InstrumentedFactory

Battery

rated_amp_hours # given
batt_r_ohms # given
initial_soc_per # given
battery_profile_csv # given
start_time # optional

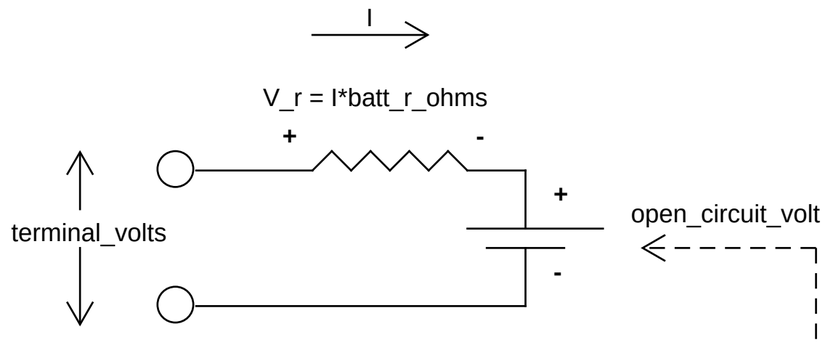
open_circuit_volts
soc_per
amp_hours
last_current_amps
last_terminal_volts
last_sample_time

_amp_hours_given_amps(amps, time)
_amps_given_terminal_volts(terminal_volts)
_create_battery_model(battery_profile_csv)
fn_soc_to_ocv(soc)

build_ocv_soc_profile

```
def _amps_given_terminal_volts(terminal_volts):
  voc = self.fn_soc_to_ocv(self.soc_per)
  v_r_volts = terminal_volts - voc
  amps = v_r_volts / self.batter_r_ohms
  return amps
```

```
def _amp_hours_given_amps(amps, time):
  delta_t_sec = (time - self.last_sample_time).total_seconds()
  amp_hours = amps * delta_t_sec / 3600.0
  return amp_hours
```



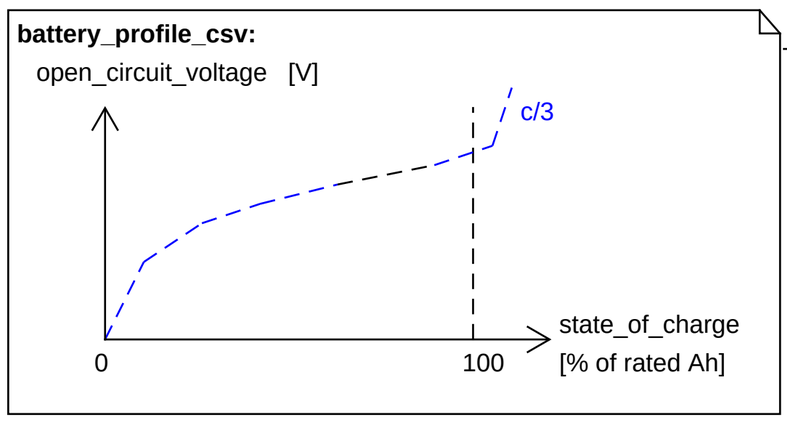
terminal_volts = V_r + open_circuit_volts
terminal_volts = I*batt_r_ohms + open_circuit_volts

```
def _create_battery_model(battery_profile_csv):
  data_ocv_soc = np.getfromtxt(
    self.battery_profile_csv,
    delimiter=',',
    skip_header=1,
    names=['state_of_charge', 'open_circuit_voltage'],
    dtype='float, float')
  self.fn_soc_to_ocv = \
    interp1d(
     data_ocv_soc['state_of_charge'],
     data_ocv_soc['open_circuit_voltage']
    )
```

build_ocv_soc_profile

```
entry /
  self.start_time = datetime.now()
  self.fn_soc_to_ocv = \
    self.create_battery_model(self.battery_profile_csv)
```

battery_profile_csv:



volts_to_amps

```
volts as e /
  self.post_lifo(
    Event(signal=signals.volts_and_time,
      payload=Volts(volts=e.payload.volts, time=datetime.now()))
```

```
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=Amps(amps=e.payload.amps, time=datetime.now())
```

```
AmpsAndTime = namedtuple(
  'AmpsAndTime', ['amps', 'time'])
```

```
AmpHours = namedtuple(
  'AmpsHours', ['amp_hours])
```

```
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 = jtime
  self.last_terminal_voltage = terminal_voltage
  self.post_lifo(
    Event(signal=signal.Amp_Hours,
      payload=AmpHours(amp_hours=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)
```