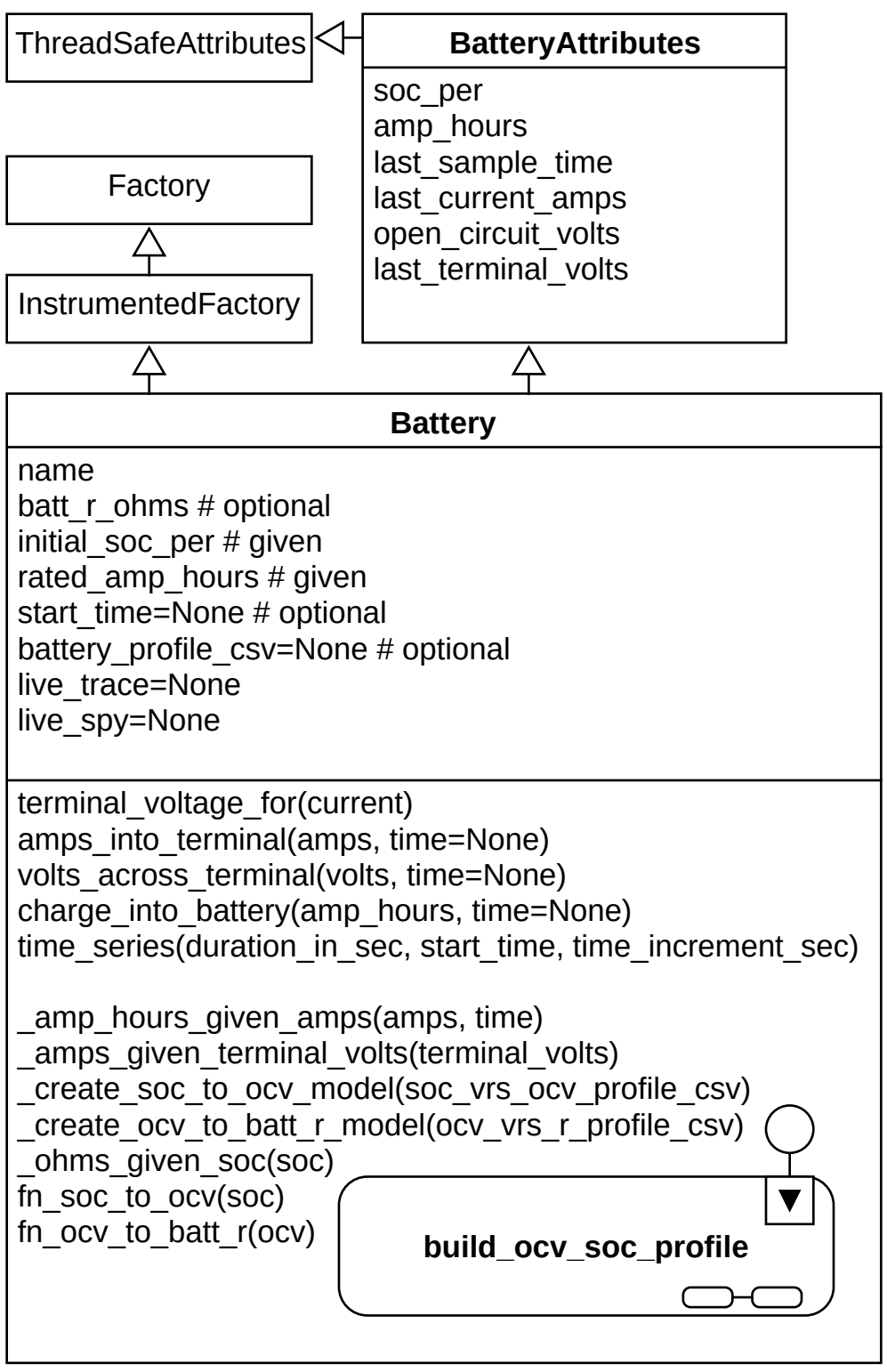
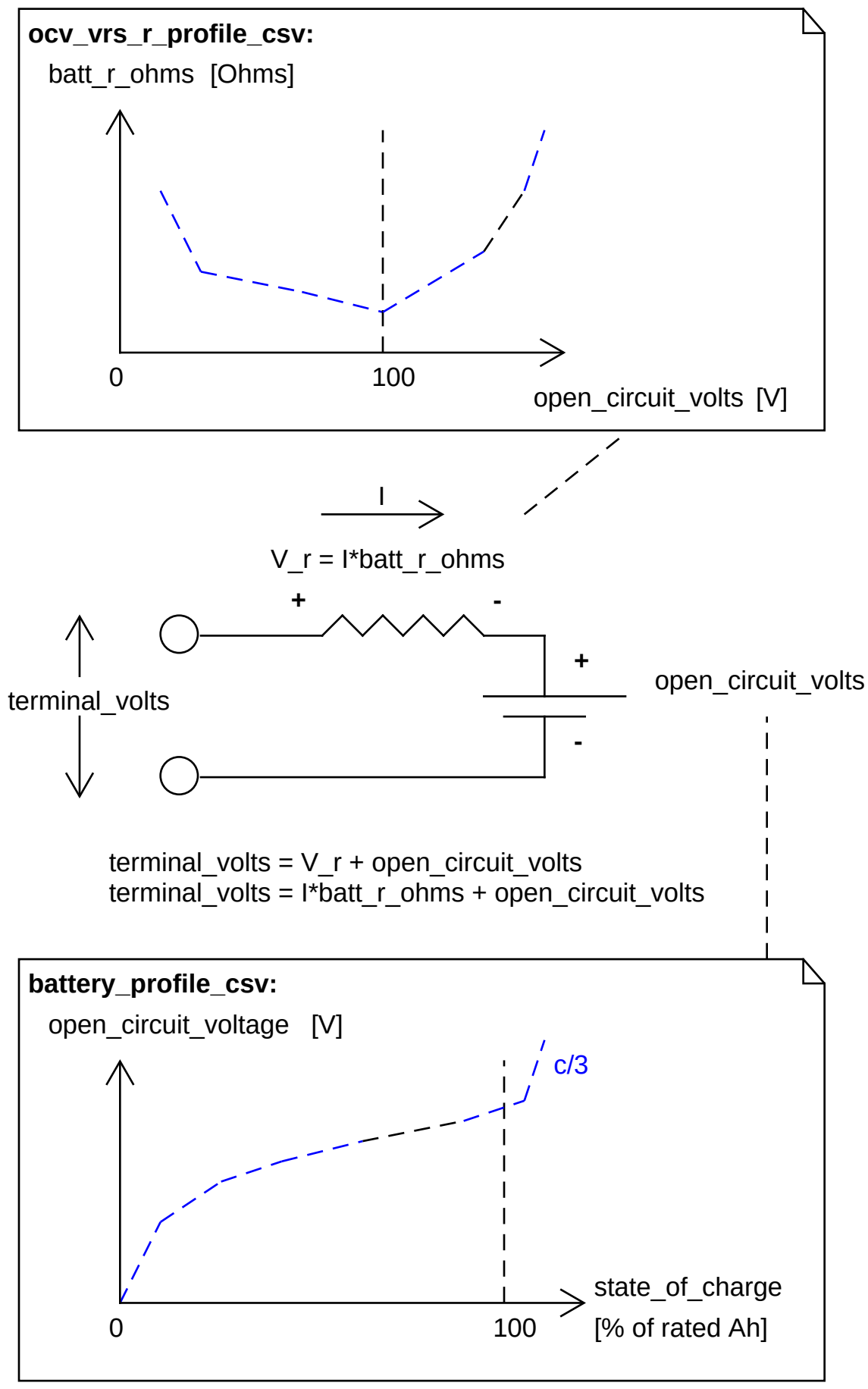
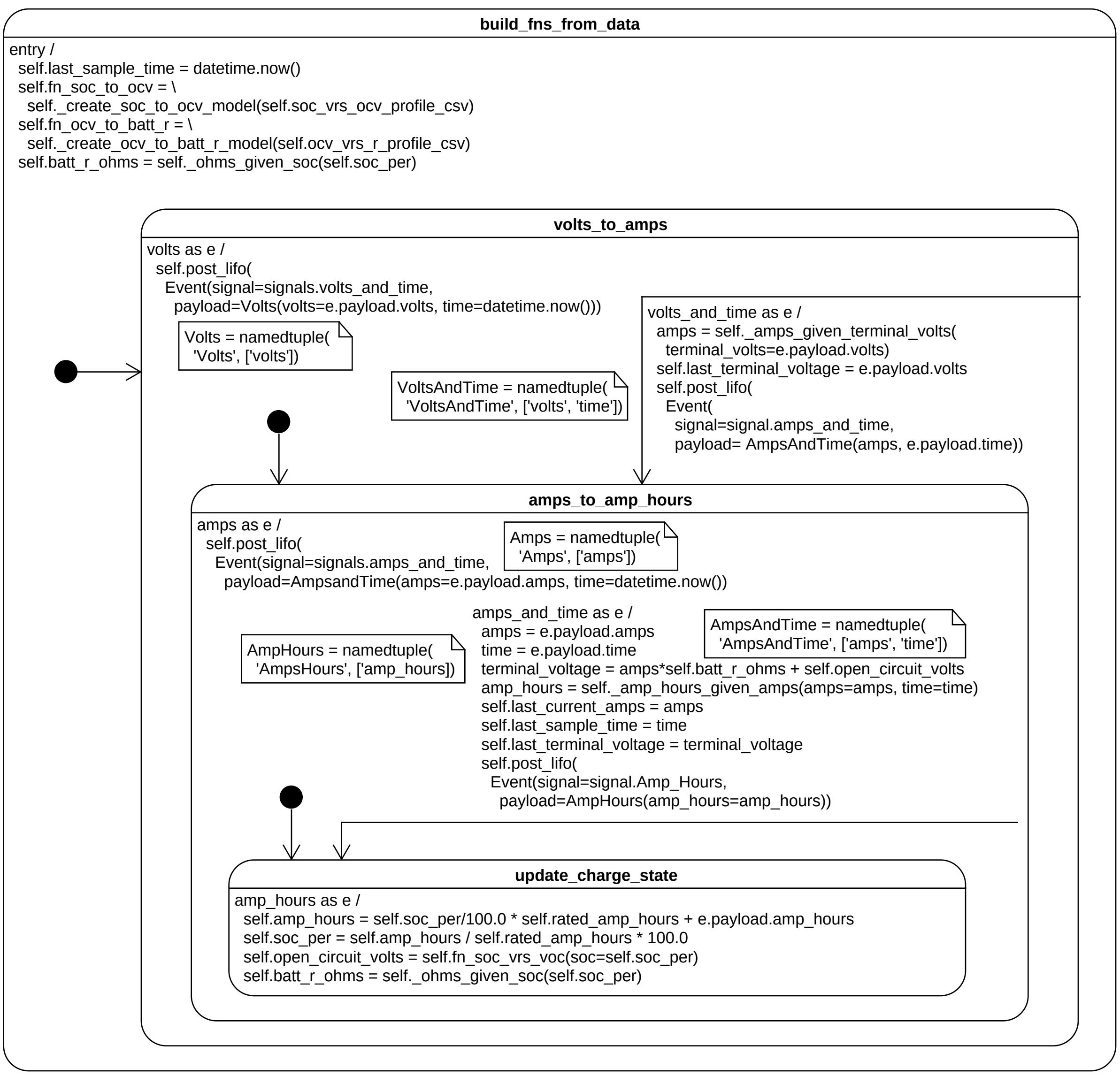

ThreadSafeAttributes
BatteryAttributes
soc_per
amp_hours
last_sample_time
last_current_amps
open_circuit_volts
last_terminal_volts
Factory
InstrumentedFactory
Battery
name
batt_r_ohms # optional
initial_soc_per # given
rated_amp_hours # given
start_time=None # optional
battery_profile_csv=None # optional
live_trace=None
live_spy=None
terminal_voltage_for(current)
amps_into_terminal(amps, time=None)
volts_across_terminal(volts, time=None)
charge_into_battery(amp_hours, time=None)
time_series(duration_in_sec, start_time, time_increment_sec)
_amp_hours_given_amps(amps, time)
_amps_given_terminal_volts(terminal_volts)
_create_soc_to_ocv_model(soc_vrs_ocv_profile_csv)
_create_ocv_to_batt_r_model(ocv_vrs_r_profile_csv)
_ohms_given_soc(soc)
fn_soc_to_ocv(soc)
fn_ocv_to_batt_r(ocv)
build_ocv_soc_profile


ocv_vrs_r_profile_csv:
batt_r_ohms [Ohms]
0
100
open_circuit_volts [V]
I
V_r = I*batt_r_ohms
+
-
+
-
terminal_volts
open_circuit_volts
terminal_volts = V_r + open_circuit_volts
terminal_volts = I*batt_r_ohms + open_circuit_volts
battery_profile_csv:
open_circuit_voltage [V]
c/3
0
100
state_of_charge
[% of rated Ah]


build_fns_from_data
entry /
self.last_sample_time = datetime.now()
self.fn_soc_to_ocv = \
self._create_soc_to_ocv_model(self.soc_vrs_ocv_profile_csv)
self.fn_ocv_to_batt_r = \
self._create_ocv_to_batt_r_model(self.ocv_vrs_r_profile_csv)
self.batt_r_ohms = self._ohms_given_soc(self.soc_per)
volts_to_amps
volts as e /
self.post_lifo(
Event(signal=signals.volts_and_time,
payload=Volts(volts=e.payload.volts, time=datetime.now()))
Volts = namedtuple(
'Volts', ['volts'])
VoltsAndTime = namedtuple(
'VoltsAndTime', ['volts', 'time'])
volts_and_time as e /
amps = self._amps_given_terminal_volts(
terminal_volts=e.payload.volts)
self.last_terminal_voltage = e.payload.volts
self.post_lifo(
Event(
signal=signal.amps_and_time,
payload= AmpsAndTime(amps, e.payload.time))
amps_to_amp_hours
amps as e /
self.post_lifo(
Event(signal=signals.amps_and_time,
payload=AmpsandTime(amps=e.payload.amps, time=datetime.now())
Amps = namedtuple(
'Amps', ['amps'])
amps_and_time as e /
amps = e.payload.amps
time = e.payload.time
terminal_voltage = amps*self.batt_r_ohms + self.open_circuit_volts
amp_hours = self._amp_hours_given_amps(amps=amps, time=time)
self.last_current_amps = amps
self.last_sample_time = time
self.last_terminal_voltage = terminal_voltage
self.post_lifo(
Event(signal=signal.Amp_Hours,
payload=AmpHours(amp_hours=amp_hours))
AmpsAndTime = namedtuple(
'AmpsAndTime', ['amps', 'time'])
AmpHours = namedtuple(
'AmpsHours', ['amp_hours])
update_charge_state
amp_hours as e /
self.amp_hours = self.soc_per/100.0 * self.rated_amp_hours + e.payload.amp_hours
self.soc_per = self.amp_hours / self.rated_amp_hours * 100.0
self.open_circuit_volts = self.fn_soc_vrs_voc(soc=self.soc_per)
self.batt_r_ohms = self._ohms_given_soc(self.soc_per)