C to 20C (59F to 68F). Otherwise, the range will drop to 0C to 5C (32F to 41F) to lower the heater power requirements. It obtains the sunlight level from my weather system, which directly senses solar power with a pyrometer. The total time that the heater is powered on is tracked internally.
- Monitors the temperature of the batteries, providing alarm events for under-temperature (<0C, which might occur on heater failure) or over-temperature (>45C, which might occur if the batteries generate too much heat when the ambient temperature is high).
- Interrogates the Outback Mate3S every 60 seconds (by default) through the standard JSON API, processes the result, and holds it internally.
- Posts an event every 60 seconds. This event results in solar system data being published in our database.
- Publishes a solar system report message every 60 seconds, which any MOP client can subscribe to.

Why does the world need ShedSolar?

Well, probably the world doesn't actually *need* ShedSolar—it's mainly here for the author's personal use and enjoyment, but with some faintish hope that someone else with the same challenges the author faced will also find it useful.

Dependencies

ShedSolar has several dependencies:

- *MOP* is a message-oriented programming module the author also wrote, freely available from [here](#).
- *Util* is a utilities module the author also wrote, freely available from [here](#).
- *JSON* is the bog-standard Java JSON module, freely available from [here](#).
- *commons-suncalc* performs sun-related calculations, freely available from [here](#)

Why is ShedSolar's code so awful?

The author is a retired software and hardware engineer who did this just for fun, and who (so far, anyway) has no code reviewers to upbraid him. Please feel free to fill in this gap! You may contact the author at tom@dilatush.com.

Software Components

TempReader

The **TempReader** provides feedback for ShedSolar about the temperature inside the battery box. In normal operation, ShedSolar uses this information to determine when the battery box heater needs to be turned on or off, and also to verify that the heater is working correctly.

ShedSolar has two thermocouples, both located inside the battery box. One of them, for battery temperature, is located underneath one of the batteries, out of the path of air circulated by the heater's fan. The other one, for heater output temperature, is located in front of the fan, directly in the path of the air pushed out of the heater by its fan. These two thermocouples are attached to a pair of MAX31855 type K thermocouple interface chips, which are in turn attached to the Raspberry Pi by SPI.

One might expect that reading the temperature with such a setup would be a near-trivial affair, but such is not the case. For reasons that we've never been able to figure out, both of the MAX31855 chips read anomalous values (lower than actual) for about 2 seconds out of every 10. To deal with this, we've implemented a noise filter that throws out samples that are furthest from the mean of over 10 seconds of samples. This seems to work quite well.

The MAX31855 chips are capable of detecting the common kinds of thermocouple failures (shorts and opens). They also happen to provide ambient temperature (at the chip), albeit at a lower resolution than the thermocouple temperature. The **TempReader** module publishes all of this information in **InfoView** instances:

public final variable	Information
batteryTemperatureSensorStatus	A TempSensorStatus structure with detailed information about the status of the sensor.
batteryTemperature	The temperature of the battery thermocouple, in °C ±0.25°C.
heaterTemperatureSensorStatus	A TempSensorStatus structure with detailed information about the status of the sensor.
heaterTemperature	The temperature of the heater output thermocouple, in °C ±0.25°C.
ambientTemperature	The temperature of the MAX31855 chip (inside the ShedSolar box) in °C, ±0.0625°C.

Outbacker

The **Outbacker** provides information read periodically from the Outback charger/inverter, so long as the ShedSolar system can "talk" to it. The Outback charger/inverter has a simple JSON API at `<IP address>/Dev_status.cgi?&Port=0`. Outbacker reads this URL by default once per minute, decodes and collects the information ShedSolar is interested in, and publishes it as an **OutbackData** structure in an **InfoView** at `public final outback`.

ShedSolarActor

The **ShedSolarActor** provides a MOP postoffice for ShedSolar. It also subscribes to the per-minute weather report from our weather station to get the solar irradiance and outdoor temperature information that are useful to ShedSolar. It publishes this information in **InfoView** instances `public final solarIrradiance` (reporting watts per square meter) and `outsideTemperature` (reporting °C, $\pm 1\%$).

LightDetector

The **LightDetector** provides ShedSolar with a key piece of information: whether there is enough light for the solar panels to produce electricity (which will charge the batteries). ShedSolar uses this information to determine how cold it may allow the batteries to get. The batteries can safely get colder (down to 0°C) when they're not being charged, but to be charged they should be warmer (at least 10°C, but preferably more like 20°C). Allowing the batteries to get colder (when it is safe to do so) has the advantage of requiring less heater power.

At first blush, it may seem that this job is as simple as knowing whether it is daytime or nighttime, but in that pesky real world it's actually a bit more complicated. Most importantly, the solar panels may be covered with snow — so even if the sun is shining brightly, they can't generate any power. Usually the snow will slide off after a few hours of sunlight, but it is possible for the snow to stay on even longer. In addition, thick clouds and precipitation can (and in the winter, often do) dim the sunlight so much the solar panels cannot produce power.

LightDetector has three different ways to detect when it's light, depending on what information it has available. In preference order:

1. **Directly detecting the solar panels' power production:** If the state of charge of the solar system's batteries is under about 98%, the solar panels will produce power whenever there's enough sun shining on them. The **Outbacker**, if it can communicate with the Outback charger/inverter, provides the solar panel's output voltage and current. The output voltage times the output current equals the output power, and if this is over about 100 watts, then we know that (a) it's daylight, (b) there aren't enough clouds to block the sun, and (c) the panels are not covered with snow. If ShedSolar can "talk" with the Outback charger/inverter and the batteries' state of charge is under 98%, this is the information we use.
2. **Reading the solar power from our weather station's pyrometer:** When we can't use the solar panels power production as our information source, the next best is our weather station's pyrometer (a device for measuring total solar power output). From observation, the solar panels (if not blocked by snow) will produce power when the pyrometer is reporting above about 225

watts per square meter of solar irradiation. The weather station is located about 150 meters from the shed's solar panels, so it will not always have the same sun conditions as the panels (scudding clouds, for instance, can leave one in sunlight while the other is shaded). However, in the absence of direct solar panel power production measurement, it's our next best source of information. Normally our weather station sends out reports once a minute. The `ShedSolarActor` subscribes to these reports and publishes the information if it's available.

3. **Using computed sunrise and sunset times to determine when it is day or night:** If neither method above is available, `LightDetector` falls back to simply computing whether it's day or night. This requires just the latitude and longitude of the shed's solar panels, and the time.

`LightDetector` publishes its information in the `InfoView` variable `public final light`, which contains the `LightDetector.Mode` value of either `LIGHT` or `DARK`.

BatteryTempLED

The `BatteryTempLED` has a very simple job: to flash the battery temperature LED, using the duty cycle of the LED to indicate the battery temperature. The LED has two modes:

1. If battery temperature information is not available, the LED "rapid flashes" (by default at 0.8Hz) to indicate a problem.
2. If battery temperature is available, the LED's duty cycle (by default over a 2 second period) varies to indicate the battery temperature. At the low-end (LED always off) the battery temperature is by default 0°C or less. At the high-end (LED always on), the battery temperature is by default 45°C or greater. Thus, by default, the LED's duty cycle is 50% (half on, half off) when the battery temperature is 22.5°C (72.5°F).

HeaterControl

The `HeaterControl` turns the heater (in the battery box) on and off, along with the associated heater power indicator LED. `HeaterControl` uses information from the other ShedSolar components to decide when to do this. It has four modes of operation, depending on whether battery temperature information or heater output temperature is available. Each of these modes is handled by a different component:

Information Available	Component that handles it
battery temperature <i>and</i> heater output temperature	Normal mode, handled by <code>NormalHeaterController</code>
<i>only</i> battery temperature	Battery-only mode, handled by <code>BatteryOnlyHeaterController</code>
<i>only</i> heater output temperature	Heater-only mode, handled by <code>HeaterOnlyHeaterController</code>
<i>neither</i> battery temperature or heater output temperature	NoTemps mode, handled by <code>NoTempsHeaterController</code>

NormalHeaterController

In normal mode, `NormalHeaterController` can do the best job maintaining battery temperature. The battery temperature is used to directly sense when the heater needs to be turned on or off. The heater output temperature is used to verify that the heater has turned on or off. A failure to turn on *could* indicate that the heater has failed. However, by observation we know that the heater has a sort of thermal "fuse" that prevents it from turning on if the heater is still hot. We speculate that this thermal fuse doesn't trip during normal operation as the heater's fan is blowing cold air through the heater constantly, preventing it from overheating. However, immediately after turning the heater off the flow of air ceases, and this causes the inside of the heater (and the thermal fuse) to heat up. In any case, we have learned that if the heater fails to turn on, simply waiting for a few minutes for it to cool down will solve the problem. So `NormalHeaterController` has logic to detect the failure of the heater to start, and when that happens, to wait for a while before trying again. Only after several attempts (and failures) will it trigger an alert about a heater failure. After the heater has been on for a while and is then turned off, `NormalHeaterController` also has logic to enforce a cooldown period. In addition, the availability of heater output temperature allows a safety measure: if the heater output temperature gets higher than is safe for the batteries, the heater will be shut off.

BatteryOnlyHeaterController

In battery-only mode, `BatteryOnlyHeaterController` is very similar to `NormalHeaterController`, except that confirmation of the heater working is indirect (and takes longer), and the overtemperature safety measure can't be implemented.

HeaterOnlyHeaterController

In heater-only mode, `HeaterOnlyHeaterController` can't sense the battery temperature while the heater is running, but if the heater has been off for a while (a few minutes), the air temperature around the batteries (where the heater output thermocouple is located) will be a reasonable approximation of the battery temperature. `HeaterOnlyHeaterController` leverages this fact, by running the heater for a fixed period, turning off and waiting for the air temperature to cool down to roughly the batteries' temperature, *then* using that temperature to decide when to turn the heater back on.

NoTempsHeaterController

In no-temps mode we have the most challenging scenario for the `HeaterControl`, implemented by `NoTempsHeaterController`. It falls back on a method with no direct feedback—the "open loop" solution. First it gets the outside temperature (either from the ambient temperature capability of `TempReader` or from our weather station via `ShedSolarActor`). Then it figures the difference between the outside temperature and the target temperature for the batteries. Finally, it uses a simple formula to calculate the duty cycle that the heater needs to run in order to stay above the target temperature. This formula assumes a linear relationship between average heater power and average temperature difference between the battery box and the outside air.

Some implementation notes...

Hardware

The hardware used in this project, excluding cables and connections, is as follows:

- One Raspberry Pi 3B+ (with CanaKit wall wart power supply)
- Two Adafruit 269 thermocouple interfaces (MAX 31855 chip)
- Two type K thermocouples with 2 meter leads
- One AOLE ASH-10DA solid state relay (10 amp, 120VAC output, 3 volt input)
- One Omron LY2-UA-006244 relay
- Three 5mm LEDs (one green, two red)
- Three 220 ohm, 1/4 watt resistors

The Raspberry Pi is the heart of the system. One thermocouple and interface measures the temperature of the batteries (it's placed physically under a battery, where there is no air flow). The other thermocouple measures the air temperature at the output of the heater; this allows the Raspberry Pi to sense whether the heater is working. The solid state relay controls the heater. The electro-mechanical relay senses the output of the solid state relay; this allows the Raspberry Pi to sense whether the solid state relay is working. The author assumes that the two most likely failure points are (a) the heater, which has moving parts and hot parts, and (b) the solid state relay, simply because it's dealing with power lines. The LEDs are driving by software, with the following meanings:





















- **Battery Temperature:** a 0.5 Hz flashing indicator whose duty cycle indicates the battery temperature: From 0% on to 100% on indicates 0C to 45C, which is the range of temperatures that my solar system batteries (Discover AES 42-48-6650 LiFePO4) may safely be charged. This LED fast-flashes if the battery temperature can't be read.
- **Heater Power:** this indicator is on when the heater has been turned on.
- **Status:** A flashing indicator that encodes some simple status information (see [Status Codes](#) section below).

Raspberry Pi I/O Usage

The following I/O pins are used for this project:

- **GPIO 14 / SCLK:** the SPI clock, to both thermocouple interfaces
- **GPIO 13 / MISO:** The SPI data in, to both thermocouple interfaces
- **GPIO 10 / CE0:** The SPI chip enable 0, to the battery thermocouple interface
- **GPIO 11 / CE1:** The SPI chip enable 1, to the heater thermocouple interface
- **GPIO 0:** Sense relay (pulled high, low means SSR is outputting 120VAC)
- **GPIO 2:** Battery Temperature LED (red), low is on

- **GPIO 3:** Heater Power LED (red), low is on
- **GPIO 4:** Status LED (green), low is on
- **GPIO 5:** Heater SSR, low is on

Raspberry Pi 3 Model B (J8 Header)					
GPIO#	NAME			NAME	GPIO#
	3.3 VDC Power	1		2	5.0 VDC Power
8	GPIO 8 SDA1 (I2C)	3		4	5.0 VDC Power
9	GPIO 9 SCL1 (I2C)	5		6	Ground
7	GPIO 7 GPCLK0	7		8	GPIO 15 TxD (UART) 15
	Ground	9		10	GPIO 16 RxD (UART) 16
0	GPIO 0	11		12	GPIO 1 PCM_CLK/PWM0 1
2	GPIO 2	13		14	Ground
3	GPIO 3	15		16	GPIO 4 4
	3.3 VDC Power	17		18	GPIO 5 5
12	GPIO 12 MOSI (SPI)	19		20	Ground
13	GPIO 13 MISO (SPI)	21		22	GPIO 6 6
14	GPIO 14 SCLK (SPI)	23		24	GPIO 10 CE0 (SPI) 10
	Ground	25		26	GPIO 11 CE1 (SPI) 11
30	SDA0 (I2C ID EEPROM)	27		28	SCL0 (I2C ID EEPROM) 31
21	GPIO 21 GPCLK1	29		30	Ground
22	GPIO 22 GPCLK2	31		32	GPIO 26 PWM0 26
23	GPIO 23 PWM1	33		34	Ground
24	GPIO 24 PCM_FS/PWM1	35		36	GPIO 27 27
25	GPIO 25	37		38	GPIO 28 PCM_DIN 28
	Ground	39		40	GPIO 29 PCM_DOUT 29

Attention! The GPIO pin numbering used in this diagram is intended for use with WiringPi / Pi4J. This pin numbering is not the raw Broadcom GPIO pin numbers.

<http://www.pi4j.com>

Status Codes

These are the codes displayed by the status indicator. There may be multiple status codes, in which case the status indicator will be off briefly between the codes. Once all the codes have been displayed, the status indicator will be off for a longer pause, then start over again. A short flash on indicates a zero, a long flash a one. The codes it can display are shown below. They are transmitted MSB first.

Code	Status
0	Ok - no problems detected
1	State of charge under 20%
10	Batteries undertemperature
11	Batteries overtemperature
00	Barn router unreachable
01	Shed router unreachable
000	Internet unreachable
001	CPO not connected
110	Solid state relay failure
111	Heater failure
1100	Weather reports not being received
1101	Outback MATE3S not responding
0000	Battery thermocouple open circuit
0001	Battery thermocouple shorted to Vcc
0010	Battery thermocouple shorted to ground
0100	Heater thermocouple open circuit
0101	Heater thermocouple shorted to Vcc
0110	Heater thermocouple shorted to ground

How is ShedSolar licensed?

MOP is licensed with the quite permissive MIT license:

Created: November 16, 2020

Author: Tom Dilatush link:mailto:tom@dilatush.com

Github: <https://github.com/SlightlyLoony/ShedSolar>

License: MIT

Copyright 2020, 2021 by Tom Dilatush (aka "SlightlyLoony")

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so.

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE A AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.