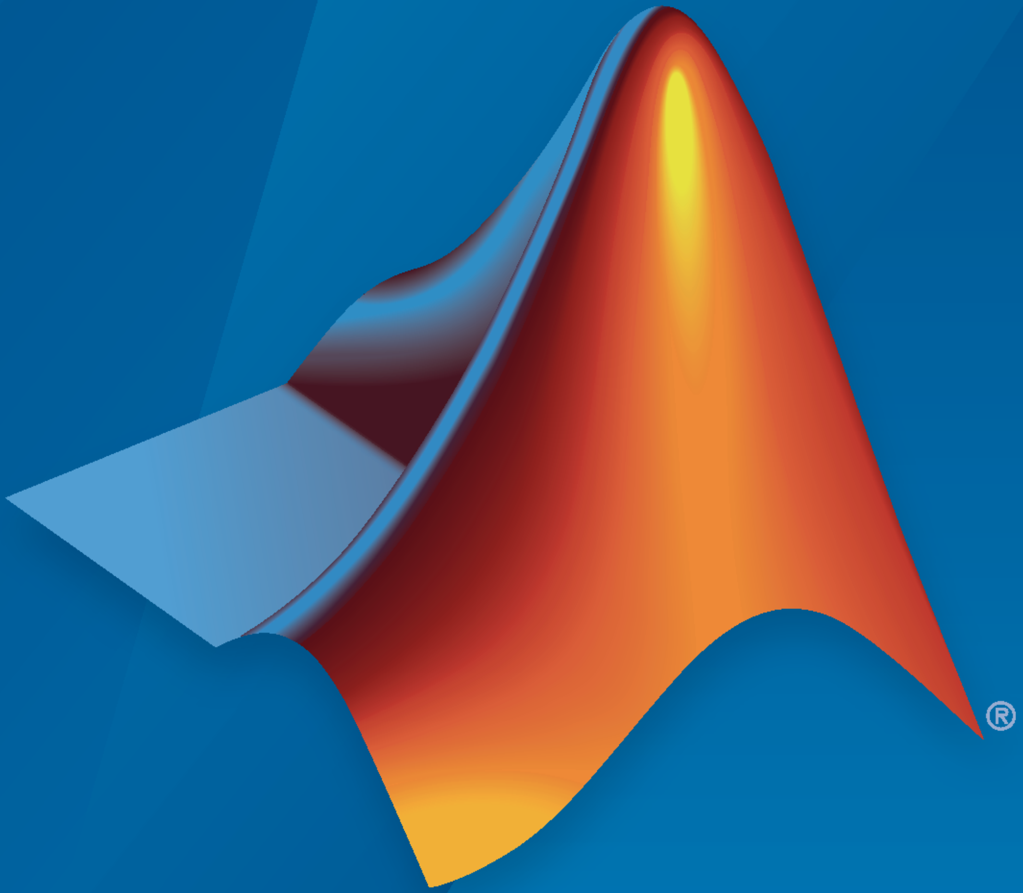
Simscape™ Battery™

Reference



MATLAB® & SIMULINK®

R2023a

MathWorks®

How to Contact MathWorks

Latest news:	`www.mathworks.com`
Sales and services:	`www.mathworks.com/sales_and_services`
User community:	`www.mathworks.com/matlabcentral`
Technical support:	`www.mathworks.com/support/contact_us`
Phone:	508-647-7000

The MathWorks, Inc.
1 Apple Hill Drive
Natick, MA 01760-2098

Simscape™ Battery™ Reference

© COPYRIGHT 2022–2023 by The MathWorks, Inc.

The software described in this document is furnished under a license agreement. The software may be used or copied only under the terms of the license agreement. No part of this manual may be photocopied or reproduced in any form without prior written consent from The MathWorks, Inc.

FEDERAL ACQUISITION: This provision applies to all acquisitions of the Program and Documentation by, for, or through the federal government of the United States. By accepting delivery of the Program or Documentation, the government hereby agrees that this software or documentation qualifies as commercial computer software or commercial computer software documentation as such terms are used or defined in FAR 12.212, DFARS Part 227.72, and DFARS 252.227-7014. Accordingly, the terms and conditions of this Agreement and only those rights specified in this Agreement, shall pertain to and govern the use, modification, reproduction, release, performance, display, and disclosure of the Program and Documentation by the federal government (or other entity acquiring for or through the federal government) and shall supersede any conflicting contractual terms or conditions. If this License fails to meet the government's needs or is inconsistent in any respect with federal procurement law, the government agrees to return the Program and Documentation, unused, to The MathWorks, Inc.

Trademarks

MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See `www.mathworks.com/trademarks` for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

Patents

MathWorks products are protected by one or more U.S. patents. Please see `www.mathworks.com/patents` for more information.

Revision History

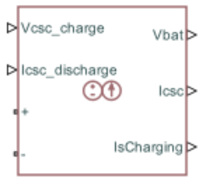
September 2022	Online only	New for Version 1.0 (Release 2022b)
March 2023	Online only	Revised for Version 1.1 (Release 2023a)

Contents

Blocks

Active Interface

Active interface between battery and cell supervisory circuit



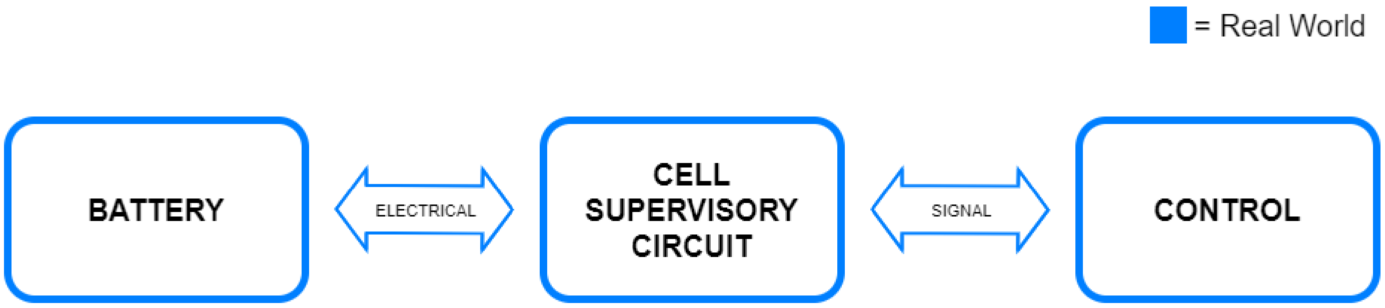
Libraries:
Simscape / Battery / HIL

Description

The Active Interface block provides an abstracted active interface between a battery and a cell supervisory circuit. Use this block with desktop simulations and hardware-in-the-loop battery emulation hardware.

A cell supervisory circuit is an electronic circuit that connects to battery cells or parallel assemblies. You can use this circuit to normalize voltages or states-of-charge (SOC) inside a module or pack. In the real world, you implement this equipment between your battery and the battery management system (BMS) control.

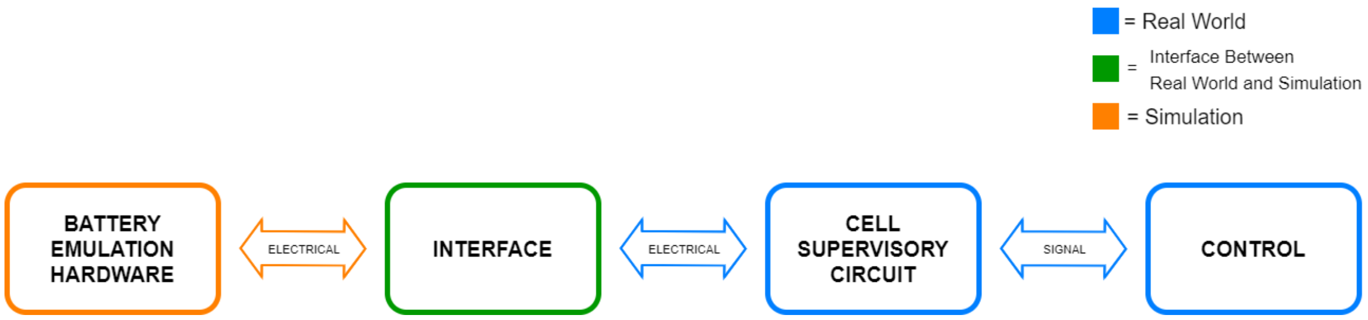
In this figure, the items in blue represent real-world equipment, voltages, and currents.



To avoid damaging your battery and to test your cell balancing algorithms under standard operational conditions and fault conditions, you can use hardware-in-the-loop battery emulation hardware. An abstracted interface facilitates connecting, setting, and configuring the passive balancing BMS to your emulated battery.

In this figure:

- The blue color represents real-world equipment, voltages, and currents.
- The green color represents the interface between the real-world equipment, voltages, and currents, as well as the simulated equipment, voltages, and currents.
- The orange color represents the simulated equipment, voltages, and currents.



The Active Interface block provides the abstract representation of power electronic equipment without the need to model it in detail. For example, if you have a battery pack with different levels of temperature or voltage within its constituent parts, you can use this block to check if the BMS is correctly balancing your pack after a variety of operational scenarios.

If the input signal **Vcsc_charge** is less than the simulated battery voltage, **Vbat**, then:

- The simulated battery is discharging.
- The value at the **IsCharging** output port is `0` (`false`).
- The **Icsc_discharge** input port specifies the current in the cell supervisory circuit.
- The value at the **Icsc** output port is numerically identical to the value at the **Icsc_discharge** input port. This output is a measurement of the current supplied to the cell supervisory circuit.

If the input signal **Vcsc_charge** is greater than or equal to **Vbat**, then:

- The simulated battery is charging.
- The value at the **IsCharging** output port is `1` (`True`).
- The **Vcsc_charge** input port specifies the voltage across the battery minus the voltage drop across the resistance of the cell supervisory circuit.
- The block ignores the **Icsc_discharge** input.
- The **Icsc** output port is a measurement of the current supplied to the cell supervisory circuit. Therefore, when the BMS is charging the battery, this current has a negative value.

SOC Optional Ports

Use the optional SOC ports of the Active Interface block to avoid implementing an SOC estimation algorithm in the BMS, to validate and verify the SOC estimation algorithm against the simulated SOC, or to represent a real-world battery.

- If, in your model, the battery has an SOC output port, expose the SOC input port of this block and connect the ports appropriately. To expose the SOC input port of this block, set the **Expose SOC input port** parameter to `Yes`.
- To avoid implementing an SOC estimation algorithm in the BMS or to validate and verify the SOC estimation algorithm against the simulated SOC, expose the SOC output port of this block and connect it to your BMS. To expose the SOC output port of this block, set the **Expose SOC output port** parameter to `Yes`.
- To represent a real-world battery, do not expose the SOC ports of this block.

Thermal and Temperature Optional Ports

Use the optional thermal and temperature ports of the Active Interface block to represent a battery with a temperature sensor or to validate and verify the temperature estimation algorithm against the simulated temperature.

- If, in your model, the battery has a temperature sensor, expose the thermal port **H** of this block and connect it to the thermal network. To expose the thermal port of this block, set the **Thermal port** parameter to `Model`.
- To validate and verify the temperature estimation algorithm against the simulated temperature, expose both the thermal port **H** and the temperature measurement port **T** of this block and connect them to the thermal network and your BMS, respectively. To expose the temperature measurement port of this block, set the **Expose temperature measurement port** parameter to `Yes`.
- If, in your model, the battery does not include thermal effects or does not have a temperature sensor, do not expose the thermal and temperature ports of this block.

Ports

Input

Vcsc_charge — Charging voltage from cell supervisory circuit, V
physical signal

Charging voltage from the cell supervisory circuit, in volt.

Icsc_discharge — Discharge current from cell supervisory circuit, A
physical signal

Discharge current from the cell supervisory circuit, in ampere.

SOC — Battery state-of-charge, unitless
physical signal

State-of-charge of the battery, in percent.

Dependencies

To enable this port, set the **Expose SOC input port** parameter to `Yes`.

Output

Vbat — Battery voltage, V
physical signal

Voltage across the battery, in volt.

Icsc — Cell supervisory circuit current, A
physical signal

Current flowing through the cell supervisory circuit, in ampere.

Dependencies

To enable this port, set the **Expose cell supervisory circuit current measurement port** parameter to `Yes`.

isCharging — Charging status, unitless
physical signal

Physical signal output port that indicates whether the BMS is charging or discharging the battery. If the value of this output is `1`, the BMS is charging the battery. If the value of this output is `0`, the BMS is discharging the battery.

SOC — Battery state-of-charge, unitless
physical signal

State-of-charge of the battery, in percent.

Dependencies

To enable this port, set the **Expose SOC output port** parameter to `Yes`.

T — Temperature measurement, K
physical signal

Temperature of the battery, in kelvin.

Dependencies

To enable this port, set the **Expose temperature measurement port** parameter to `Yes`.

Conserving

+ — Positive terminal
electrical

Electrical conserving port associated with the positive terminal of the interface.

- — Negative terminal
electrical

Electrical conserving port associated with the negative terminal of the interface.

H — Thermal port
thermal

Thermal conserving port.

Dependencies

To enable this port, set the **Thermal port** parameter to `Model`.

Parameters

Cell supervisory circuit resistance — Cell supervisory circuit resistance
`1e-6 Ohm` (default)

Representative resistance of the cell supervisory circuit that ensures a small difference between the values at the **Vcsc_discharge** and **Vbat** ports in the abstracted equations. This voltage difference ensures a smooth transition between the charging and discharging modes.

Expose SOC input port — SOC input port visibility
`No` (default) | `Yes`

Option to expose the battery SOC input port.

Expose SOC output port — SOC output port visibility
`Yes` (default) | `No`

Option to expose the battery SOC output port.

Dependencies

To enable this parameter, set **Expose SOC input port** to `Yes`.

Thermal port — Thermal port visibility
`Omit` (default) | `Model`

Option to model the thermal port of the passive interface.

Expose temperature measurement port — Temperature measurement port visibility
`Yes` (default) | `No`

Option to expose the port that measures the temperature of the battery.

Dependencies

To enable this parameter, set **Thermal port** to `Model`.

Version History

Introduced in R2022b

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

See Also

Passive Interface

Topics

"Real-Time Model Preparation Workflow"
"Real-Time Simulation Workflow"
"Hardware-in-the-Loop Simulation Workflow"

arrayOfThermalNodesConnector

Concatenate arrays of thermal nodes into single array of thermal nodes port

Libraries:
Simscape / Battery / Thermal

Description

The arrayOfThermalNodesConnector block concatenates different arrays of thermal nodes into a single array of thermal nodes. This block acts as a mux for the Simscape™ thermal domain by concatenating all input array of nodes into a single 1-D array.

To specify the number of arrays of thermal nodes that you want to concatenate, set the **numInputConnections** parameter. This action enables a number of ports and parameters equal to the value of the **numInputConnections** parameter. This block supports from one to 50 different connections from different battery modules with different `ThermalNodes` properties.

To specify the number of elements in each connection node, set the **inNodekNumElements** parameter, where *k* is an integer in the range [1, **numInputConnections**].

Ports

Conserving

inConnk — Array of thermal nodes *k* to concatenate
thermal array-of-nodes

Array of thermal nodes *k* that you want to concatenate, where *k* is an integer in the range [1, **numInputConnections**]. The number of ports of this type is equal to the value to which you set the **numInputConnections** parameter.

outConn — Concatenated array of thermal nodes
thermal array-of-nodes

Concatenated array of thermal nodes.

Parameters

numInputConnections — Number of array of thermal nodes to concatenate
`one` (default) | `two` | `three` | `four` | ...

Number of array of thermal nodes that you want to concatenate. You can specify any value between `one` and `fifty`.

inNodekNumElements — Number of elements in connection node *k*
`1` (default)

Number of elements in connection node *k*, where *k* is an integer in the range [1, **numInputConnections**]. The number of parameters of this type is equal to the value to which you set the **numInputConnections** parameter.

Version History

Introduced in R2023a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

See Also

Simscape Blocks

Edge Cooling | Parallel Channels | U-shaped Channels

Objects

`Cell` | `ParallelAssembly` | `Module` | `ModuleAssembly` | `Pack`

Functions

`buildBattery`

Topics

"Connect Cooling Plate to Battery Blocks"

Battery

Behavioral battery model

Libraries:
Simscape / Electrical / Sources
Simscape / Battery / Cells

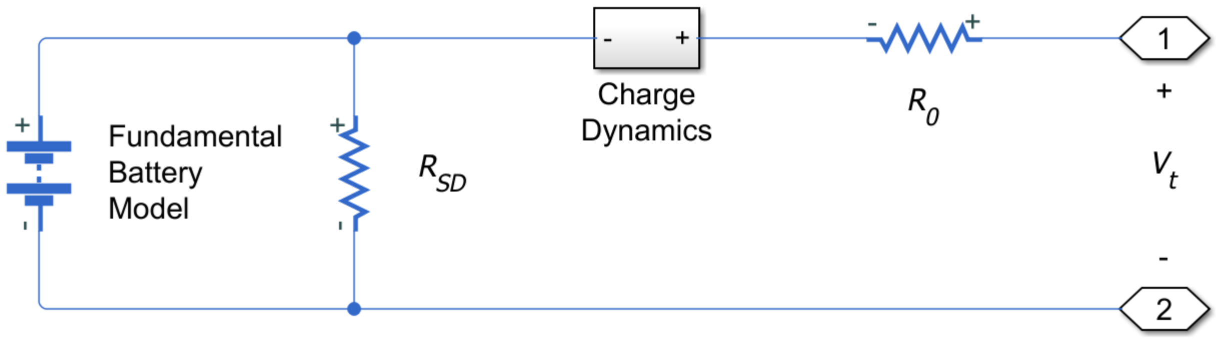
Description

The Battery block represents a simple battery model. You can also expose the charge output port and the thermal port of the battery.

To measure the internal charge level of the battery, in the **Main** section, set the **Expose charge measurement port** to `Yes`. This action exposes an extra physical signal port that outputs the internal state of charge. Use this functionality to change load behavior as a function of state of charge, without the complexity of building a charge state estimator.

To simulate the thermal effects of the battery, in the **Thermal Port** section, set the **Thermal port** parameter to `Model`. This action exposes an extra thermal port, which represents the battery thermal mass. When you select this option, provide additional parameters to define battery behavior at a second temperature. For more information, see “Modeling Thermal Effects” on page 1-11.

The battery equivalent circuit is made up of the fundamental battery model, the self-discharge resistance R_{SD}, the charge dynamics model, and the series resistance R_0.



Battery Model

If you select `Infinite` for the **Battery charge capacity** parameter, the block models the battery as a series resistor and a constant voltage source. If you select `Finite` for the **Battery charge capacity** parameter, the block models the battery as a series resistor and a charge-dependent voltage source. In the finite case, the voltage is a function of charge and has the following relationship:

$$V = V_0\left(\frac{\mathrm{SOC}}{1 - \beta(1 - \mathrm{SOC})}\right)$$

where:

- SOC (state-of-charge) is the ratio of current charge to rated battery capacity.
- V_0 is the voltage when the battery is fully charged at no load, as defined by the **Nominal voltage, Vnom** parameter.
- β is a constant that is calculated so that the battery voltage is *V1* when the charge is *AH1*. Specify the voltage *V1* and cell capacity*AH1* using block parameters. *AH1* is the charge when the no-load (open-circuit) voltage is *V1*, and *V1* is less than the nominal voltage.

The equation defines an approximate relationship between voltage and remaining charge. This approximation replicates the increasing rate of voltage drop at low charge values, and ensures that the battery voltage becomes zero when the charge level is zero. The advantage of this model is that it requires few parameters, which are readily available on most datasheets.

Modeling Battery Fade

For battery models with finite battery charge capacity, you can model battery performance deterioration depending on the number of discharge cycles. This deterioration is referred to as battery fade. To enable battery fade, set the **Battery fade** parameter to `Enabled`. This setting exposes additional parameters in the **Fade** section.

The block implements battery fade by scaling certain battery parameter values that you specify in the **Main** section, depending on the number of completed discharge cycles. The block uses multipliers λ_{AH}, λ_{R0}, and λ_{V1} on the **Cell capacity (Ah rating)**, **Internal resistance**, and **Voltage V1 when charge is AH1** parameter values, respectively. These multipliers, in turn, depend on the number of discharge cycles:

$$\lambda_{AH} = 1 - k_1 N^{0.5}$$

$$\lambda_{R0} = 1 + k_2 N^{0.5}$$

$$\lambda_{V1} = 1 - k_3 N$$

$$N = N_0 + \frac{1}{AH}\int_0^t \frac{i(t) \cdot H(i(t))}{\lambda_{AH}(t)} dt$$

where:

- λ_{AH} is the multiplier for battery nominal capacity.
- λ_{R0} is the multiplier for battery series resistance.
- λ_{V1} is the multiplier for voltage *V1*.
- *N* is the number of discharge cycles completed.
- N_0 is the number of full discharge cycles completed before the start of the simulation.
- *AH* is the rated battery capacity in ampere-hours.
- *i(t)* is the instantaneous battery output current.
- *H(i(t))* is the Heaviside function of the instantaneous battery output current. This function returns 0 if the argument is negative, and 1 if the argument is positive.

The block calculates the coefficients k_1, k_2, and k_3 by substituting the parameter values you provide in the **Fade** section into these battery equations. For example, the default set of block parameters corresponds to the following coefficient values:

- k_1 = 1e-2
- k_2 = 1e-3
- k_3 = 1e-3

You can also define a starting point for a simulation based on the previous charge-discharge history by using the high-priority variable **Discharge cycles**. For more information, see “Variables” (Simscape Electrical).

Modeling Thermal Effects

If you set the **Thermal port** parameter to `Model`, you must provide additional parameters to define battery behavior at a second temperature. The extended equations for the voltage when you expose the thermal port are:

$$V = V_{0T}\left(\frac{\mathrm{SOC}}{1-\beta_T(1-\mathrm{SOC})}\right)$$

$$V_{0T} = V_0(1 + \lambda_V(T - T_1))$$

where:

- T is the battery temperature.
- T_1 is the nominal measurement temperature.
- λ_V is the parameter temperature dependence coefficient for V_0.
- $\beta_T = \beta\left[1 + \lambda_\beta(T - T_1)\right]$.
- λ_β is the parameter temperature dependence coefficient for β.
- β is calculated in the same way as “Battery Model” on page 1-9, using the temperature-modified nominal voltage V_{0T}.

The internal series resistance, self-discharge resistance, and any charge-dynamic resistances are also functions of temperature:

$$R_T = R(1 + \lambda_R(T - T_1))$$

where λ_R is the parameter temperature dependence coefficient.

All the temperature dependence coefficients are determined from the corresponding values you provide at the nominal and second measurement temperatures. If you include charge dynamics in the model, the time constants vary with temperature in the same way.

The battery temperature is determined from a summation of all the ohmic losses included in the model:

$$M_{th}\dot{T} = \sum_i V_{T,i}{}^2/R_{T,i}$$

where:

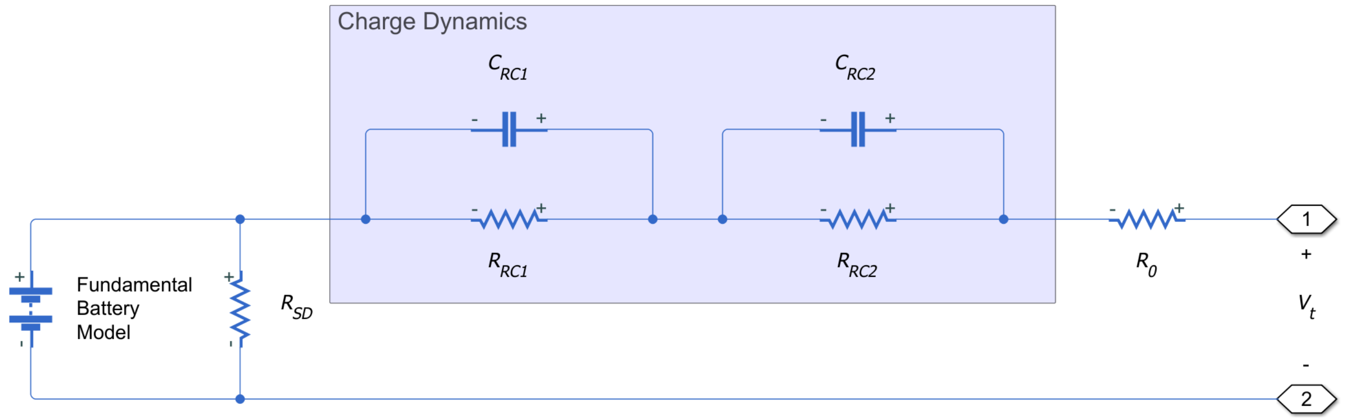
- M_{th} is the battery thermal mass.
- *i* corresponds to the *i*th ohmic loss contributor. Depending on how you have configured the block, the losses include:
  - Series resistance
  - Self-discharge resistance
  - First charge dynamics segment
  - Second charge dynamics segment
  - Third charge dynamics segment
  - Fourth charge dynamics segment
  - Fifth charge dynamics segment
- $V_{T,i}$ is the voltage drop across resistor *i*.
- $R_{T,i}$ is resistor *i*.

Modeling Charge Dynamics

You can model battery charge dynamics by using the **Charge dynamics** parameter:

- `No dynamics` — The equivalent circuit contains no parallel RC sections. There is no delay between terminal voltage and internal charging voltage of the battery.
- `One time-constant dynamics` — The equivalent circuit contains one parallel RC section. Specify the time constant using the **First time constant** parameter.
- `Two time-constant dynamics` — The equivalent circuit contains two parallel RC sections. Specify the time constants using the **First time constant** and **Second time constant** parameters.
- `Three time-constant dynamics` — The equivalent circuit contains three parallel RC sections. Specify the time constants using the **First time constant**, **Second time constant**, and **Third time constant** parameters.
- `Four time-constant dynamics` — The equivalent circuit contains four parallel RC sections. Specify the time constants using the **First time constant**, **Second time constant**, **Third time constant**, and **Fourth time constant** parameters.
- `Five time-constant dynamics` — The equivalent circuit contains five parallel RC sections. Specify the time constants using the **First time constant**, **Second time constant**, **Third time constant**, **Fourth time constant**, and **Fifth time constant** parameters.

This figure shows the equivalent circuit for the block configured with two time-constant dynamics.



In the diagram:

- R_{RC1} and R_{RC2} are the parallel RC resistances. Specify these values with the **First polarization resistance** and **Second polarization resistance** parameters, respectively.
- C_{RC1} and C_{RC2} are the parallel RC capacitances. The time constant τ for each parallel section relates the R and C values using the relationship $C = \tau/R$. Specify τ for each section using the **First time constant** and **Second time constant** parameters, respectively.
- R_0 is the series resistance. Specify this value with the **Internal resistance** parameter.

Modeling Battery Aging

For battery models with finite battery charge capacity, you can model the battery performance deterioration that occurs when the battery is not used. Calendar aging affects both the internal resistance and capacity. In particular, the resistance increase depends by various mechanisms such as the creation of Solid Electrolyte Interface (SEI) at both anode and cathode and the corrosion of the current collector. These processes mainly depends on the storage temperature, the storage state of charge, and time.

Note The Battery block only applies the calendar aging during initialization. When you set the **Calendar aging** parameter to `Enabled`, the block exposes the **Vector of time intervals** parameter that represents the time the battery has aged before the start of the simulation. Calendar aging during the simulation is not covered with this option.

This equation defines the terminal resistance increase of the battery due to calendar aging:

$$\alpha_r(T, V_{oc}) = (bV_{oc} - c)e^{-\frac{qd}{kT}},$$

$$R = R_0\left(1 + \sum_{i=1}^{i=n} \alpha_r(T_i, V_{oc})\left(t_i^a - t_{i-1}^a\right)\right),$$

where:

- V_{oc} is the **Normalized open-circuit voltage during storage, V/Vnom**.
- R_0 is the **Internal resistance**.
- t_i is the time sample derived from the **Vector of time intervals** parameter.

- T_i is derived from the **Vector of temperatures** parameter.
- b is the **Linear scaling for voltage, b**.
- c is the **Constant offset for voltage, c**.
- d is the **Temperature-dependent exponential increase, d**.
- a is the **Time exponent, a**.
- q is the electron's elementary charge, in C.
- k is the Boltzmann constant, in J/K.

The *R1_age_multiplier* variable in the **Simscape Results Explorer** stores the calendar aging of the battery in terms of resistance increase:

$$R1\_age\_multiplier = \frac{R}{R_0}.$$

For thermal modeling options of the block, if you set the **Storage condition** parameter to `Fixed open-circuit voltage`, you must specify the additional **Open-circuit voltage measurement temperature** parameter to convert the storage open-circuit voltage to the temperature-independent state of charge during storage:

$$SOC = \frac{OCV\left(1 - \beta\left(T_{OCV_{measurement}}\right)\right)}{1 - \beta\left(T_{OCV_{measurement}}\right)OCV}.$$

The open-circuit voltage according to the storage temperature is then defined by this equation:

$$OCV_{T_{storage}} = \frac{SOC}{1 - \beta\left(T_{storage}\right)\left(1 - \beta\left(T_{storage}\right)SOC\right)}.$$

Finally, this equation defines the terminal resistance increase of the battery due to calendar aging according to the storage temperature:

$$R = R_0\left(1 + \sum_{i=1}^{i=n} \alpha_r\left(T_{storage_i}, OCV_{T_{storage_i}}\right)\left(t_i^a - t_{i-1}^a\right)\right).$$

Plotting Voltage-Charge Characteristics

A quick plot feature lets you visualize the voltage-charge characteristic for the battery model parameter values. To plot the characteristics, right-click a Battery block in your model and, from the context menu, select **Electrical > Basic characteristic**. The software automatically computes a set of bias conditions, based on the block parameter values, and opens a figure window containing a plot of no-load voltage versus the state-of-charge (SOC) for the block. For more information, see “Plot Basic Characteristics for Battery Blocks” (Simscape Electrical).

Variables

To set the priority and initial target values for the block variables prior to simulation, use the **Initial Targets** section in the block dialog box or Property Inspector. For more information, see “Set Priority and Initial Target for Block Variables”.

Nominal values provide a way to specify the expected magnitude of a variable in a model. Using system scaling based on nominal values increases the simulation robustness. Nominal values can come from different sources, one of which is the **Nominal Values** section in the block dialog box or Property Inspector. For more information, see “System Scaling by Nominal Values”.

When you model battery fade, the **Discharge cycles** variable lets you specify the number of charge-discharge cycles completed prior to the start of simulation. If you disable battery fade modeling, this variable is not used by the block.

Assumptions and Limitations

- The self-discharge resistance is assumed not to depend strongly on the number of discharge cycles.
- For the thermal modeling option of the battery, you provide fade data only for the reference temperature operation. The block applies the same derived λ_{AH}, λ_{R0}, and λ_{V1} multipliers to parameter values corresponding to the second temperature.
- When using the thermal block modeling options, use caution when operating at temperatures outside of the temperature range bounded by the **Measurement temperature** and **Second measurement temperature** values. The block uses linear interpolation for the derived equation coefficients, and simulation results can become nonphysical outside of the specified range. The block checks that the internal series resistance, self-discharge resistance, and nominal voltage always remain positive. If there is a violation, the block issues error messages.

Ports

Output

q — Battery charge level, C
physical signal

Physical signal port that outputs the internal charge, in the units of coulomb (C). Use this output port to change load behavior as a function of charge, without the complexity of building a charge state estimator.

Dependencies

To enable this port, set **Expose charge measurement port** to `Yes`.

Conserving

+ — Positive terminal
electrical

Electrical conserving port associated with the battery positive terminal.

- — Negative terminal
electrical

Electrical conserving port associated with the battery negative terminal.

H — Battery thermal mass
thermal

Thermal conserving port that represents the battery thermal mass. When you expose this port, provide additional parameters to define battery behavior at a second temperature. For more information, see “Modeling Thermal Effects” on page 1-11.

Dependencies

To enable this port, in the **Thermal Port** section, set **Thermal port** to `Model`.

Parameters

Main

Nominal voltage, Vnom — Output voltage when battery is fully charged
`12 V` (default) | positive number

No-load voltage across the battery when it is fully charged.

Current directionality — Enable direction of current
`Disabled` (default) | `Enabled`

Whether to enable current directionality. If you set this parameter to `Enabled`, the terminal resistance depends on the direction of the current.

Internal resistance — Battery internal resistance
`2 Ohm` (default) | positive number

Internal connection resistance of the battery.

Dependencies

To enable this parameter, set **Current directionality** to `Disabled`.

Internal resistance during charging — Battery internal resistance during charge
`2 Ohm` (default) | positive number

Internal connection resistance of the battery during charge phase.

Dependencies

To enable this parameter, set **Current directionality** to `Enabled`.

Internal resistance during discharging — Battery internal resistance during discharge
`2 Ohm` (default) | positive number

Internal connection resistance of the battery during discharge phase.

Dependencies

To enable this parameter, set **Current directionality** to `Enabled`.

Battery charge capacity — Select battery model
`Infinite` (default) | `Finite`

Select one of the options for modeling the charge capacity of the battery:

- `Infinite` — The battery voltage is independent of charge drawn from the battery.
- `Finite` — The battery voltage decreases as charge decreases.

Cell capacity (Ah rating) — Nominal battery capacity when fully charged
`50 hr*A` (default) | positive number

Maximum (nominal) battery charge in ampere-hours. To specify a target value for the initial battery charge at the start of simulation, use the high-priority **Charge** variable. For more information, see “Variables” (Simscape Electrical).

Dependencies

To enable this parameter, set **Battery charge capacity** to `Finite`.

Voltage V1 when charge is AH1 — Output voltage at charge level AH1
`11.5 V` (default) | positive number

Fundamental battery output voltage when the charge level is AH1, as specified by the **Charge AH1 when no-load voltage is V1** parameter. This parameter must be less than **Nominal voltage, Vnom**.

Dependencies

To enable this parameter, set **Battery charge capacity** to `Finite`.

Charge AH1 when no-load voltage is V1 — Charge level when the no-load output voltage is V1
`25 hr*A` (default) | positive number

Battery charge level corresponding to the no-load output voltage specified by the **Voltage V1 when charge is AH1** parameter.

Dependencies

To enable this parameter, set **Battery charge capacity** to `Finite`.

Self-discharge — Select whether to model the self-discharge resistance of the battery
`Disabled` (default) | `Enabled`

Select whether to model the self-discharge resistance of the battery. The block models this effect as a resistor across the terminals of the fundamental battery model.

As temperature increases, self-discharge resistance decreases, causing self-discharge to increase. If the decrease in resistance is too fast, thermal runaway of the battery and numerical instability can occur. You can resolve this by doing any of the following:

- Decrease the thermal resistance
- Decrease the gradient of the self-discharge resistance with respect to temperature
- Increase the self-discharge resistance

Dependencies

To enable this parameter, set **Battery charge capacity** to `Finite`.

Self-discharge resistance — Resistance that represents battery self-discharge
`2000 Ohm` (default) | positive number

Resistance across the fundamental battery model that represents battery self-discharge.

Dependencies

To enable this parameter, set **Self-discharge** to `Enabled`.

Measurement temperature — Temperature at which the block parameters are measured
`298.15 K` (default) | positive number

Temperature T_1, at which the block parameters in the **Main** section are measured. For more information, see “Modeling Thermal Effects” on page 1-11.

Dependencies

To enable this parameter, in the **Thermal Port** section, set **Thermal port** to `Model`.

Expose charge measurement port — Whether to expose charge measurement port
`No` (default) | `Yes`

Whether to expose the charge measurement port and measure the internal charge level of the battery.

Dynamics

Charge dynamics — Battery charge dynamics model
`No dynamics` (default) | `One time-constant dynamics` | `Two time-constant dynamics` | `Three time-constant dynamics` | `Four time-constant dynamics` | `Five time-constant dynamics`

Select how to model battery charge dynamics. This parameter determines the number of parallel RC sections in the equivalent circuit:

- `No dynamics` — The equivalent circuit contains no parallel RC sections. There is no delay between terminal voltage and internal charging voltage of the battery.
- `One time-constant dynamics` — The equivalent circuit contains one parallel RC section. Specify the time constant using the **First time constant** parameter.
- `Two time-constant dynamics` — The equivalent circuit contains two parallel RC sections. Specify the time constants using the **First time constant** and **Second time constant** parameters.
- `Three time-constant dynamics` — The equivalent circuit contains three parallel RC sections. Specify the time constants using the **First time constant**, **Second time constant**, and **Third time constant** parameters.
- `Four time-constant dynamics` — The equivalent circuit contains four parallel RC sections. Specify the time constants using the **First time constant**, **Second time constant**, **Third time constant**, and **Fourth time constant** parameters.
- `Five time-constant dynamics` — The equivalent circuit contains five parallel RC sections. Specify the time constants using the **First time constant**, **Second time constant**, **Third time constant**, **Fourth time constant**, and **Fifth time constant** parameters.

First polarization resistance — First RC resistance
`0.005 Ohm` (default) | positive number

Resistance of the first parallel RC section. This parameter primarily affects the ohmic losses of the RC section.

Dependencies

To enable this parameter, set **Charge dynamics** to `One time-constant dynamics`, `Two time-constant dynamics`, `Three time-constant dynamics`, `Four time-constant dynamics`, or `Five time-constant dynamics`.

First time constant — First RC time constant
`30 s` (default) | positive number

Time constant of the first parallel RC section. This value is equal to *RC* and affects the dynamics of the RC section.

Dependencies

To enable this parameter, set **Charge dynamics** to `One time-constant dynamics`, `Two time-constant dynamics`, `Three time-constant dynamics`, `Four time-constant dynamics`, or `Five time-constant dynamics`.

Second polarization resistance — Second RC resistance
`0.005 Ohm` (default) | positive number

Resistance of the second parallel RC section. This parameter primarily affects the ohmic losses of the RC section.

Dependencies

To enable this parameter, set **Charge dynamics** to `Two time-constant dynamics`, `Three time-constant dynamics`, `Four time-constant dynamics`, or `Five time-constant dynamics`.

Second time constant — Second RC time constant
`30 s` (default) | positive number

Time constant of the second parallel RC section. This value is equal to *RC* and affects the dynamics of the RC section.

Dependencies

To enable this parameter, set **Charge dynamics** to `Two time-constant dynamics`, `Three time-constant dynamics`, `Four time-constant dynamics`, or `Five time-constant dynamics`.

Third polarization resistance — Third RC resistance
`0.005 Ohm` (default) | positive number

Resistance of the third parallel RC section. This parameter primarily affects the ohmic losses of the RC section.

Dependencies

To enable this parameter, set **Charge dynamics** to `Three time-constant dynamics`, `Four time-constant dynamics`, or `Five time-constant dynamics`.

Third time constant — Third RC time constant
`30 s` (default) | positive number

Time constant of the third parallel RC section. This value is equal to *RC* and affects the dynamics of the RC section.

Dependencies

To enable this parameter, set **Charge dynamics** to `Three time-constant dynamics`, `Four time-constant dynamics`, or `Five time-constant dynamics`.

Fourth polarization resistance — Fourth RC resistance
`0.005 Ohm` (default) | positive number

Resistance of the fourth parallel RC section. This parameter primarily affects the ohmic losses of the RC section.

Dependencies

To enable this parameter, set **Charge dynamics** to `Four time-constant dynamics` or `Five time-constant dynamics`.

Fourth time constant — Fourth RC time constant
`30 s` (default) | positive number

Time constant of the fourth parallel RC section. This value is equal to *RC* and affects the dynamics of the RC section.

Dependencies

To enable this parameter, set **Charge dynamics** to `Four time-constant dynamics` or `Five time-constant dynamics`.

Fifth polarization resistance — Fifth RC resistance
`0.005 Ohm` (default) | positive number

Resistance of the fifth parallel RC section. This parameter primarily affects the ohmic losses of the RC section.

Dependencies

To enable this parameter, set **Charge dynamics** to `Five time-constant dynamics`.

Fifth time constant — Fifth RC time constant
`30 s` (default) | positive number

Time constant of the fifth parallel RC section. This value is equal to *RC* and affects the dynamics of the RC section.

Dependencies

To enable this parameter, set **Charge dynamics** to `Five time-constant dynamics`.

Fade

Battery fade — Select whether to model battery performance deterioration with aging
`Disabled` (default) | `Enabled`

Select whether to include battery fade modeling:

- `Disabled` — The battery performance is not age dependent.
- `Enabled` — The battery performance changes depending on the number of completed charge-discharge cycles. Selecting this option exposes additional parameters in this section, which define the battery performance after a certain number of discharge cycles. The block uses these parameter values to calculate the scaling coefficients k_1, k_2, and k_3. For more information, see “Modeling Battery Fade” on page 1-10.

Dependencies

To enable this parameter, in the **Main** section, set **Battery charge capacity** to `Finite`. If **Battery charge capacity** is `Infinite`, the **Fade** section is empty.

Number of discharge cycles, N — Number of cycles that defines a second set of data points
`100` (default) | positive number

Number of charge-discharge cycles after which the other parameters in this section are measured. This second set of data points defines the scaling coefficients k_1, k_2, and k_3, used in modeling battery fade.

Dependencies

To enable this parameter, set **Battery fade** to `Enabled`.

Cell capacity after N discharge cycles — Maximum battery capacity after N discharge cycles
`45 hr*A` (default) | positive number

Maximum battery charge, in ampere-hours, after the number of discharge cycles specified by the **Number of discharge cycles, N** parameter.

Dependencies

To enable this parameter, set **Battery fade** to `Enabled`.

Internal resistance after N discharge cycles — Battery internal resistance after N discharge cycles
`2.02 Ohm` (default) | positive number

Battery internal resistance after the number of discharge cycles specified by the **Number of discharge cycles, N** parameter.

Dependencies

To enable this parameter, set **Battery fade** to `Enabled` and, in the **Main** section, set **Current Directionality** to `Disabled`.

Average internal resistance after N discharge cycles — Battery average internal resistance after N discharge cycles
`2.02 Ohm` (default) | positive number

Average of the battery charge and discharge internal resistance after the number of discharge cycles specified by the **Number of discharge cycles, N** parameter.

Dependencies

To enable this parameter, set **Battery fade** to `Enabled` and, in the **Main** section, set **Current Directionality** to `Enabled`.

Voltage V1 at charge AH1 after N discharge cycles — Output voltage at charge level AH1 after N discharge cycles
`10.35 V` (default) | positive number

Fundamental battery model output voltage, at charge level AH1, after the number of discharge cycles specified by the **Number of discharge cycles, N** parameter.

Dependencies

To enable this parameter, set **Battery fade** to `Enabled`.

Calendar Aging

To expose these parameters, in the **Main** tab, set **Battery charge capacity** to `Finite`.

Calendar aging — Calendar aging option
`Disabled` (default) | `Enabled`

Whether to enable calendar aging of the battery.

Storage condition — Storage condition
`Fixed open-circuit voltage` (default) | `Fixed state of charge`

Whether to specify the open-circuit voltage or the state of charge during storage.

Dependencies

To enable this parameter, set **Calendar aging** to `Enabled`.

Normalized open-circuit voltage during storage, V/Vnom — Normalized open-circuit voltage during storage
`0.9` (default) | scalar

Normalized open-circuit voltage during storage.

Dependencies

To enable this parameter, set **Calendar aging** to `Enabled` and **Storage condition** to `Fixed open-circuit voltage`.

Open-circuit measurement temperature — Open-circuit temperature
`298.15` K (default) | positive number

Open-circuit measurement temperature.

Dependencies

To enable this parameter, expose the block thermal port and set **Calendar aging** to `Enabled` and **Storage condition** to `Fixed open-circuit voltage`.

State of charge during storage (%) — Percentage state of charge during storage
`60` (default) | positive number

State of charge during storage, in percentage.

Dependencies

To enable this parameter, set **Calendar aging** to `Enabled` and **Storage condition** to `Fixed state of charge`.

Vector of time intervals — Time intervals
`[0]` d (default) | scalar

Time intervals. This parameter must be equal in size to **Vector of temperatures**.

Dependencies

To enable this parameter, set **Calendar aging** to `Enabled`.

Vector of temperatures — Storage temperatures
`[273]` K (default) | scalar

Set of storage temperatures. This parameter must be equal in size to **Vector of time intervals**.

Dependencies

To enable this parameter, set **Calendar aging** to `Enabled`.

Linear scaling for voltage, b — Voltage linear scaling
`2.2134e6` (default) | scalar

Linear scaling coefficient for open-circuit voltage.

Dependencies

To enable this parameter, set **Calendar aging** to `Enabled`.

Constant offset for voltage, c — Voltage constant offset
`1.632e6` (default) | scalar

Constant offset for open-circuit voltage.

Dependencies

To enable this parameter, set **Calendar aging** to `Enabled`.

Temperature-dependent exponential increase, d — Temperature-dependent exponential increase
`0.515833569 V` (default) | scalar

Temperature-dependent exponential increase.

Dependencies

To enable this parameter, set **Calendar aging** to `Enabled`.

Time exponent, a — Time exponent
`0.75` (default) | scalar

Time exponent.

Dependencies

To enable this parameter, set **Calendar aging** to `Enabled`.

Temperature Dependence

This section appears only for blocks with exposed thermal port. To expose the thermal port, in the **Thermal Port** section, set **Thermal port** to `Model`. For more information, see "Modeling Thermal Effects" on page 1-11.

Nominal voltage at second measurement temperature — Output voltage when battery is fully charged
`12 V` (default) | positive number

No-load voltage across the battery at the second measurement temperature when it is fully charged.

Internal resistance at second measurement temperature — Battery internal resistance
`2.2 Ohm` (default) | positive number

Internal connection resistance of the battery at the second measurement temperature.

Voltage V1 at second measurement temperature — Output voltage at charge level AH1
`11.4 V` (default) | positive number

Fundamental battery model output voltage at the second measurement temperature and at charge level AH1, as specified by the **Charge AH1 when no-load voltage is V1** parameter.

Dependencies

To enable this parameter, in the **Main** section, set **Battery charge capacity** parameter to `Finite`.

Self-discharge resistance at second measurement temperature — Resistance that represents battery self-discharge
`2200 Ohm` (default) | positive number

Resistance across the fundamental battery model at the second measurement temperature. This resistance represents the self-discharge.

Dependencies

To enable this parameter, in the **Main** section, set **Self-discharge resistance** parameter to `Enabled`.

First polarization resistance at second measurement temperature — First RC resistance at second measurement temperature
`0.005 Ohm` (default) | positive number

Resistance of the first parallel RC section at the second measurement temperature.

Dependencies

To enable this parameter, set **Charge dynamics** to `One time-constant dynamics`, `Two time-constant dynamics`, `Three time-constant dynamics`, `Four time-constant dynamics`, or `Five time-constant dynamics`.

First time constant at second measurement temperature — First RC time constant at second measurement temperature
`30 s` (default) | positive number

Time constant of the first parallel RC section at the second measurement temperature.

Dependencies

To enable this parameter, set **Charge dynamics** to `One time-constant dynamics`, `Two time-constant dynamics`, `Three time-constant dynamics`, `Four time-constant dynamics`, or `Five time-constant dynamics`.

Second polarization resistance at second measurement temperature — Second RC resistance at second measurement temperature
`0.005 Ohm` (default) | positive number

Resistance of the second parallel RC section at the second measurement temperature.

Dependencies

To enable this parameter, set **Charge dynamics** to `Two time-constant dynamics`, `Three time-constant dynamics`, `Four time-constant dynamics`, or `Five time-constant dynamics`.

Second time constant at second measurement temperature — Second RC time constant at second measurement temperature
`30 s` (default) | positive number

Time constant of the second parallel RC section at the second measurement temperature.

Dependencies

To enable this parameter, set **Charge dynamics** to `Two time-constant dynamics`, `Three time-constant dynamics`, `Four time-constant dynamics`, or `Five time-constant dynamics`.

Third polarization resistance at second measurement temperature — Third RC resistance at second measurement temperature
`0.005 Ohm` (default) | positive number

Resistance of the third parallel RC section at the second measurement temperature.

Dependencies

To enable this parameter, set **Charge dynamics** to `Three time-constant dynamics`, `Four time-constant dynamics`, or `Five time-constant dynamics`.

Third time constant at second measurement temperature — Third RC time constant at second measurement temperature
`30 s` (default) | positive number

Time constant of the third parallel RC section at the second measurement temperature.

Dependencies

To enable this parameter, set **Charge dynamics** to `Three time-constant dynamics`, `Four time-constant dynamics`, or `Five time-constant dynamics`.

Fourth polarization resistance at second measurement temperature — Fourth RC resistance at second measurement temperature
`0.005 Ohm` (default) | positive number

Resistance of the fourth parallel RC section at the second measurement temperature.

Dependencies

To enable this parameter, set **Charge dynamics** to `Four time-constant dynamics` or `Five time-constant dynamics`.

Fourth time constant at second measurement temperature — Fourth RC time constant at second measurement temperature
`30 s` (default) | positive number

Time constant of the fourth parallel RC section at the second measurement temperature.

Dependencies

To enable this parameter, set **Charge dynamics** to `Four time-constant dynamics` or `Five time-constant dynamics`.

Fifth polarization resistance at second measurement temperature — Fifth RC resistance at second measurement temperature
`0.005 Ohm` (default) | positive number

Resistance of the fifth parallel RC section at the second measurement temperature.

Dependencies

To enable this parameter, set **Charge dynamics** to `Five time-constant dynamics`.

Fifth time constant at second measurement temperature — Fifth RC time constant at second measurement temperature
`30 s` (default) | positive number

Time constant of the fifth parallel RC section at the second measurement temperature.

Dependencies

To enable this parameter, set **Charge dynamics** to `Five time-constant dynamics`.

Second measurement temperature — Temperature at which the block parameters in this section are measured
`273.15 K` (default) | positive number

Temperature T_2, at which the block parameters in the **Temperature Dependence** section are measured. For more information, see "Modeling Thermal Effects" (Simscape Electrical).

To specify the initial temperature at the start of simulation, use the high-priority **Temperature** variable. For more information, see "Variables" (Simscape Electrical).

Thermal Port

Thermal port — Whether to expose thermal port
`Omit` (default) | `Model`

Whether to expose the thermal port of the block and simulate the thermal effects of the battery.

Thermal mass — Thermal mass associated with the thermal port
`30000 J/K` (default) | positive number

Thermal mass associated with the thermal port H. It represents the energy required to raise the temperature of the thermal port by one degree.

Dependencies

To enable this parameter, set **Thermal port** to `Model`.

Version History

Introduced in R2008b

R2023a: The Modeling option parameter has been replaced

Behavior changed in R2023a

The **Modeling option** parameter of the block has been removed and replaced with the **Expose charge measurement port** and **Thermal port** parameters. To expose the internal charge output port, set the **Expose charge measurement port** parameter to `Yes`. To expose the thermal port and simulate the thermal effects, set the **Thermal port** parameter to `Model`.

As a result of this change, if you have created a custom component by using the Battery block in an earlier release, you must update the custom component accordingly.

```
For batteryecm.battery_thermal:
charge_measurement_port=simscape.enum.thermaleffects.model
```

```
For batteryecm.battery_instrumented:
charge_measurement_port=simscape.enum.battery.enable.yes

For batteryecm.battery_thermal_instrumented:
charge_measurement_port=simscape.enum.thermaleffects.model
charge_measurement_port=simscape.enum.battery.enable.yes
```

References

[1] Ramadass, P., B. Haran, R. E. White, and B. N. Popov. "Mathematical modeling of the capacity fade of Li-ion cells." *Journal of Power Sources*. 123 (2003), pp. 230–240.

[2] Ning, G., B. Haran, and B. N. Popov. "Capacity fade study of lithium-ion batteries cycled at high discharge rates." *Journal of Power Sources*. 117 (2003), pp. 160–169.

Extended Capabilities

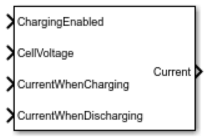
C/C++ Code Generation
Generate C and C++ code using Simulink® Coder™.

See Also

Battery (Table-Based) | DC Voltage Source | Controlled Voltage Source

Battery CC-CV

Constant current constant voltage charging algorithm



Libraries:
Simscape / Battery / BMS / Current Management

Description

This block implements a constant-current (CC), constant-voltage (CV) charging algorithm for a battery. For a discharging battery, the block uses the value of the **CurrentWhenDischarging** input port.

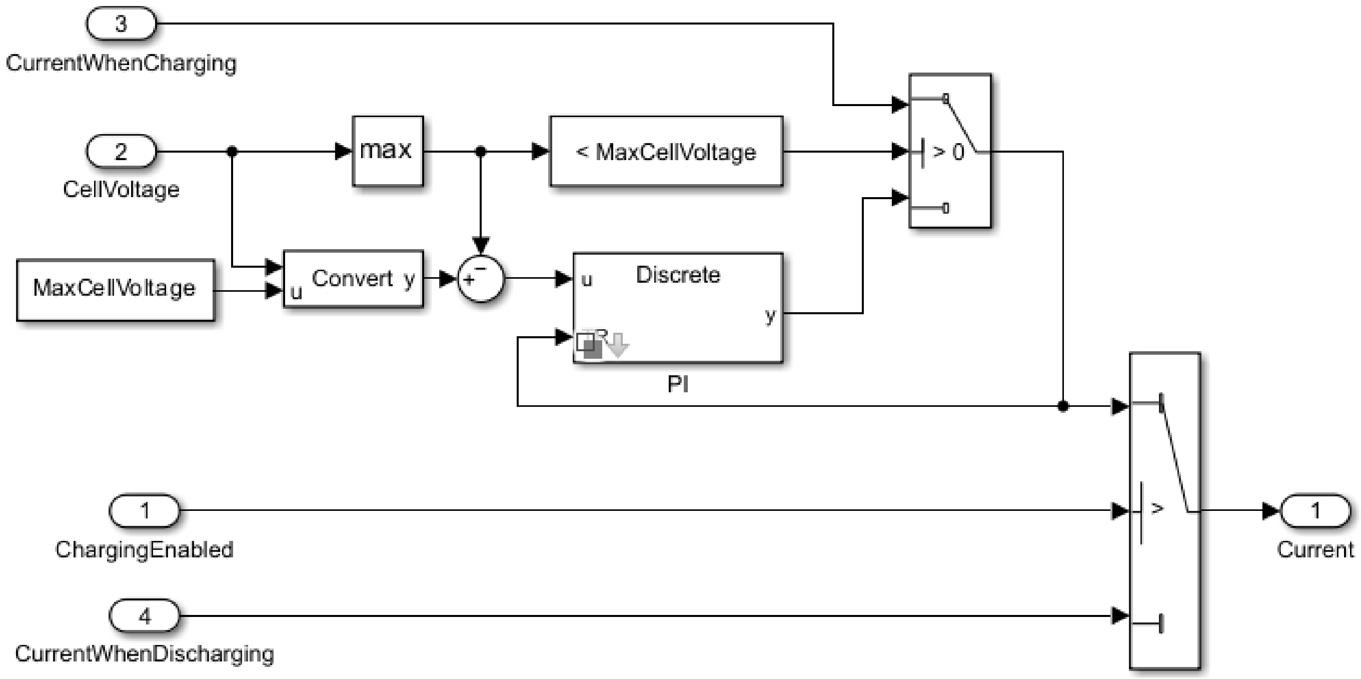
This block supports both single-precision and double-precision floating-point simulation.

Note To enable single-precision floating-point simulation, the data type of all inputs and parameters, except for the **Sample time (-1 for inherited)** parameter, must be `single`.

You can switch between continuous and discrete implementations of the block by using the **Sample time (-1 for inherited)** parameter. To configure the block for continuous time, set the **Sample time (-1 for inherited)** parameter to `0`. To configure the block for discrete time, set the **Sample time (-1 for inherited)** parameter to a positive, nonzero value, or to `-1` to inherit the sample time from an upstream block.

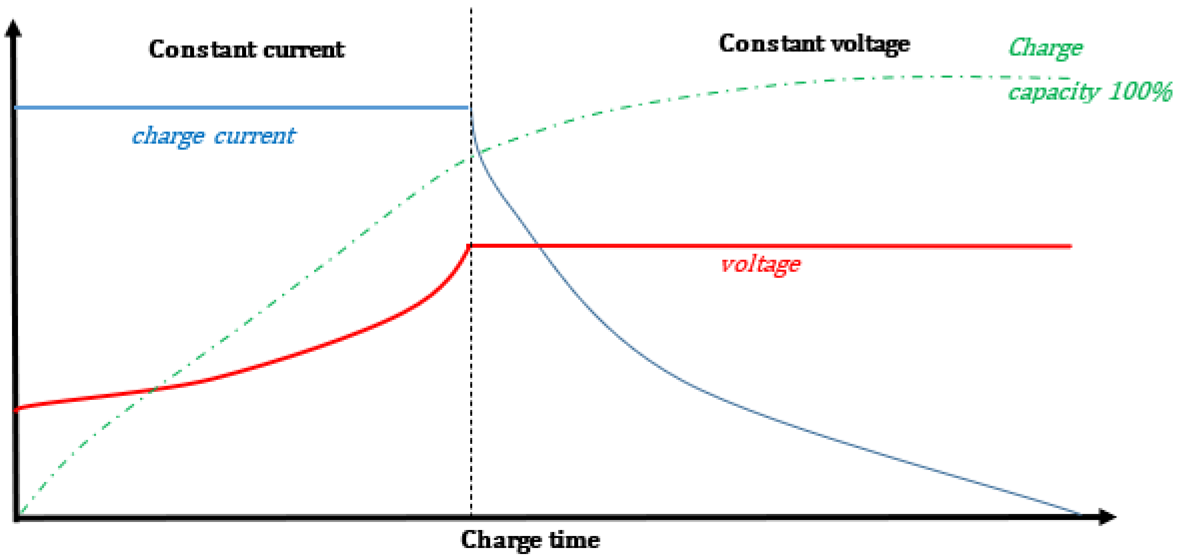
Note Continuous-time implementation of this block works only in double-precision floating-point simulation. If you provide single-precision floating-point parameters and inputs, this block casts them to double-precision floating-point values to prevent errors.

This diagram illustrates the overall structure of the block:



Equations

This block implements the CC-CV algorithm in constant-current and constant-voltage modes. This figure shows the operation of these modes:



This equation defines the battery reference current that the block outputs:

$$Current = \begin{cases} Maximum\ charge\ current, & if\ battery\ is\ charging\ and\ v_{meas} < v_{\max} \\ \left(K_p + K_i\frac{1}{s}\right)(v_{\max} - v_{meas}), & if\ battery\ is\ charging\ and\ v_{meas} \geq v_{\max} \\ Maximum\ discharge\ current, & if\ battery\ is\ discharging \end{cases}$$

where

- v_{max} is the value of the **Maximum cell voltage (V)** parameter.
- v_{meas} is the voltage of the highest cell.
- K_p and K_i are the values of the **Controller proportional gain** and **Controller integral gain** parameters.

Ports

Input

ChargingEnabled — Whether to enable battery charging
boolean

Whether to enable battery charging, specified as `1` (enabled) or `0` (disabled).

CellVoltage — Cell voltages
scalar | vector

Voltage of the cells, specified as a scalar for a single cell or a vector for multiple cells.

CurrentWhenCharging — Current for charging
scalar

Value of the current that the battery uses for charging, specified as a scalar.

CurrentWhenDischarging — Current for discharging
scalar

Value of the current that the battery uses for discharging, specified as a scalar.

Output

Current — Computed reference current
scalar

Reference current for the battery pack, returned as a scalar.

Parameters

Maximum cell voltage (V) — Maximum allowable cell voltage
`4.5` (default) | scalar

Maximum allowable cell voltage, in volt.

Controller proportional gain — Proportional gain
`100` (default) | scalar

Proportional gain, K_p, of the PI controller.

Controller integral gain — Integral gain
`10` (default) | scalar

Integral gain, K_i, of the PI controller.

Controller anti-windup gain — Anti-windup gain
`1` (default) | scalar

Anti-windup gain of the PI controller.

Gain of signal-tracking feedback loop — Tracking coefficient of PI controller
`1` (default) | scalar

Tracking coefficient, K_t, of the PI controller.

The block feeds the difference between the tracking signal and the controller output back to the integrator input with a gain value K_t. Use this parameter to specify the gain in that feedback loop.

Signal tracking has several applications, including bumpless control transfer and avoiding windup in multiloop control structures.

Sample time (-1 for inherited) — Block sample time
`-1` (default) | 0 | positive integer

Time between consecutive block executions. During execution, the block produces outputs and, if appropriate, updates its internal state. For more information, see “What Is Sample Time?” and “Specify Sample Time”.

For inherited discrete-time operation, specify this parameter as `-1`. For discrete-time operation, specify this parameter as a positive integer. For continuous-time operation, specify this parameter as `0`.

If this block is in a masked subsystem or a variant subsystem that allows you to switch between continuous operation and discrete operation, promote the sample time parameter. Promoting the sample time parameter ensures correct switching between the continuous and discrete implementations of the block. For more information, see “Promote Block Parameters on a Mask”.

Version History

Introduced in R2022b

Extended Capabilities

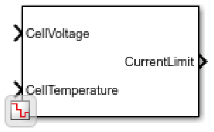
C/C++ Code Generation
Generate C and C++ code using Simulink® Coder™.

See Also

Battery Charging Current Limit | Battery Discharging Current Limit

Battery Charging Current Limit

Maximum battery charging current



Libraries:
Simscape / Battery / BMS / Current Management

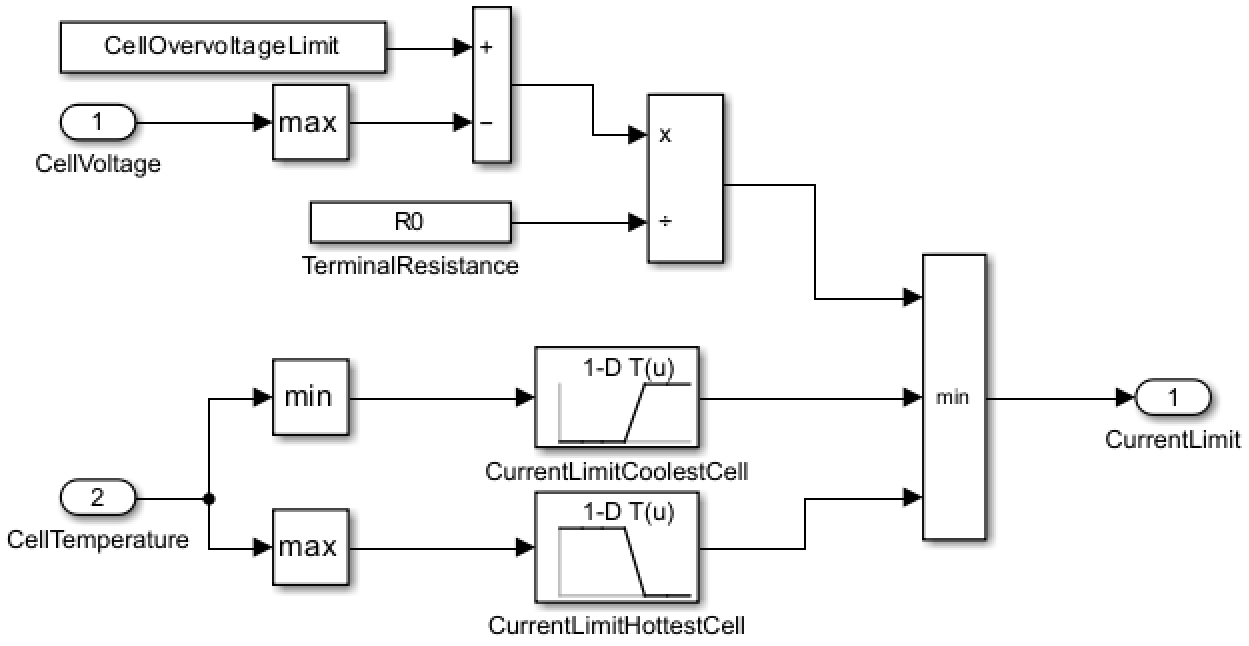
Description

This block calculates the maximum charging current of a battery. Limiting the charging and discharging currents is an important consideration when you model battery packs.

This block supports both single-precision and double-precision floating-point simulation.

Note To enable single-precision floating-point simulation, the data type of all inputs and parameters must be `single`.

This diagram shows the structure of the block:



Equations

The charging current limit is equal to the minimum of these values:

- The temperature-dependent limit for the coolest and hottest cells
- The theoretical current based on the overvoltage limit of the cell
- The terminal resistance

$$CurrentLimit = \min(i_1, i_2, i_3)$$
$$i_1 = \frac{OvervoltageLimit - \max(CellVoltage)}{R_0}$$
$$i_2 = f(min(CellTemperature))$$
$$i_3 = g(max(CellTemperature))$$

where

- *OvervoltageLimit* is the value of the **Cell overvoltage limit (V)** parameter.
- *CellVoltage* is the value of the **CellVoltage** input port.
- R_0 is the value of the **Terminal resistance (ohm)** parameter.
- *CellTemperature* is the value of the **CellTemperature** input port.

Ports

Input

CellVoltage — Cell voltages
scalar | vector

Cell voltage, in volt, specified as a scalar for a single cell or a vector for multiple cells.

CellTemperature — Cell temperatures
scalar | vector

Temperature of the cells, specified as a scalar for a single cell or a vector for multiple cells. The size of this input port must be equal to the size of the **CellVoltage** parameter.

Output

CurrentLimit — Charging current limit
scalar

Charging current limit for the battery pack, returned as a scalar.

Parameters

Cell overvoltage limit (V) — Overvoltage limit for cell
`4.5` (default) | positive scalar

Limit over which the battery cell is overcharged, in volt.

Terminal resistance (ohm) — Terminal resistance
`0.0025` (default) | positive scalar

Terminal resistance, R_0, in ohm.

Vector of temperatures, T — Cell temperatures
`[243.15, 258.15, 273.15, 288.15, 303.15, 318.15, 333.15]` (default) | vector of scalars

Cell temperatures. The physical unit of this parameter must be the same as the physical unit of the **CellTemperature** input port.

Current limit for coolest cell, iCool(T), (A) — Current limit for coolest cell
`[ 0, 0, 0, 0, 100, 100, 100]` (default) | vector of positive scalars

Current limit for the coolest cell, in ampere. The size of this vector must be equal to the size of the **Vector of temperature, T** parameter.

Current limit for hottest cell, iHot(T), (A) — Current limit for hottest cell
`[100, 100, 100, 100, 0, 0, 0]` (default) | vector of positive scalars

Current limit for the hottest cell, in ampere. The size of this vector must be equal to the size of the **Vector of temperature, T** parameter.

Version History

Introduced in R2022b

Extended Capabilities

C/C++ Code Generation

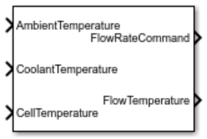
Generate C and C++ code using Simulink® Coder™.

See Also

Battery CC-CV | Battery Discharging Current Limit

Battery Coolant Control

Battery coolant control algorithm



Libraries:
Simscape / Battery / BMS / Thermal Management

Description

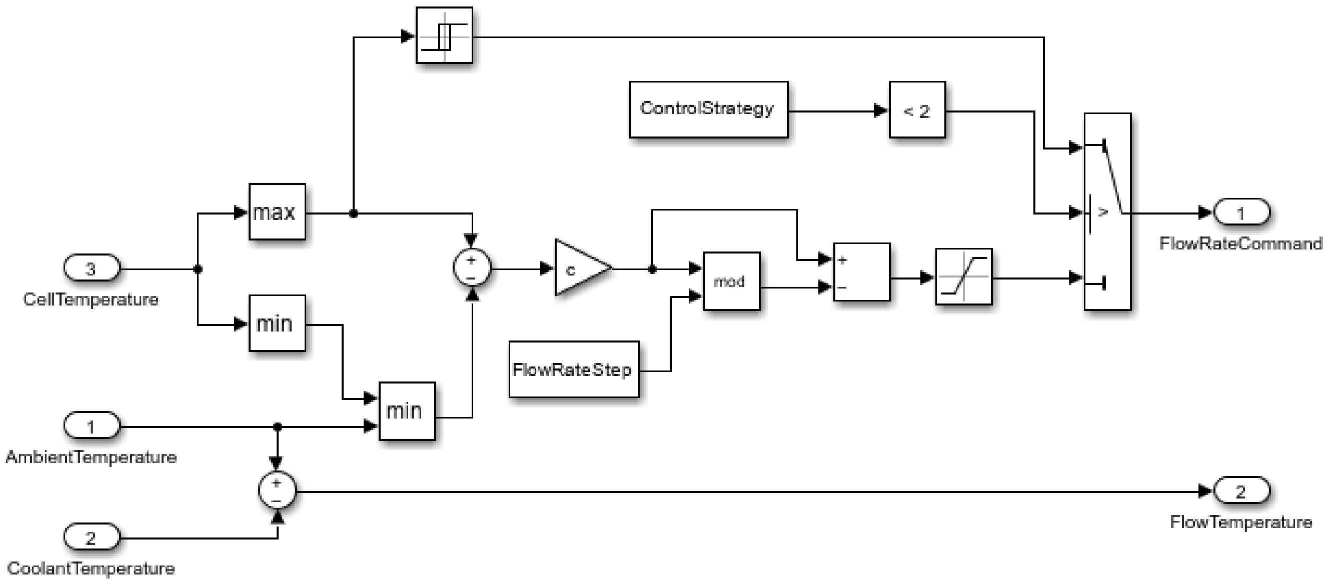
This block implements a battery coolant control algorithm.

Temperature is crucial in batteries because high temperatures significantly decrease the battery life. It is import to keep the temperature of each cell of the battery under control and activate heaters or coolers to keep the temperature within safe limits. If the temperature is too high, the battery management system (BMS) enables a cooling fan.

This block supports both single-precision and double-precision floating-point simulation.

Note To enable single-precision floating-point simulation, the data type of all inputs and parameters must be `single`.

This diagram shows the structure of the block:



Equations

The Battery Coolant Control block uses this equation to compute the flow temperature:

$$FlowTemperature = T_{amb} - T_{cool}$$

where T_{amb} is the ambient temperature and T_{cool} is the coolant temperature.

If the value at the **FlowRateCommand** output port is equal to `0`, there is no flow in the battery. If this value is equal to `1`, then the flow rate in the battery takes its highest possible value.

If you set the **Control strategy** parameter to `ON-OFF control`, the block uses this equation to calculate the flow rate command:

$$FlowRateCommand = \begin{cases} 1, & T_{hottest} \geq T_{on} \\ FlowRateCommand_{old}, & T_{off} < T_{hottest} < T_{on} \\ 0, & T_{hottest} < T_{off} \end{cases}$$

where

- $T_{hottest}$ is the temperature of the hottest cell.
- T_{on} is the switch-on temperature.
- T_{off} is the switch-off temperature, which must be less than T_{on}.
- $FlowRateCommand_{old}$ is the flow rate command at the previous time step.

If you set the **Control strategy** parameter to `Step control`, the block uses these equations to calculate the flow rate command:

$$f(\Delta T) = c(T_{hottest} - \min(T_{coolest}, T_{amb} - T_{cool}))$$
$$FlowRateCommand = \min(f(\Delta T) - \mathrm{mod}(f(\Delta T), step), 1)$$

where

- $T_{coolest}$ is the temperature of the coolest cell.
- *step* is the flow rate step command. For example, if *step = 0.2*, the value at the **FlowRateCommand** output port is one of these values: [0,0.2,0.4,0.6,0.8,1].
- *c* controls how aggressive the coolant strategy is. This value is a constant such that, if $f(\Delta T) \geq 1$, the value at the **FlowRateCommand** output port is equal to 1. Otherwise, the block scales the flow rate command linearly to `0`.

The step control strategy allows intermediate flow commands with the value you specify for the **Flow rate step** parameter. To compute the actual command, the block uses a function of the temperature gradient.

Ports

Input

CellTemperature — Cell temperature
scalar | vector

Temperature of the battery cell, specified as a scalar for a single cell or a vector for multiple cells.

AmbientTemperature — Ambient temperature
scalar

Temperature of the ambient, specified as a scalar.

CoolantTemperature — Coolant temperature
scalar

Temperature of the coolant, specified as a scalar.

Output

FlowRateCommand — Flow rate command
scalar

Flow rate command, returned as a scalar in the range [0, 1]. This output denotes the flow rate relative to the maximum and minimum flow rate values. A value of 0 corresponds to the minimum flow rate. A value of 1 corresponds to the maximum flow rate.

FlowTemperature — Flow temperature
scalar

Temperature of the flow, returned as a positive or negative scalar. The total coolant inlet temperature is equal to the sum of this value and the value at the **AmbientTemperature** input port.

Parameters

Control strategy — Control strategy
`ON-OFF control` (default) | `Step control`

Control strategy for the calculation of the flow rate command.

Switch-on temperature — Coolant switch-on temperature
`305.15` (default) | positive scalar

Temperature at which the coolant pump switches on. The value of this parameter must be greater than or equal to the value of the **Switch-off temperature** parameter.

Dependencies

To enable this parameter, set **Control strategy** to `ON-OFF control`.

Switch-off temperature — Coolant switch-off temperature
`301.15` (default) | positive scalar

Temperature at which the coolant pump switches off.

Dependencies

To enable this parameter, set **Control strategy** to `ON-OFF control`.

Constant used in f(deltaT)=c*deltaT — Constant for coolant strategy
`0.1` (default) | positive scalar

Constant that controls how aggressive the coolant strategy is.

Dependencies

To enable this parameter, set **Control strategy** to `Step control`.

Flow rate step — Flow rate step
`0.2` (default) | positive scalar

Step value used in the calculation of the flow rate command. The value of this parameter must be less than or equal to `1`.

Dependencies

To enable this parameter, set **Control strategy** to `Step control`.

Version History

Introduced in R2022b

Extended Capabilities

C/C++ Code Generation

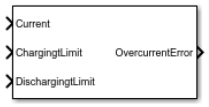
Generate C and C++ code using Simulink® Coder™.

See Also

Battery Heater Control

Battery Current Monitoring

Monitor for battery current



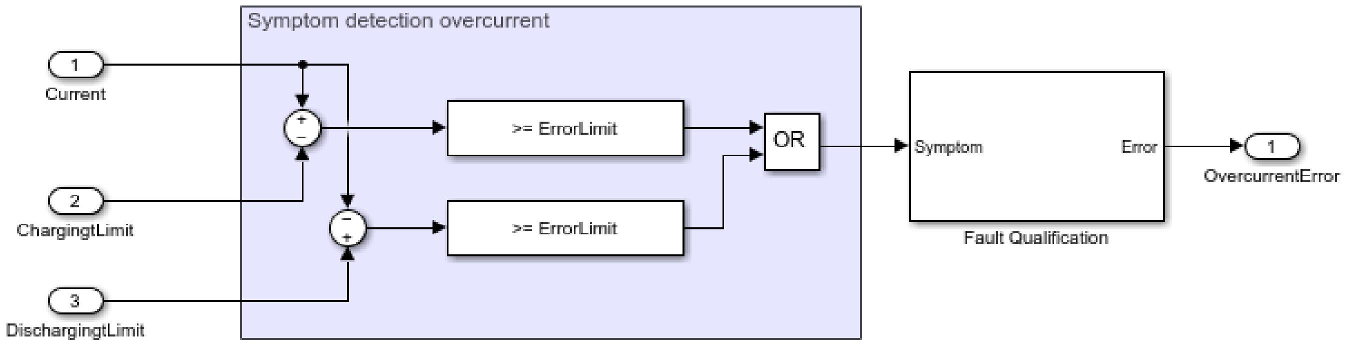
Libraries:
Simscape / Battery / BMS / Protection

Description

This block implements battery overcurrent monitoring. Overcurrent protection is necessary in electrical circuits. A battery that is exposed to overcurrent experiences a considerable increase in cell temperature and is in a dangerously unstable state.

You can switch between continuous and discrete implementations of the block by using the **Sample time (-1 for inherited)** parameter. To configure the block for continuous time, set the **Sample time (-1 for inherited)** parameter to `0`. To configure the block for discrete time, set the **Sample time (-1 for inherited)** parameter to a positive, nonzero value, or to `-1` to inherit the sample time from an upstream block.

This diagram shows the structure of the block:



Equations

This block computes the overcurrent symptom with this equation:

$$Overcurrent\,Symptom = (i - \ Chargelimit) > ErrorLimit \parallel (\,DischargeLimit - i\,) > ErrorLimit\,.$$

The Battery Current Monitoring block then passes the overcurrent symptom to a Fault Qualification block, which qualifies the error.

Ports

Input

PackCurrent — Battery pack current
scalar

Battery pack current, in ampere, specified as a scalar.

ChargingLimit — Charging current limit
scalar

Charging current limit of the battery, in ampere, specified as a scalar.

DischargingLimit — Discharging current limit
scalar

Discharging current limit of the battery, in ampere, specified as a scalar.

Output

OvercurrentError — Indication of overcurrent error
0 | 1

Indication of an overcurrent error. If this output is equal to 1, the battery is in an overcurrent state.

Parameters

Current error limit (A) — Current error limit
0 (default) | nonnegative scalar

Threshold to allow small deviation from the charging and discharging limits, in ampere.

Qualification time (s) — Error qualification time
2 (default) | positive scalar

Time required to qualify the error, in seconds.

Disqualification time (s) — Error disqualification time
0 (default) | nonnegative scalar

Time required to disqualify the error, in seconds. If you set this parameter to 0, the block does not disqualify the error.

Sample time (-1 for inherited) — Block sample time
-1 (default) | 0 | positive integer

Time between consecutive block executions. During execution, the block produces outputs and, if appropriate, updates its internal state. For more information, see “What Is Sample Time?” and “Specify Sample Time”.

For inherited discrete-time operation, specify this parameter as -1. For discrete-time operation, specify this parameter as a positive integer. For continuous-time operation, specify this parameter as 0.

If this block is in a masked subsystem or a variant subsystem that allows you to switch between continuous operation and discrete operation, promote the sample time parameter. Promoting the sample time parameter ensures correct switching between the continuous and discrete implementations of the block. For more information, see “Promote Block Parameters on a Mask”.

Version History

Introduced in R2022b

Extended Capabilities

C/C++ Code Generation

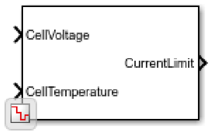
Generate C and C++ code using Simulink® Coder™.

See Also

Battery Temperature Monitoring | Battery Voltage Monitoring | Fault Qualification

Battery Discharging Current Limit

Maximum battery discharging current



Libraries:
Simscape / Battery / BMS / Current Management

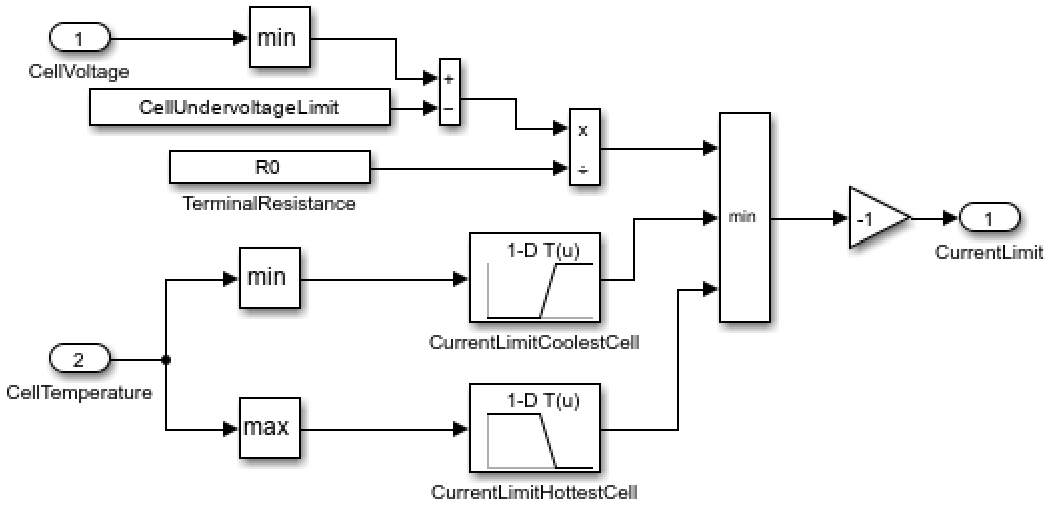
Description

This block calculates the maximum discharging current of a battery. Limiting the charging and discharging currents is an important consideration when you model battery packs.

This block supports both single-precision and double-precision floating-point simulation.

Note To enable single-precision floating-point simulation, the data type of all inputs and parameters must be `single`.

This diagram shows the structure of the block:



Equations

The discharging current limit is equal to the minimum of these values:

- The temperature-dependent limit for the coolest and hottest cells
- The theoretical current based on the undervoltage limit of the cell
- The terminal resistance

$$CurrentLimit = -\min(i_1, i_2, i_3)$$
$$i_1 = \frac{\min(CellVoltage) - UndervoltageLimit}{R_0}$$
$$i_2 = f(min(CellTemperature))$$
$$i_3 = g(max(CellTemperature))$$

where

- *UndervoltageLimit* is the value of the **Cell undervoltage limit (V)** parameter.
- *CellVoltage* is the value of the **CellVoltage** input port.
- R_0 is the value of the **Terminal resistance (ohm)** parameter.
- *CellTemperature* is the value of the **CellTemperature** input port.

Ports

Input

CellVoltage — Cell voltages
scalar | vector

Cell voltage, in volt, specified as a scalar for a single cell or a vector for multiple cells.

CellTemperature — Cell temperatures
scalar | vector

Temperature of the cells, specified as a scalar for a single cell or a vector for multiple cells. The size of this input port must be equal to the size of the **CellVoltage** parameter.

Output

CurrentLimit — Discharging current limit
scalar

Discharging current limit for the battery pack, returned as a scalar.

Parameters

Cell undervoltage limit (V) — Undervoltage limit for cell
`2.5` (default) | positive scalar

Limit under which the battery cell is undercharged, in volt.

Terminal resistance (ohm) — Terminal resistance
`0.0025` (default) | positive scalar

Terminal resistance, R_0, in ohm.

Vector of temperatures, T — Cell temperatures
`[243.15, 258.15, 273.15, 288.15, 303.15, 318.15, 333.15]` (default) | vector of scalars

Cell temperatures. The physical unit of this parameter must be the same as the physical unit of the **CellTemperature** input port.

Current limit for coolest cell, iCool(T), (A) — Current limit for coolest cell
`[ 0, 0, 0, 0, 100, 100, 100]` (default) | vector of positive scalars

Current limit for the coolest cell, in ampere. The size of this vector must be equal to the size of the **Vector of temperature, T** parameter.

Current limit for hottest cell, iHot(T), (A) — Current limit for hottest cell
`[100, 100, 100, 100, 0, 0, 0]` (default) | vector of positive scalars

Current limit for the hottest cell, in ampere. The size of this vector must be equal to the size of the **Vector of temperature, T** parameter.

Version History

Introduced in R2022b

Extended Capabilities

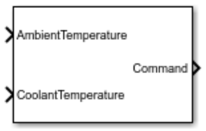
C/C++ Code Generation
Generate C and C++ code using Simulink® Coder™.

See Also

Battery CC-CV | Battery Charging Current Limit

Battery Heater Control

Battery heater control algorithm



Libraries:
Simscape / Battery / BMS / Thermal Management

Description

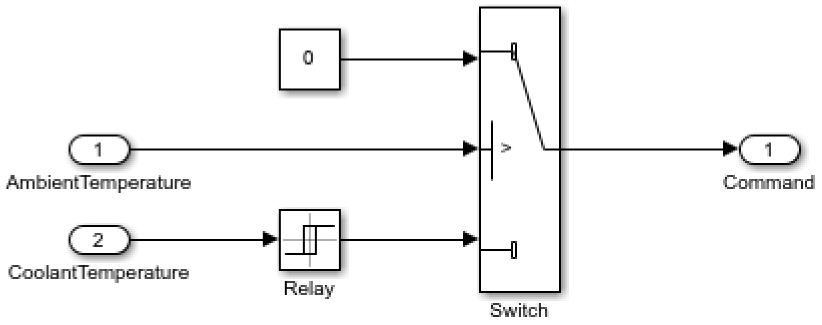
This block implements a battery heater control algorithm.

Temperature is crucial in batteries as low temperatures significantly decrease the battery life. It is then fundamental to keep the temperature of each cell of the battery under control and activate heaters or coolers to keep the temperature under safe limits. If the temperature is too low, the battery management system (BMS) enables a heater.

This block supports both single-precision and double-precision floating-point simulation.

Note To enable single-precision floating-point simulation, the data type of all inputs and parameters must be `single`.

This diagram shows the structure of the block:



Equations

The Battery Heater Control block activates the heater only if the temperature of the ambient drops below the value of the **Ambient temperature threshold for heater** parameter. If the temperature drops below this threshold, the value at the **Command** output port is equal to:

$$Command = \begin{cases} 0, & T_{coolant} \geq T_{off} \\ Command_{old}, & T_{on} < T_{coolant} < T_{off} \\ 1, & T_{coolant} \leq T_{on} \end{cases}$$

where

- $T_{coolant}$ is the coolant temperature.
- T_{on} is the coolant temperature at which the heater switches on.
- T_{off} is the coolant temperature at which the heater switches off. This value is greater than T_{on}.
- $Command_{old}$ is the command at the previous time step.

Ports

Input

AmbientTemperature — Ambient temperature
scalar

Temperature of the ambient, specified as a scalar.

CoolantTemperature — Coolant temperature
scalar

Temperature of the coolant, specified as a scalar.

Output

Command — Heater command
scalar

Heater command, returned as a scalar. If this value is equal to `0`, the heater switches off. If this value is equal to `1`, the heater switches on.

Parameters

Ambient temperature threshold for heater — Ambient temperature threshold for heater
`278.15` (default) | positive scalar

Ambient temperature at which the block activates the heater.

Switch-on coolant temperature — Switch-on coolant temperature
`278.15` (default) | positive scalar

Coolant temperature at which the heater switches on.

Switch-off coolant temperature — Switch-off coolant temperature
`290.15` (default) | positive scalar

Coolant temperature at which the heater turns off. The value of this parameter must be greater than the value of the **Switch-on coolant temperature** parameter.

Version History

Introduced in R2022b

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using Simulink® Coder™.

See Also

Battery Coolant Control

Battery (Table-Based)

Tabulated battery model

Libraries:
Simscape / Electrical / Sources
Simscape / Battery / Cells

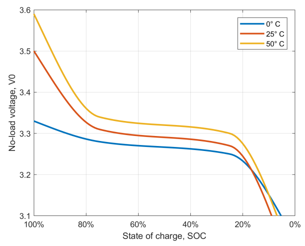
Description

The Battery (Table-Based) block represents a high-fidelity battery model. The block calculates open-circuit voltage as a function of charge level and optional temperature using lookup tables and includes several modeling options:

- Self-discharge
- Battery fade
- Charge dynamics
- Calendar aging

Note The block can use linear or nearest interpolation and extrapolation for all the table based parameters. For rows and columns, it follows the row-column convention, whereas rows are indexed first and, subsequently, columns.

The plot shows a battery whose performance varies with temperature and state of charge changes, as typically found on a datasheet.



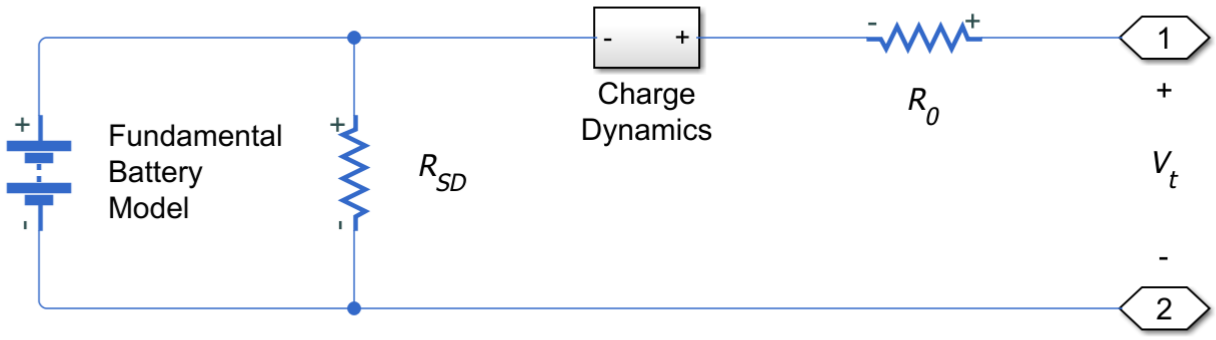
Use this block to parameterize batteries with complex open-circuit voltage behavior from datasheets or experimental results. For a simpler representation of a battery, see the Battery block.

The Battery (Table-Based) block has two optional ports that you can expose by setting the corresponding parameters. The extra physical signal port, **SOC**, outputs the internal state of charge. Use this functionality to change the load behavior as a function of state of charge, without the complexity of building a charge state estimator. To expose the **SOC** port, in the **Main** setting, set the **Expose SOC measurement port** parameter to `Yes`.

To expose the thermal port, in the **Thermal** setting, set the **Thermal port** parameter to `Model`. The thermal port represents the battery thermal mass.

You can also choose different built-in parameterizations for this block. For more information, see the "Predefined Parameterization" on page 1-55 section.

The battery equivalent circuit is made up of the fundamental battery model, the self-discharge resistance R_{SD}, the charge dynamics model, and the series resistance R_0.



Battery Model

The block calculates the open-circuit voltage, or the voltage across the fundamental battery model by interpolation:

$$v_0 = v_0(\mathrm{SOC}, T)$$

Where:

- v_0 is the open-circuit voltage of the battery. Specify the grid of lookup values using the **Open-circuit voltage, V0(SOC,T)** parameter if tabulating parameters over temperature, or **Open-circuit voltage, V0(SOC)** otherwise.
- SOC is the ratio of current charge to nominal battery capacity specified in the **Cell capacity, AH** parameter along with the effects of the temperature dependent fade percentage change in cell capacity, $\delta_{AH}(n, Tfade)$, specified in the **Percentage change in cell capacity, dAH(N, Tfade)** parameter. Specify the SOC breakpoints using the **Vector of state-of-charge values, SOC** parameter. The block estimates the nominal battery capacity based on the number of cycles and the temperature of the battery by interpolating the specified temperature dependent fade characteristic and the **Cell capacity, AH** parameter.

  SOC represents the normalized data with respect to q_{nom}.

  For the lookup-table based fade characteristic option,

  $$q_{nom}(T, n) = \left(1 + \frac{\delta_{AH}(n, T_{fade})}{100}\right) * AH\ Ah\,.$$

  For the equation-based fade characteristic option,

  $$q_{nom}(T, n) = \left(1 + \frac{\delta_{AH}}{100}\sqrt{\frac{n}{N}}\right) * AH\ Ah\,.$$

  Finally, SOC is obtained from the following equation.

  $$SOC(t) = \frac{1}{q_{nom}(T, n)} \int \left(i(t) - \frac{V_{open}(T, n, t)}{R_{SD}(T, n)}\right) dt$$

Where:

  - q_{nom} is the cell capacity of the battery. Specify this value using the **Cell capacity, AH** parameter.
  - N is the reference number of discharge cycles over which you specify percent change of several battery parameters. Set this value using the **Number of discharge cycles, N** parameter.
  - n is the present number of cycles of the battery.
  - δ_{AH} is the percentage change in cell capacity of the battery after N discharge cycles.
- T is the battery temperature. Specify the T breakpoints using the **Vector of temperatures, T** parameter if tabulating the parameters over temperature.

The block also models the series resistance R_0 as a function of state of charge and optional temperature. Specify the grid of lookup values for the series resistance using the **Terminal resistance, R0(SOC,T)** parameter if tabulating the parameters over temperature, or **Terminal resistance, R0(SOC)** otherwise.

Modeling Self-Discharge

When the battery terminals are open-circuit, it is still possible for internal currents to discharge the battery. This behavior is called self-discharge. To enable this effect, set the **Self-discharge** parameter to `Enabled`.

The block models these internal currents with a temperature-dependent resistance $R_{SD}(T)$ across the terminals of the fundamental battery model. You can specify the lookup values for this resistance using the **Self-discharge resistance, Rleak(T)** parameter if tabulating the parameters over temperature, or **Self-discharge resistance, Rleak** otherwise.

Modeling Charge Dynamics

Batteries are not able to respond instantaneously to load changes. They require some time to achieve a steady-state. This time-varying property is a result of battery charge dynamics and is modeled using parallel RC sections in the equivalent circuit.

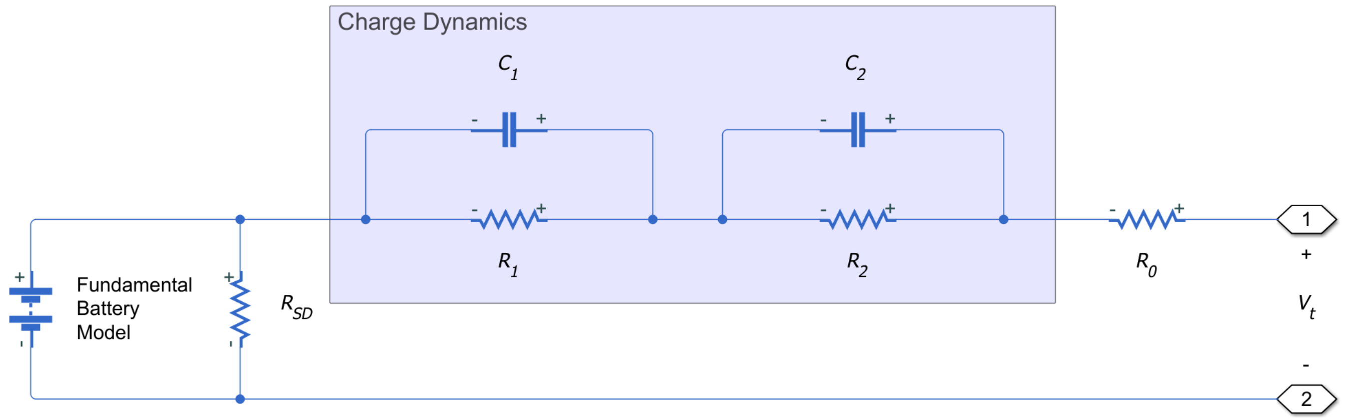
You can model battery charge dynamics using the **Charge dynamics** parameter:

- `No dynamics` — The equivalent circuit contains no parallel RC sections. There is no delay between terminal voltage and internal charging voltage of the battery.
- `One time-constant dynamics` — The equivalent circuit contains one parallel RC section. Specify the time constant using the **First time constant, tau1(SOC,T)** parameter if tabulating parameters over temperature or **First time constant, tau1(SOC)** otherwise.
- `Two time-constant dynamics` — The equivalent circuit contains two parallel RC sections. Specify the time constants using the **First time constant, tau1(SOC,T)** and **Second time constant, tau2(SOC,T)** parameters if tabulating parameters over temperature or **First time constant, tau1(SOC)** and **Second time constant, tau2(SOC)** otherwise.
- `Three time-constant dynamics` — The equivalent circuit contains three parallel RC sections. Specify the time constants using the **First time constant, tau1(SOC,T)**, **Second time constant, tau2(SOC,T)**, and **Third time constant, tau3(SOC,T)** parameters if tabulating parameters over temperature or **First time constant, tau1(SOC)**, **Second time constant, tau2(SOC)**, and **Third time constant, tau3(SOC)** otherwise.
- `Four time-constant dynamics` — The equivalent circuit contains four parallel RC sections. Specify the time constants using the **First time constant, tau1(SOC,T)**, **Second time constant,**

tau2(SOC,T), **Third time constant, tau3(SOC,T)**, and **Fourth time constant, tau4(SOC,T)** parameters if tabulating parameters over temperature or **First time constant, tau1(SOC)**, **Second time constant, tau2(SOC)**, **Third time constant, tau3(SOC)**, and **Fourth time constant, tau4(SOC)** otherwise.

- `Five time-constant dynamics` — The equivalent circuit contains five parallel RC sections. Specify the time constants using the **First time constant, tau1(SOC,T)**, **Second time constant, tau2(SOC,T)**, **Third time constant, tau3(SOC,T)**, **Fourth time constant, tau4(SOC,T)**, and **Fifth time constant, tau5(SOC,T)** parameters if tabulating parameters over temperature or **First time constant, tau1(SOC)**, **Second time constant, tau2(SOC)**, **Third time constant, tau3(SOC)**, **Fourth time constant, tau4(SOC)**, and **Fifth time constant, tau5(SOC)** otherwise.

This diagram shows the equivalent circuit for the block configured with two time-constant dynamics.



In the diagram:

- R_1 and R_2 are the parallel RC resistances. Specify these values with the **First polarization resistance, R1(SOC,T)** and **Second polarization resistance, R2(SOC,T)** parameters, respectively, if tabulating parameters over temperature or **First polarization resistance, R1(SOC)** and **Second polarization resistance, R2(SOC)** otherwise.
- C_1 and C_2 are the parallel RC capacitances. The time constant τ for each parallel section relates the R and C values using the relationship $C = \tau/R$. Specify τ for each section using the **First time constant, tau1(SOC,T)** and **Second time constant, tau2(SOC,T)** parameters, respectively, if tabulating parameters over temperature or **First time constant, tau1(SOC)** and **Second time constant, tau2(SOC)** otherwise.
- R_0 is the series resistance. Specify this value with the **Terminal resistance, R0(SOC,T)** parameter if tabulating parameters over temperature or **Terminal resistance, R0(SOC)** otherwise.

Modeling Battery Fade

Battery fade is the deterioration of battery performance over repeated charge and discharge cycles. When the **Fade characteristic** parameter is set to `Equations`, the battery fade is modeled as follows.

The open-circuit voltage across the fundamental battery model fades proportionally with the number of discharge cycles n:

$$v_{0,\,\mathrm{fade}} = v_0\left(1 + \frac{\delta_{v0}}{100}\frac{n}{N}\right)$$

Where δ_{v_0} is the percent change in open-circuit voltage after N discharge cycles. Specify δ_{v_0} using the **Change in open-circuit voltage after N discharge cycles (%)** parameter.

The nominal charge, from which state of charge is calculated, fades with the square root of number of discharge cycles:

$$q_{nom,\,\mathrm{fade}} = q_{nom}\left(1 + \frac{\delta_{AH}}{100}\sqrt{\frac{n}{N}}\right)$$

All resistances in the battery model also fade with the square root of the number of discharge cycles:

$$R_{i,\,\mathrm{fade}} = R_i\left(1 + \frac{\delta_{R_i}}{100}\sqrt{\frac{n}{N}}\right)$$

Where:

- R_i is the i^{th} resistance
- δ_{R_i} is the percent change in this resistance over N cycles

Depending on how you have configured the block, the resistances might include:

- The series resistance — Specify the percent change over N cycles using the **Change in terminal resistance after N discharge cycles (%)** parameter.
- The self-discharge resistance — Specify the percent change over N cycles using the **Change in self-discharge resistance after N discharge cycles (%)** parameter.
- The first charge dynamics resistance — Specify the percent change over N cycles using the **Change in first polarization resistance after N discharge cycles (%):** parameter.
- The second charge dynamics resistance — Specify the percent change over N cycles using the **Change in second polarization resistance after N discharge cycles (%)** parameter.
- The third charge dynamics resistance — Specify the percent change over N cycles using the **Change in third polarization resistance after N discharge cycles (%)** parameter.
- The fourth charge dynamics resistance — Specify the percent change over N cycles using the **Change in fourth polarization resistance after N discharge cycles (%)** parameter.
- The fifth charge dynamics resistance — Specify the percent change over N cycles using the **Change in fifth polarization resistance after N discharge cycles (%)** parameter.

Note You can also model the battery fade characteristic by using lookup tables (temperature independent) or lookup tables (temperature dependent). Choosing any of these two options changes the blocks parameters accordingly. For more information, see the “Fade” on page 1-65 parameters tab.

Modeling Thermal Effects

The battery temperature is determined from a summation of all the ohmic losses included in the model:

$$M_{th}\dot{T} = \sum_i V_{T,i}{}^2/R_{T,i}$$

Where:

- M_{th} is the battery thermal mass.
- i corresponds to the i^{th} ohmic loss contributor. Depending on how you have configured the block, the losses might include:
  - The series resistance
  - The self-discharge resistance
  - The first charge dynamics segment
  - The second charge dynamics segment
  - The third charge dynamics segment
  - The fourth charge dynamics segment
  - The fifth charge dynamics segment
- $V_{T,i}$ is the voltage drop across resistor i.
- $R_{T,i}$ is resistor i.

Modeling Battery Aging

You can model the battery performance deterioration that occurs when the battery is not used. Calendar aging affects both the internal resistance and capacity. In particular, the resistance increase depends by various mechanisms such as the creation of Solid Electrolyte Interface (SEI) at both anode and cathode and the corrosion of the current collector. These processes mainly depends on the storage temperature, the storage state of charge, and time.

You can model the calendar aging in the Battery (Table-Based) block by setting the **Modeling option** parameter to:

- `Equation-based`
- `Tabulated: temperature`
- `Tabulated: time and temperature`

Note The Battery (Table-Based) block only applies the calendar aging during initialization. When you set the **Internal resistance calendar aging** or **Capacity calendar aging** parameters to `Enabled`, the block exposes the **Vector of time intervals** parameter that represents the time the battery has aged before the start of the simulation. Calendar aging during the simulation is not covered with this option.

Equation-based

This equation defines the terminal resistance increase of the battery due to calendar aging:

$$\alpha_r(T, V_{oc}) = (bV_{oc} - c)e^{-\frac{qd}{kT}},$$

$$R = R_0\left(1 + \sum_{i=1}^{i=n} \alpha_r(T_i, V_{oc})\left(t_i^a - t_{i-1}^a\right)\right),$$

where:

- V_{oc} is the **Normalized open-circuit voltage during storage, V/Vnom**.
- R_0 is the **Internal resistance**.
- t_i is the time sample derived from the **Vector of time intervals** parameter.
- T_i is derived from the **Vector of temperatures, T** parameter.
- b is the **Terminal resistance linear scaling for voltage, b**.
- c is the **Terminal resistance constant offset for voltage, c**.
- d is the **Terminal resistance temperature-dependent exponential increase, d**.
- a is the **Terminal resistance time exponent, a**.
- q is the electron's elementary charge, in C.
- k is the Boltzmann constant, in J/K.

This equation defines the capacity decrease of the battery due to calendar aging:

$$\alpha_c(T, V_{oc}) = (bV_{oc} - c)e^{-\frac{qd}{kT}},$$
$$C = C_0\left(1 - \sum_{i=1}^{i=n} \alpha_c(T_i, V_{oc})\left(t_i^a - t_{i-1}^a\right)\right),$$

where:

- C_0 is the **initial capacity**.
- b is the **Capacity linear scaling for voltage, b**.
- c is the **Capacity constant offset for voltage, c**.
- d is the **Capacity temperature-dependent exponential increase, d**.
- a is the **Capacity time constant, a**.

If you set the **Storage condition** parameter to `Specify state-of-charge during storage`, the block converts the state of charge during storage into normalized open-circuit voltage using the tabulated voltage $\mathbf{V_0}$ against the state of charge and the temperature during storage.

Tabulated: temperature

The aged terminal resistance is the product between the terminal resistance, $R_0(SOC,T)$, the percentage resistance increase, dR_0, and the power law that describes the time dependence of the calendar aging:

$$R_0(SOC, T; t_{st}, T_{st}) = R_0(SOC, T)\left(1 + \sum_{i=1}^{n} \frac{dR_0(T_{st,i})}{100}\left(\frac{t^a{}_{st,i} - t^a{}_{st,i-1}}{t^a{}_{st,m}}\right)\right),$$

where:

- T is the battery temperature. Specify the T breakpoints using the **Vector of temperatures, T** parameter if tabulating the parameters over temperature.
- T_{st} is the **Vector of storage temperatures**.
- $t_{st,i}$ and $t_{st,i-1}$ are the time samples derived from the **Vector of time intervals**.
- t_0 is assumed to be null.

- $t_{st,m}$ is the moment in time at which the resistance increase, dR_0, is measured.

The same equation applies to the calculation of the aged battery capacity.

Tabulated: time and temperature

The aged terminal resistance is the product between the terminal resistance, $R_0(SOC,T)$ and dR_0:

$$R_0(SOC, T; \Delta t_{st}, T_{st}) = R_0(SOC, T)\left(1 + \sum_{i=1}^{n} \frac{dR_0(\Delta t_{st,i}, T_{st,i})}{100}\right).$$

The same equation applies to the calculation of the aged battery capacity.

Predefined Parameterization

There are multiple available built-in parameterizations for the Battery (Table-Based) block.

This pre-parameterization data allows you to set up the block to represent components by specific suppliers. The parameterizations of these batteries match the discharge curves in the manufacturer data sheets. To load a predefined parameterization, double-click the Battery (Table-Based) block, click the **<click to select>** hyperlink of the **Selected part** parameter and, in the Block Parameterization Manager window, select the part you want to use from the list of available components.

The available pre-parameterized data model steady state electrical parameters of a lithium-ion battery. The **Open-circuit voltage, V0(SOC)**, **Terminal resistance, R0(SOC, T)**, **Cell capacity, AH**, **Percentage change in cell capacity, dAH**, and **Percentage change in Open-circuit voltage, dV0(N)** parameters are parameterized from characteristics curves in the manufacture datasheets. The change in series resistance with the battery cycle life, the thermal mass, and the dynamic RC network parameters are not parameterized. Instead, the net resistance of the RC network resistors is summed to the series resistance of the specific pre-parameterized data. If you need to fill the RC parameter data, subtract the net RC network resistance from the series resistance data.

The available data is parameterized for 1C discharge current for different temperatures up to the minimum terminal voltage in the datasheet. The data below the minimum terminal voltage is extrapolated. The **Open-circuit voltage, V0(SOC)** parameter is approximated to be temperature independent. At lower temperature, in the low SOC region, the terminal resistances are constant and equal to the value of the resistance at the minimum terminal voltage. This might lead to higher thermal losses.

Note The predefined parameterizations of Simscape components use available data sources for the parameter values. Engineering judgement and simplifying assumptions are used to fill in for missing data. As a result, expect deviations between simulated and actual physical behavior. To ensure accuracy, validate the simulated behavior against experimental data and refine component models as necessary.

For more information about pre-parameterization and for a list of the available components, see “List of Pre-Parameterized Components” (Simscape Electrical).

Plotting Voltage-Charge Characteristics

A quick plot feature lets you visualize the voltage-charge characteristic for the battery model parameter values. To plot the characteristics, right-click a Battery (Table-Based) block in your model and, from the context menu, select **Electrical** > **Basic characteristics**. The software automatically computes a set of bias conditions, based on the block parameter values, and opens a figure window containing a plot of open-circuit voltage versus the state-of-charge (SOC) for the block. For more information, see “Plot Basic Characteristics for Battery Blocks” (Simscape Electrical).

Ports

Output

SOC — Battery charge level, C
physical signal

Physical signal port that outputs the internal state of charge. Use this output port to change load behavior as a function of state of charge, without the complexity of building a charge state estimator. The state of charge is the normalized value estimated from the ratio of current charge with the nominal battery capacity, $q_{nom}(T,n)$. The block estimates the current battery charge by integrating the battery terminal output current. To convert the state of charge into actual charge, you must use the correct nominal battery capacity for each temperature.

Dependencies

To enable this port, in the **Main** setting, set the **Expose SOC measurement port** parameter to `Yes`.

Conserving

+ — Positive terminal
electrical

Electrical conserving port associated with the battery positive terminal.

- — Negative terminal
electrical

Electrical conserving port associated with the battery negative terminal.

H — Battery thermal mass
thermal

Thermal conserving port that represents the battery thermal mass.

Dependencies

To enable this port, in the **Thermal** setting, set the **Thermal port** parameter to `Yes`.

Parameters

Selected part — Predefined parameterization option
`<click to select>` (default)

Whether to parameterize the block to represent components by specific suppliers. Click the **<click to select>** hyperlink to open the Block Parameterization Manager window. For more information on the Block Parameterization Manager, see “Load Predefined Parameterizations” (Simscape Electrical).

Main

Vector of state-of-charge values, SOC — SOC breakpoints
`[0, .1, .25, .5, .75, .9, 1]` (default) | vector

Vector of state-of-charge breakpoints defining the points at which you specify lookup data. This vector must be strictly ascending. The state-of-charge value is calculated with respect to the nominal battery capacity specified in the **Cell capacity, AH** parameter. `SOC` is the ratio of the available battery charge, $q_{battery}$ and the nominal battery capacity, $q_{nom}(T,n)$. You must make sure that, for each temperature, `SOC = 1` represents the respective battery charge capacity specified in the **Cell capacity, AH** parameter, assuming a fresh battery with a number of cycles, N, equal to 1 and $\delta_{AH}(n = 1, T_{fade}) = 0$.

$$SOC = \frac{q_{battery}}{q_{nom}(T,n)}$$
$$for\, N = 1\; and\; \delta_{AH}(n, T_{fade}) = 0,\;\; q_{nom}(T,n) = AH\,.$$

Temperature dependent tables — Select whether to tabulate battery parameters over temperature
`Yes` (default) | `No`

Select whether to tabulate battery parameters over temperature.

Current directionality — Enable direction of current
`Disabled` (default) | `Enabled`

Whether to enable current directionality. If you set this parameter to `Enabled`, the terminal resistance depends on the direction of the current.

Terminal voltage operating range [Min Max] — Terminal voltage operating range
`[0, inf]` V (default) | vector of positive numbers

Operating range of the terminal voltage. This parameter must be a vector of size 1-by-2 that defines the minimum and maximum values of the terminal voltage.

Vector of temperatures, T — *T* breakpoints
`[278, 293, 313]` K (default) | vector of positive numbers

Vector of temperature breakpoints defining the points at which you specify lookup data. This vector must be strictly ascending and greater than `0` K.

Dependencies

This parameter is visible only when the **Temperature dependent tables** parameter is set to `Yes`.

Open-circuit voltage, V0(SOC,T) — *V0* lookup table with temperature breakpoint
`[3.49, 3.5, 3.51; 3.55, 3.57, 3.56; 3.62, 3.63, 3.64; 3.71, 3.71, 3.72; 3.91, 3.93, 3.94; 4.07, 4.08, 4.08; 4.19, 4.19, 4.19]` V (default) | matrix of nonnegative numbers

Lookup data for open-circuit voltages across the fundamental battery model at the specified SOC and temperature breakpoints.

Dependencies

This parameter is visible only when the **Temperature dependent tables** parameter is set to `Yes`.

Open-circuit voltage, V0(SOC) — *V0* lookup table
`[3.5057, 3.566, 3.6337, 3.7127, 3.9259, 4.0777, 4.1928] V` (default) | matrix of nonnegative numbers

Lookup data for open-circuit voltages across the fundamental battery model at the specified SOC.

Dependencies

This parameter is visible only when the **Temperature dependent tables** parameter is set to `No`.

Terminal resistance, R0(SOC,T) — R0 lookup table with temperature breakpoint
`[.0117, .0085, .009; .011, .0085, .009; .0114, .0087, .0092; .0107, .0082, .0088; .0107, .0083, .0091; .0113, .0085, .0089; .0116, .0085, .0089] Ohm` (default) | matrix of nonnegative numbers

Lookup data for series resistance of the battery at the specified SOC and temperature breakpoints.

Dependencies

This parameter is visible only when the **Temperature dependent tables** parameter is set to `Yes` and the **Current directionality** parameter is set to `Disabled`.

Terminal resistance during discharging, R0(SOC,T) — R0 lookup table with temperature breakpoint during discharging phase
`[.0117, .0085, .009; .011, .0085, .009; .0114, .0087, .0092; .0107, .0082, .0088; .0107, .0083, .0091; .0113, .0085, .0089; .0116, .0085, .0089] Ohm` (default) | matrix of nonnegative numbers

Lookup data for series resistance of the battery at the specified SOC and temperature breakpoints during the discharging phase.

Dependencies

This parameter is visible only when the **Temperature dependent tables** parameter is set to `Yes` and the **Current directionality** parameter is set to `Enabled`.

Terminal resistance during charging, R0(SOC,T) — R0 lookup table with temperature breakpoint during charging phase
`[.0117, .0085, .009; .011, .0085, .009; .0114, .0087, .0092; .0107, .0082, .0088; .0107, .0083, .0091; .0113, .0085, .0089; .0116, .0085, .0089] Ohm` (default) | matrix of nonnegative numbers

Lookup data for series resistance of the battery at the specified SOC and temperature breakpoints during the charging phase.

Dependencies

This parameter is visible only when the **Temperature dependent tables** parameter is set to `Yes` and the **Current directionality** parameter is set to `Enabled`.

Terminal resistance, R0(SOC) — R0 lookup table
`[.0085, .0085, .0087, .0082, .0083, .0085, .0085] Ohm` (default) | matrix of nonnegative numbers

Lookup data for series resistance of the battery at the specified SOC.

Dependencies

This parameter is visible only when the **Temperature dependent tables** parameter is set to `No` and the **Current directionality** parameter is set to `Disabled`.

Terminal resistance during discharging, R0(SOC) — R0 lookup table during discharging phase
`[.0085, .0085, .0087, .0082, .0083, .0085, .0085] Ohm` (default) | matrix of nonnegative numbers

Lookup data for series resistance of the battery at the specified SOC during the discharging phase.

Dependencies

This parameter is visible only when the **Temperature dependent tables** parameter is set to `No` and the **Current directionality** parameter is set to `Enabled`.

Terminal resistance during charging, R0(SOC) — R0 lookup table during charging phase
`[.0085, .0085, .0087, .0082, .0083, .0085, .0085] Ohm` (default) | matrix of nonnegative numbers

Lookup data for series resistance of the battery at the specified SOC during the charging phase.

Dependencies

This parameter is visible only when the **Temperature dependent tables** parameter is set to `No` and the **Current directionality** parameter is set to `Enabled`.

Cell capacity, AH — Battery capacity when fully charged
`27 hr*A` (default) | nonnegative scalar

Cell capacity of the battery. The block calculates the state of charge by dividing the accumulated charge by this value. The block calculates accumulated charge by integrating the battery current.

Self-discharge — Select whether to model the self-discharge resistance of the battery
`Disabled` (default) | `Enabled`

Select whether to model the self-discharge resistance of the battery. The block models this effect as a resistor across the terminals of the fundamental battery model.

As temperature increases, self-discharge resistance decreases, causing self-discharge to increase. If the decrease in resistance is too fast, thermal runaway of the battery and numerical instability can occur. You can resolve this instability by making any of these changes:

- Decrease the thermal resistance
- Decrease the gradient of the self-discharge resistance with respect to temperature
- Increase the self-discharge resistance

Self-discharge resistance, Rleak(T) — Resistance that represents battery self-discharge at temperature breakpoints
`[8000, 7000, 6000] Ohm` (default) | vector of positive numbers

Lookup data for self-discharge resistance of the battery at the specified temperature breakpoints. This resistance acts across the terminals of the fundamental battery model.

Dependencies

Enabled when the **Self-discharge** parameter is set to `Enabled` and **Temperature dependent tables** is set to `Yes`.

Self-discharge resistance, Rleak — Resistance that represents battery self-discharge
`7e3 Ohm` (default) | positive scalar

Lookup data for self-discharge resistance of the battery. This resistance acts across the terminals of the fundamental battery model.

Dependencies

This parameter is visible only when the **Self-discharge** parameter is set to `Enabled` and the **Temperature dependent tables** parameter is set to `No`.

.

Extrapolation method for all tables — Method of extrapolation for tables
`Nearest` (default) | `Linear` | `Error`

Extrapolation method for all the table based parameters:

- `Linear` — Estimates values beyond the dataset by creating a tangent line at the end of the known data and extending it beyond that limit.
- `Nearest` — Extrapolates a value at query point that is the value at the nearest sample grid point.
- `Error` — Returns an error if the value goes beyond the known dataset. If you select this option, the block does not use extrapolation.

Expose SOC measurement port — Method of extrapolation for tables
`No` (default) | `Yes`

Whether to expose the state-of-charge measurement port.

Dynamics

Charge dynamics — Battery charge dynamics model
`No dynamics` (default) | `One time-constant dynamics` | `Two time-constant dynamics` | `Three time-constant dynamics` | `Four time-constant dynamics` | `Five time-constant dynamics`

Select how to model battery charge dynamics. This parameter determines the number of parallel RC sections in the equivalent circuit:

- `No dynamics` — The equivalent circuit contains no parallel RC sections. There is no delay between terminal voltage and internal charging voltage of the battery.
- `One time-constant dynamics` — The equivalent circuit contains one parallel RC section. Specify the time constant using the **First time constant, tau1(SOC,T)** parameter if tabulating parameters over temperature or **First time constant, tau1(SOC)** otherwise.
- `Two time-constant dynamics` — The equivalent circuit contains two parallel RC sections. Specify the time constants using the **First time constant, tau1(SOC,T)** and **Second time constant, tau2(SOC,T)** parameters if tabulating parameters over temperature or **First time constant, tau1(SOC)** and **Second time constant, tau2(SOC)** otherwise.

- `Three time-constant dynamics` — The equivalent circuit contains three parallel RC sections. Specify the time constants using the **First time constant, tau1(SOC,T)**, **Second time constant, tau2(SOC,T)**, and **Third time constant, tau3(SOC,T)** parameters if tabulating parameters over temperature or **First time constant, tau1(SOC)**, **Second time constant, tau2(SOC)**, and **Third time constant, tau3(SOC)** otherwise.
- `Four time-constant dynamics` — The equivalent circuit contains four parallel RC sections. Specify the time constants using the **First time constant, tau1(SOC,T)**, **Second time constant, tau2(SOC,T)**, **Third time constant, tau3(SOC,T)**, and **Fourth time constant, tau4(SOC,T)** parameters if tabulating parameters over temperature or **First time constant, tau1(SOC)**, **Second time constant, tau2(SOC)**, **Third time constant, tau3(SOC)**, and **Fourth time constant, tau4(SOC)** otherwise.
- `Five time-constant dynamics` — The equivalent circuit contains five parallel RC sections. Specify the time constants using the **First time constant, tau1(SOC,T)**, **Second time constant, tau2(SOC,T)**, **Third time constant, tau3(SOC,T)**, **Fourth time constant, tau4(SOC,T)**, and **Fifth time constant, tau5(SOC,T)** parameters if tabulating parameters over temperature or **First time constant, tau1(SOC)**, **Second time constant, tau2(SOC)**, **Third time constant, tau3(SOC)**, **Fourth time constant, tau4(SOC)**, and **Fifth time constant, tau5(SOC)** otherwise.

First polarization resistance, R1(SOC,T) — First RC resistance at temperature breakpoints
`[.0109, .0029, .0013; .0069, .0024, .0012; .0047, .0026, .0013; .0034, .0016, .001; .0033, .0023, .0014; .0033, .0018, .0011; .0028, .0017, .0011] Ohm` (default) | matrix of positive numbers

Lookup data for the first parallel RC resistance at the specified SOC and temperature breakpoints. This parameter primarily affects the ohmic losses of the RC section.

Dependencies

To enable this parameter, set **Charge dynamics** to `One time-constant dynamics`, `Two time-constant dynamics`, `Three time-constant dynamics`, `Four time-constant dynamics`, or `Five time-constant dynamics` and **Temperature dependent tables** to `Yes`.

First polarization resistance, R1(SOC) — First RC resistance
`[.0029, .0024, .0026, .0016, .0023, .0018, .0017] Ohm` (default) | matrix of positive numbers

Lookup data for the first parallel RC resistance at the specified SOC. This parameter primarily affects the ohmic losses of the RC section.

Dependencies

To enable this parameter, set **Charge dynamics** to `One time-constant dynamics`, `Two time-constant dynamics`, `Three time-constant dynamics`, `Four time-constant dynamics`, or `Five time-constant dynamics` and **Temperature dependent tables** to `No`.

First time constant, tau1(SOC,T) — First RC time constant at temperature breakpoints
`[20, 36, 39; 31, 45, 39; 109, 105, 61; 36, 29, 26; 59, 77, 67; 40, 33, 29; 25, 39, 33] s` (default) | matrix of positive numbers

Lookup data for the first parallel RC time constant at the specified SOC and temperature breakpoints.

Dependencies

To enable this parameter, set **Charge dynamics** to `One time-constant dynamics`, `Two time-constant dynamics`, `Three time-constant dynamics`, `Four time-constant dynamics`, or `Five time-constant dynamics` and **Temperature dependent tables** to `Yes`.

First time constant, tau1(SOC) — First RC time constant
`[36, 45, 105, 29, 77, 33, 39]` `s` (default) | matrix of positive numbers

Lookup data for the first parallel RC time constant at the specified SOC.

Dependencies

To enable this parameter, set **Charge dynamics** to `One time-constant dynamics`, `Two time-constant dynamics`, `Three time-constant dynamics`, `Four time-constant dynamics`, or `Five time-constant dynamics` and **Temperature dependent tables** to `No`.

Second polarization resistance, R2(SOC,T) — Second RC resistance at temperature breakpoints
`[.0109, .0029, .0013; .0069, .0024, .0012; .0047, .0026, .0013; .0034, .0016, .001; .0033, .0023, .0014; .0033, .0018, .0011; .0028, .0017, .0011]` `Ohm` (default) | matrix of positive numbers

Lookup data for the second parallel RC resistance at the specified SOC and temperature breakpoints. This parameter primarily affects the ohmic losses of the RC section.

Dependencies

To enable this parameter, set **Charge dynamics** to `Two time-constant dynamics`, `Three time-constant dynamics`, `Four time-constant dynamics`, or `Five time-constant dynamics` and **Temperature dependent tables** to `Yes`.

Second polarization resistance, R2(SOC) — Second RC resistance
`[.0029, .0024, .0026, .0016, .0023, .0018, .0017]` `Ohm` (default) | matrix of positive numbers

Lookup data for the second parallel RC resistance at the specified SOC. This parameter primarily affects the ohmic losses of the RC section.

Dependencies

To enable this parameter, set **Charge dynamics** to `Two time-constant dynamics`, `Three time-constant dynamics`, `Four time-constant dynamics`, or `Five time-constant dynamics` and **Temperature dependent tables** to `No`.

Second time constant, tau2(SOC,T) — Second RC time constant at temperature breakpoints
`[20, 36, 39; 31, 45, 39; 109, 105, 61; 36, 29, 26; 59, 77, 67; 40, 33, 29; 25, 39, 33]` `s` (default) | matrix of positive numbers

Lookup data for the second parallel RC time constant at the specified SOC and temperature breakpoints.

Dependencies

To enable this parameter, set **Charge dynamics** to `Two time-constant dynamics`, `Three time-constant dynamics`, `Four time-constant dynamics`, or `Five time-constant dynamics` and **Temperature dependent tables** to `Yes`.

Second time constant, tau2(SOC) — Second RC time constant
`[36, 45, 105, 29, 77, 33, 39]` `s` (default) | matrix of positive numbers

Lookup data for the second parallel RC time constant at the specified SOC.

Dependencies

To enable this parameter, set **Charge dynamics** to `Two time-constant dynamics`, `Three time-constant dynamics`, `Four time-constant dynamics`, or `Five time-constant dynamics` and **Temperature dependent tables** to `No`.

Third polarization resistance, R3(SOC,T) — Third RC resistance at temperature breakpoints
`[.0109, .0029, .0013; .0069, .0024, .0012; .0047, .0026, .0013; .0034, .0016, .001; .0033, .0023, .0014; .0033, .0018, .0011; .0028, .0017, .0011]` `Ohm` (default) | matrix of positive numbers

Lookup data for the third parallel RC resistance at the specified SOC and temperature breakpoints. This parameter primarily affects the ohmic losses of the RC section and **Temperature dependent tables** to `Yes`.

Dependencies

To enable this parameter, set **Charge dynamics** to `Three time-constant dynamics`, `Four time-constant dynamics`, or `Five time-constant dynamics`.

Third polarization resistance, R3(SOC) — Third RC resistance
`[.0029, .0024, .0026, .0016, .0023, .0018, .0017]` `Ohm` (default) | matrix of positive numbers

Lookup data for the third parallel RC resistance at the specified SOC. This parameter primarily affects the ohmic losses of the RC section.

Dependencies

To enable this parameter, set **Charge dynamics** to `Three time-constant dynamics`, `Four time-constant dynamics`, or `Five time-constant dynamics` and **Temperature dependent tables** to `No`.

Third time constant, tau3(SOC,T) — Third RC time constant at temperature breakpoints
`[20, 36, 39; 31, 45, 39; 109, 105, 61; 36, 29, 26; 59, 77, 67; 40, 33, 29; 25, 39, 33]` `s` (default) | matrix of positive numbers

Lookup data for the third parallel RC time constant at the specified SOC and temperature breakpoints.

Dependencies

To enable this parameter, set **Charge dynamics** to `Three time-constant dynamics`, `Four time-constant dynamics`, or `Five time-constant dynamics` and **Temperature dependent tables** to `Yes`.

Third time constant, tau3(SOC) — Third RC time constant
`[36, 45, 105, 29, 77, 33, 39]` `s` (default) | matrix of positive numbers

Lookup data for the third parallel RC time constant at the specified SOC.

Dependencies

To enable this parameter, set **Charge dynamics** to `Three time-constant dynamics`, `Four time-constant dynamics`, or `Five time-constant dynamics` and **Temperature dependent tables** to `No`.

Fourth polarization resistance, R4(SOC,T) — Fourth RC resistance at temperature breakpoints
`[.014, .382, .407; .028, .006, .007; .014, .007, .006; .333, .956, .912]` `Ohm` (default) | matrix of positive numbers

Lookup data for the fourth parallel RC resistance at the specified SOC and temperature breakpoints. This parameter primarily affects the ohmic losses of the RC section.

Dependencies

To enable this parameter, set **Charge dynamics** to `Four time-constant dynamics` or `Five time-constant dynamics` and **Temperature dependent tables** to `Yes`.

Fourth polarization resistance, R4(SOC) — Fourth RC resistance
`[.0109, .0029, .0013; .0069, .0024, .0012; .0047, .0026, .0013; .0034, .0016, .001; .0033, .0023, .0014; .0033, .0018, .0011; .0028, .0017, .0011]` `Ohm` (default) | matrix of positive numbers

Lookup data for the fourth parallel RC resistance at the specified SOC. This parameter primarily affects the ohmic losses of the RC section.

Dependencies

To enable this parameter, set **Charge dynamics** to `Four time-constant dynamics` or `Five time-constant dynamics` and **Temperature dependent tables** to `No`.

Fourth time constant, tau4(SOC,T) — Fourth RC time constant at temperature breakpoints
`[20, 36, 39; 31, 45, 39; 109, 105, 61; 36, 29, 26; 59, 77, 67; 40, 33, 29; 25, 39, 33]` `s` (default) | matrix of positive numbers

Lookup data for the fourth parallel RC time constant at the specified SOC and temperature breakpoints.

Dependencies

To enable this parameter, set **Charge dynamics** to `Four time-constant dynamics` or `Five time-constant dynamics` and **Temperature dependent tables** to `Yes`.

Fourth time constant, tau4(SOC) — Fourth RC time constant
`[36, 45, 105, 29, 77, 33, 39]` `s` (default) | matrix of positive numbers

Lookup data for the fourth parallel RC time constant at the specified SOC.

Dependencies

To enable this parameter, set **Charge dynamics** to `Four time-constant dynamics` or `Five time-constant dynamics` and **Temperature dependent tables** to `No`.

Fifth polarization resistance, R5(SOC,T) — Fifth RC resistance at temperature breakpoints
`[.0109, .0029, .0013; .0069, .0024, .0012; .0047, .0026, .0013; .0034, .0016, .001; .0033, .0023, .0014; .0033, .0018, .0011; .0028, .0017, .0011]` `Ohm` (default) | matrix of positive numbers

Lookup data for the fifth parallel RC resistance at the specified SOC and temperature breakpoints. This parameter primarily affects the ohmic losses of the RC section.

Dependencies

To enable this parameter, set **Charge dynamics** to `Five time-constant dynamics` and **Temperature dependent tables** to `Yes`.

Fifth polarization resistance, R5(SOC) — Fifth RC resistance
`[.0029, .0024, .0026, .0016, .0023, .0018, .0017] Ohm` (default) | matrix of positive numbers

Lookup data for the fifth parallel RC resistance at the specified SOC. This parameter primarily affects the ohmic losses of the RC section.

Dependencies

To enable this parameter, set **Charge dynamics** to `Five time-constant dynamics` and **Temperature dependent tables** to `No`.

Fifth time constant, tau5(SOC,T) — Fifth RC time constant at temperature breakpoints
`[20, 36, 39; 31, 45, 39; 109, 105, 61; 36, 29, 26; 59, 77, 67; 40, 33, 29; 25, 39, 33] s` (default) | matrix of positive numbers

Lookup data for the fifth parallel RC time constant at the specified SOC and temperature breakpoints.

Dependencies

To enable this parameter, set **Charge dynamics** to `Five time-constant dynamics` and **Temperature dependent tables** to `Yes`.

Fifth time constant, tau5(SOC) — Fifth RC time constant
`[36, 45, 105, 29, 77, 33, 39] s` (default) | matrix of positive numbers

Lookup data for the fifth parallel RC time constant at the specified SOC.

Dependencies

To enable this parameter, set **Charge dynamics** to `Five time-constant dynamics` and **Temperature dependent tables** to `No`.

Fade

Fade characteristic — Select how to define fade characteristic
`Disabled` (default) | `Equations` | `Lookup tables (temperature independent)` | `Lookup tables (temperature dependent)`

Select how to define fade characteristic:

- `Disabled` — The block does not model fade characteristic.
- `Equations` — The cell capacity and terminal resistance will be proportional to $\sqrt{N}$ whilst the open-circuit voltage will be proportional to N. If the self-discharge resistance or any number of the time constants are enabled, their values will be proportional to $\sqrt{N}$.
- `Lookup tables (temperature independent)` — Set tabulated data for the percentage change in parameters as a function of N.

- `Lookup tables (temperature dependent)` — Set tabulated data for the percentage change in parameters as a function of *N* and temperature.

Number of discharge cycles, N — Reference number of cycles for percent change calculations
`100` (default) | number from `1` to `Inf`

The number of charge-discharge cycles over which the specified percent changes occur.

Dependencies

To enable this parameter, set **Fade characteristic** to `Equations`.

Change in open-circuit voltage after N discharge cycles (%) — Percent change in open-circuit voltage after *N* cycles
`0` (default) | scalar

Percent change in the open-circuit voltage after the battery undergoes *N* discharge cycles.

Dependencies

To enable this parameter, set **Fade characteristic** to `Equations`.

Change in terminal resistance after N discharge cycles (%) — Percent change in series resistance after *N* cycles
`0` (default) | scalar

Percent change in the series resistance after the battery undergoes *N* discharge cycles.

Dependencies

To enable this parameter, set **Fade characteristic** to `Equations`.

Change in cell capacity after N discharge cycles (%) — Percent change in cell capacity after *N* cycles
`0` (default) | scalar

Percent change in the cell capacity after the battery undergoes *N* discharge cycles.

Dependencies

To enable this parameter, set **Fade characteristic** to `Equations`.

Change in self-discharge resistance after N discharge cycles (%) — Percent change in self-discharge resistance after *N* cycles
`0` (default) | scalar

Percent change in the self-discharge resistance after the battery undergoes *N* discharge cycles.

Dependencies

To enable this parameter, set **Self-discharge** to `Enabled` and **Fade characteristic** to `Equations`.

Change in first polarization resistance after N discharge cycles (%) — Percent change in first RC resistance after *N* cycles
`0` (default) | scalar

Percent change in the first RC resistance after the battery undergoes *N* discharge cycles.

Dependencies

To enable this parameter, set **Charge dynamics** to `One time-constant dynamics`, `Two time-constant dynamics`, `Three time-constant dynamics`, `Four time-constant dynamics`, or `Five time-constant dynamics` and **Fade characteristic** to `Equations`.

Change in second polarization resistance after N discharge cycles (%) — Percent change in second RC resistance after *N* cycles
`0` (default) | scalar

Percent change in the second RC resistance after the battery undergoes *N* discharge cycles.

Dependencies

To enable this parameter, set **Charge dynamics** to `Two time-constant dynamics`, `Three time-constant dynamics`, `Four time-constant dynamics`, or `Five time-constant dynamics` and **Fade characteristic** to `Equations`.

Change in third polarization resistance after N discharge cycles (%) — Percent change in third RC resistance after *N* cycles
`0` (default) | scalar

Percent change in the third RC resistance after the battery undergoes *N* discharge cycles.

Dependencies

To enable this parameter, set **Charge dynamics** to `Three time-constant dynamics`, `Four time-constant dynamics`, or `Five time-constant dynamics` and **Fade characteristic** to `Equations`.

Change in fourth polarization resistance after N discharge cycles (%) — Percent change in fourth RC resistance after *N* cycles
`0` (default) | scalar

Percent change in the fourth RC resistance after the battery undergoes *N* discharge cycles.

Dependencies

To enable this parameter, set **Charge dynamics** to `Four time-constant dynamics`, or `Five time-constant dynamics` and **Fade characteristic** to `Equations`.

Change in fifth polarization resistance after N discharge cycles (%) — Percent change in fifth RC resistance after *N* cycles
`0` (default) | scalar

Percent change in the fifth RC resistance after the battery undergoes *N* discharge cycles.

Dependencies

To enable this parameter, set **Charge dynamics** to `Five time-constant dynamics` and **Fade characteristic** to `Equations`.

Vector of discharge cycle values, N — Reference vector or numbers of cycles for percent change calculations
`[100, 200, 300]` (default) | number from `1` to `Inf`

Vector of numbers of charge-discharge cycles over which the specified percent changes occur.

Dependencies

To enable this parameter, set **Fade characteristic** to `Lookup tables (temperature independent)` or `Lookuptables (temperature dependent)`.

Vector of temperatures for fade data, Tfade — Vector temperatures at which fade lookup tables has been extracted
`[298.15, 323.15]` K (default) | number from 1 to `Inf`

Vector of temperatures at which fade lookup tables has been extracted. These temperatures are completely independent of **Vectors of temperatures, T** parameter from the **Main** tab.

Dependencies

To enable this parameter, set **Fade characteristic** to `Lookup tables (temperature dependent)`.

Percentage change in open-circuit voltage, dV0(N) — Vector of percent change in open-circuit voltage after *N* cycles
`[0, 0, 0]` (default) | vector of scalars

Vector of percent changes in the open-circuit voltage after the battery undergoes *N* discharge cycles.

Dependencies

To enable this parameter, set **Fade characteristic** to `Lookup tables (temperature independent)`.

Percentage change in terminal resistance, dR0(N) — Vector of percent change in series resistance after *N* cycles
`[0, 0, 0]` (default) | vector of scalars

Vector of percent change in the series resistance after the battery undergoes *N* discharge cycles.

Dependencies

To enable this parameter, set **Fade characteristic** to `Lookup tables (temperature independent)`.

Percentage change in cell capacity, dAH(N) — Vector of percent change in cell capacity after *N* cycles
`[0, 0, 0]` (default) | vector of scalars

Vector of percent change in the cell capacity after the battery undergoes *N* discharge cycles.

Dependencies

To enable this parameter, set **Fade characteristic** to `Lookup tables (temperature independent)`.

Percentage change in self-discharge resistance, dRleak(N) — Vector of percent change in self-discharge resistance after *N* cycles
`[0, 0, 0]` (default) | vector of scalars

Vector of percent change in the self-discharge resistance after the battery undergoes *N* discharge cycles.

Dependencies

To enable this parameter, set **Self-discharge** to `Enabled` and **Fade characteristic** to `Lookup tables (temperature independent)`.

Percentage change in first polarization resistance, dR1(N) — Vector of percent change in first RC resistance after *N* cycles
`[0, 0, 0]` (default) | vector of scalars

Vector of percent change in the first RC resistance after the battery undergoes *N* discharge cycles.

Dependencies

To enable this parameter, set **Charge dynamics** to `One time-constant dynamics`, `Two time-constant dynamics`, `Three time-constant dynamics`, `Four time-constant dynamics`, or `Five time-constant dynamics` and **Fade characteristic** to `Lookup tables (temperature independent)`.

Percent change in second polarization resistance, dR2(N) — Vector of percent change in second RC resistance after *N* cycles
`[0, 0, 0]` (default) | vector of scalars

Vector of percent change in the second RC resistance after the battery undergoes *N* discharge cycles.

Dependencies

To enable this parameter, set **Charge dynamics** to `Two time-constant dynamics`, `Three time-constant dynamics`, `Four time-constant dynamics`, or `Five time-constant dynamics` and **Fade characteristic** to `Lookup tables (temperature independent)`.

Percent change in third polarization resistance, dR3(N) — Vector of percent change in third RC resistance after *N* cycles
`[0, 0, 0]` (default) | vector of scalars

Vector of percent change in the third RC resistance after the battery undergoes *N* discharge cycles.

Dependencies

To enable this parameter, set **Charge dynamics** to `Three time-constant dynamics`, `Four time-constant dynamics`, or `Five time-constant dynamics` and **Fade characteristic** to `Lookup tables (temperature independent)`.

Percent change in fourth polarization resistance, dR4(N) — Vector of percent change in fourth RC resistance after *N* cycles
`[0, 0, 0]` (default) | vector of scalars

Vector of percent change in the fourth RC resistance after the battery undergoes *N* discharge cycles.

Dependencies

To enable this parameter, set **Charge dynamics** to `Four time-constant dynamics`, or `Five time-constant dynamics` and **Fade characteristic** to `Lookup tables (temperature independent)`.

Percent change in fifth polarization resistance, dR5(N) — Vector of percent change in fifth RC resistance after *N* cycles
`[0, 0, 0]` (default) | vector of scalars

Vector of percent change in the fifth RC resistance after the battery undergoes *N* discharge cycles.

Dependencies

To enable this parameter, set **Charge dynamics** to `Five time-constant dynamics` and **Fade characteristic** to `Lookup tables (temperature independent)`.

Percentage change in open-circuit voltage, dV0(N, Tfade) — Matrix of percent change in open-circuit voltage after *N* cycles
`[0, 0; 0, 0; 0, 0]` (default) | matrix of scalars

Matrix of percent changes in the open-circuit voltage after the battery undergoes *N* discharge cycles.

Dependencies

To enable this parameter, set **Fade characteristic** to `Lookup tables (temperature dependent)`.

Percentage change in terminal resistance, dR0(N, Tfade) — Matrix of percent change in series resistance after *N* cycles
`[0, 0; 0, 0; 0, 0]` (default) | matrix of scalars

Matrix of percent change in the series resistance after the battery undergoes *N* discharge cycles.

Dependencies

To enable this parameter, set **Fade characteristic** to `Lookup tables (temperature dependent)`.

Percentage change in cell capacity, dAH(N, Tfade) — Matrix of percent change in cell capacity after *N* cycles
`[0, 0; 0, 0; 0, 0]` (default) | matrix of scalars

Matrix of percent change in the cell capacity after the battery undergoes *N* discharge cycles.

Dependencies

To enable this parameter, set **Fade characteristic** to `Lookup tables (temperature dependent)`.

Percentage change in self-discharge resistance, dRleak(N, Tfade) — Matrix of percent change in self-discharge resistance after *N* cycles
`[0, 0; 0, 0; 0, 0]` (default) | matrix of scalars

Matrix of percent change in the self-discharge resistance after the battery undergoes *N* discharge cycles.

Dependencies

To enable this parameter, set **Self-discharge** to `Enabled` and **Fade characteristic** to `Lookup tables (temperature dependent)`.

Percentage change in first polarization resistance, dR1(N, Tfade) — Matrix of percent change in first RC resistance after *N* cycles
`[0, 0; 0, 0; 0, 0]` (default) | matrix of scalars

Matrix of percent change in the first RC resistance after the battery undergoes *N* discharge cycles.

Dependencies

To enable this parameter, set **Charge dynamics** to `One time-constant dynamics`, `Two time-constant dynamics`, `Three time-constant dynamics`, `Four time-constant dynamics`, or `Five time-constant dynamics` and **Fade characteristic** to `Lookup tables (temperature dependent)`.

Percent change in second polarization resistance, dR2(N, Tfade) — Matrix of percent change in second RC resistance after *N* cycles
`[0, 0; 0, 0; 0, 0]` (default) | matrix of scalars

Matrix of percent change in the second RC resistance after the battery undergoes *N* discharge cycles.

Dependencies

To enable this parameter, set **Charge dynamics** to `Two time-constant dynamics`, `Three time-constant dynamics`, `Four time-constant dynamics`, or `Five time-constant dynamics` and **Fade characteristic** to `Lookup tables (temperature dependent)`.

Percent change in third polarization resistance, dR3(N, Tfade) — Matrix of percent change in third RC resistance after *N* cycles
`[0, 0; 0, 0; 0, 0]` (default) | matrix of scalars

Matrix of percent change in the third RC resistance after the battery undergoes *N* discharge cycles.

Dependencies

To enable this parameter, set **Charge dynamics** to `Three time-constant dynamics`, `Four time-constant dynamics`, or `Five time-constant dynamics` and **Fade characteristic** to `Lookup tables (temperature dependent)`.

Percent change in fourth polarization resistance, dR4(N, Tfade) — Matrix of percent change in fourth RC resistance after *N* cycles
`[0, 0; 0, 0; 0, 0]` (default) | matrix of scalars

Matrix of percent change in the fourth RC resistance after the battery undergoes *N* discharge cycles.

Dependencies

To enable this parameter, set **Charge dynamics** to `Four time-constant dynamics`, or `Five time-constant dynamics` and **Fade characteristic** to `Lookup tables (temperature dependent)`.

Percent change in fifth polarization resistance, dR5(N, Tfade) — Matrix of percent change in fifth RC resistance after *N* cycles
`[0, 0; 0, 0; 0, 0]` (default) | matrix of scalars

Matrix of percent change in the fifth RC resistance after the battery undergoes *N* discharge cycles.

Dependencies

To enable this parameter, set **Charge dynamics** to `Five time-constant dynamics` and **Fade characteristic** to `Lookup tables (temperature dependent)`.

Calendar Aging

Internal resistance calendar aging — Calendar aging for internal resistance
`Disabled` (default) | `Enabled`

Whether to enable calendar aging of the internal resistance of the battery.

Capacity calendar aging — Calendar aging for capacity
`Disabled` (default) | `Enabled`

Whether to enable calendar aging of the capacity of the battery.

Modeling option — Calendar aging option
`Equation-based` (default) | `Tabulated: temperature` | `Tabulated: time and temperature`

Modeling option for internal resistance and capacity calendar aging.

Dependencies

To enable this parameter, set either **Internal resistance calendar aging** or **Capacity calendar aging** to `Enabled`.

Vector of time intervals — Time intervals
`[0] d` (default) | vector of scalars

Time intervals. This parameter must be equal in size to **Vector of storage temperatures**.

Dependencies

To enable this parameter, set **Modeling option** to either `Equation-based`, `Tabulated: temperature`, or `Tabulated: time and temperature`.

Vector of storage temperatures — Storage temperatures
`[273] K` (default) | vector of scalars

Set of storage temperatures. This parameter must be equal in size to **Vector of time intervals**.

Dependencies

To enable this parameter, set **Modeling option** to either `Equation-based`, `Tabulated: temperature`, or `Tabulated: time and temperature`.

Storage condition — Storage condition
`Specify open-circuit voltage during storage` (default) | `Specify state-of-charge during storage`

Whether to specify the open-circuit voltage or the state of charge during storage.

Dependencies

To enable this parameter, set **Modeling option** to `Equation-based`.

Normalized open-circuit voltage during storage, V/Vnom — Normalized open-circuit voltage during storage
`0.9` (default) | scalar

Normalized open-circuit voltage during storage.

Dependencies

To enable this parameter, set **Modeling option** to `Equation-based` and **Storage condition** to `Specify open-circuit voltage during storage`.

State of charge during storage — Percentage state of charge during storage
`60` (default) | positive number

State of charge during storage, in percentage.

Dependencies

To enable this parameter, set **Modeling option** to `Equation-based` and **Storage condition** to `Specify state-of-charge during storage`.

Terminal resistance linear scaling for voltage, b — Voltage linear scaling in terminal resistance
`2.2134e6` (default) | scalar

Linear scaling coefficient in terminal resistance for open-circuit voltage.

Dependencies

To enable this parameter, set **Internal resistance calendar aging** to `Enabled` and **Modeling option** to `Equation-based`.

Terminal resistance constant offset for voltage, c — Voltage constant offset in terminal resistance
`1.632e6` (default) | scalar

Constant offset in terminal resistance for open-circuit voltage.

Dependencies

To enable this parameter, set **Internal resistance calendar aging** to `Enabled` and **Modeling option** to `Equation-based`.

Terminal resistance temperature-dependent exponential increase, d — Temperature-dependent exponential increase in terminal resistance
`0.515833569 V` (default) | scalar

Temperature-dependent exponential increase in terminal resistance.

Dependencies

To enable this parameter, set **Internal resistance calendar aging** to `Enabled` and **Modeling option** to `Equation-based`.

Terminal resistance time exponent, a — Time exponent in terminal resistance
`0.75` (default) | scalar

Time exponent in terminal resistance.

Dependencies

To enable this parameter, set **Internal resistance calendar aging** to `Enabled` and **Modeling option** to either `Equation-based` or `Tabulated: temperature`.

Capacity linear scaling for voltage, b — Voltage linear scaling in capacity
`1.5097e07` (default) | scalar

Linear scaling coefficient in capacity for open-circuit voltage.

Dependencies

To enable this parameter, set **Capacity calendar aging** to `Enabled` and **Modeling option** to `Equation-based`.

Capacity constant offset for voltage, c — Voltage constant offset in capacity
`8.3625e06` (default) | scalar

Constant offset in capacity for open-circuit voltage.

Dependencies

To enable this parameter, set **Capacity calendar aging** to `Enabled` and **Modeling option** to `Equation-based`.

Capacity temperature-dependent exponential increase, d — Temperature-dependent exponential increase in capacity
`0.6011` V (default) | scalar

Temperature-dependent exponential increase in capacity.

Dependencies

To enable this parameter, set **Capacity calendar aging** to `Enabled` and **Modeling option** to `Equation-based`.

Capacity time exponent, a — Time exponent in capacity
`0.75` (default) | scalar

Time exponent in capacity.

Dependencies

To enable this parameter, set **Capacity calendar aging** to `Enabled` and **Modeling option** to either `Equation-based` or `Tabulated: temperature`.

Vector of sampled temperatures for terminal resistance calendar aging, T_ar — Sampled temperatures for terminal resistance calendar aging
`[273.15, 298.15, 323.15]` K (default) | vector of scalars

Vector of sampled temperatures for terminal resistance calendar aging.

Dependencies

To enable this parameter, set **Internal resistance calendar aging** to `Enabled` and **Modeling option** to either `Tabulated: temperature` or `Tabulated: time and temperature`.

Percentage change in terminal resistance due to calendar aging, dR0(T_ar) — Percentage change in terminal resistance due to calendar aging
`[0,0,0]` (default) | vector of scalars

Percentage change in terminal resistance due to calendar aging.

Dependencies

To enable this parameter, set **Internal resistance calendar aging** to `Enabled` and **Modeling option** to `Tabulated: time`.

Time between terminal resistance beginning of life and dR(T_ar) measurement — Time between terminal resistance beginning of life and dR(T_ar) measurement
`100` d (default) | scalar

Time between the beginning of life of the terminal resistance and the dR(T_ar) measurement.

Dependencies

To enable this parameter, set **Internal resistance calendar aging** to `Enabled` and **Modeling option** to `Tabulated: time`.

Vector of sampled temperatures for capacity calendar aging, T_ac — Sampled temperatures for capacity calendar aging
`[273.15, 298.15, 323.15]` K (default) | vector of scalars

Vector of sampled temperatures for capacity calendar aging.

Dependencies

To enable this parameter, set **Capacity calendar aging** to `Enabled` and **Modeling option** to either `Tabulated: temperature` or `Tabulated: time and temperature`.

Percentage change in capacity due to calendar aging, dAH(T_ac) — Percentage change in capacity due to calendar aging
`[0,0,0]` (default) | vector of scalars

Percentage change in capacity due to calendar aging.

Dependencies

To enable this parameter, set **Capacity calendar aging** to `Enabled` and **Modeling option** to `Tabulated: time`.

Time between capacity beginning of life and dAH(T_ac) measurement — Time between capacity beginning of life and dAH(T_ac) measurement
`100` d (default) | scalar

Time between the beginning of life of the capacity and the dR(T_ac) measurement.

Dependencies

To enable this parameter, set **Capacity calendar aging** to `Enabled` and **Modeling option** to `Tabulated: time`.

Vector of sampled storage time intervals for terminal resistance calendar aging, t_ar — Sampled storage time intervals for terminal resistance calendar aging
`[90, 180, 270, 360]` d (default) | vector of scalars

Vector of sampled storage time intervals for terminal resistance calendar aging.

Dependencies

To enable this parameter, set **Internal resistance calendar aging** to `Enabled` and **Modeling option** to `Tabulated: time and temperature`.

Percentage change in terminal resistance due to calendar aging, dR0(t_ar,T_ar) — Percentage change in terminal resistance due to calendar aging
`[0, 0, 0; 0, 0, 0; 0, 0, 0; 0, 0, 0]` (default) | matrix of scalars

Percentage change in terminal resistance due to calendar aging.

Dependencies

To enable this parameter, set **Internal resistance calendar aging** to `Enabled` and **Modeling option** to `Tabulated: time and temperature`.

Vector of sampled storage time intervals for capacity calendar aging, t_ac — Sampled storage time intervals for capacity calendar aging
`[90, 180, 270, 360]` d (default) | vector of scalars

Vector of sampled storage time intervals for capacity calendar aging.

Dependencies

To enable this parameter, set **Capacity calendar aging** to `Enabled` and **Modeling option** to `Tabulated: time and temperature`.

Percentage change in capacity due to calendar aging, dAH(t_ac,T_ac) — Percentage change in capacity due to calendar aging
`[0, 0, 0; 0, 0, 0; 0, 0, 0; 0, 0, 0]` (default) | matrix of scalars

Percentage change in capacity due to calendar aging.

Dependencies

To enable this parameter, set **Capacity calendar aging** to `Enabled` and **Modeling option** to `Tabulated: time and temperature`.

Thermal

Thermal port — Thermal port visibility
`Omit` (default) | `Model`

Whether to expose the thermal port.

Simulation temperature — Battery temperature
`298.15` K (default) | positive number

Battery temperature used in lookup tables throughout simulation.

Dependencies

To enable this parameter, set **Thermal port** to `Omit` and **Temperature dependent tables** to `Yes` or **Fade characteristic** to `Lookup tables (temperature dependent)`. For more information, see "Modeling Thermal Effects" on page 1-11.

Thermal mass — Thermal mass associated with the thermal port
`100` J/K (default)

Thermal mass associated with the thermal port H. It represents the energy required to raise the temperature of the thermal port by one degree.

Dependencies

To enable this parameter, set **Thermal port** to `Model`. For more information, see “Modeling Thermal Effects” on page 1-11.

Version History

Introduced in R2018a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

See Also

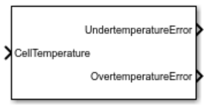
Battery

Topics

“List of Pre-Parameterized Components” (Simscape Electrical)

Battery Temperature Monitoring

Monitor for battery temperature



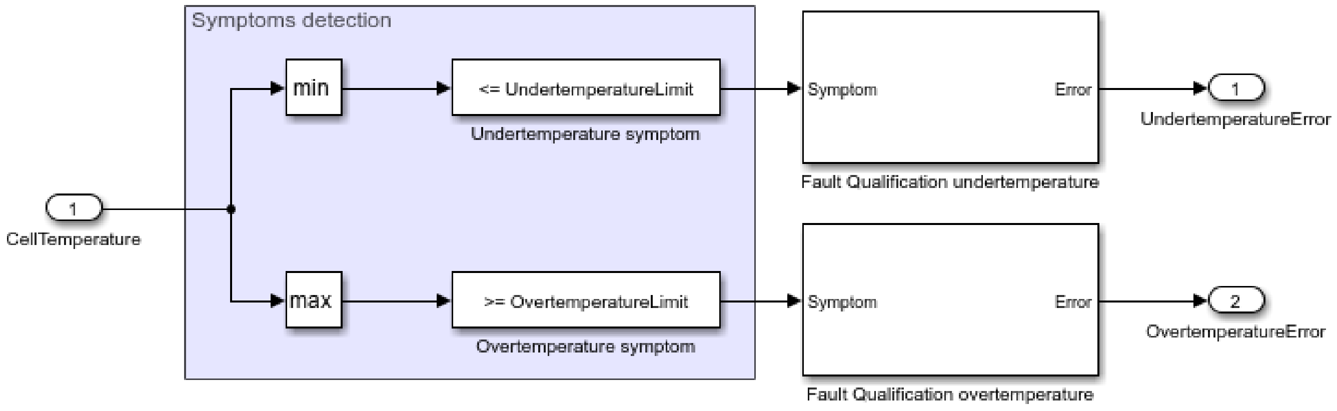
Libraries:
Simscape / Battery / BMS / Protection

Description

This block monitors the temperature of a battery. Temperature protection is necessary in electrical circuits. A battery that is exposed to extremely high or extremely low temperatures can enter a dangerously unstable state.

You can switch between continuous and discrete implementations of the block by using the **Sample time (-1 for inherited)** parameter. To configure the block for continuous time, set the **Sample time (-1 for inherited)** parameter to `0`. To configure the block for discrete time, set the **Sample time (-1 for inherited)** parameter to a positive, nonzero value, or to `-1` to inherit the sample time from an upstream block.

This diagram illustrates the overall structure of the block:



Equations

This block computes the undertemperature symptom with this equation:

$$Undertemperature\,Symptom = min(Temperature) \leq \; UndertemperatureLimit$$

This block computes the overtemperature symptom with this equation:

$$Overtemperature\,Symptom = \max(Temperature) \geq \; OvertemperatureLimit$$

The Battery Temperature Monitoring block then passes the undertermperature and overtemperature symptoms to a Fault Qualification block, which qualifies the error.

Ports

Input

Temperature — Cell temperature
scalar | vector

Temperature of the battery cell, specified as a scalar for a single cell or a vector for multiple cells.

Output

UndertemperatureError — Indication of undertemperature error
0 | 1

Indication of an undertemperature error. If this output is equal to 1, the battery is in an undertemperature state.

OvertemperatureError — Indication of overtemperature error
0 | 1

Indication of an overtemperature error. If this output is equal to 1, the battery is in an overtemperature state.

Parameters

Undertemperature limit — Undertemperature limit
270 (default) | nonnegative scalar

Temperature limit under which the battery is in an undertemperature state.

Overtemperature limit — Overtemperature limit
320 (default) | nonnegative scalar

Temperature limit over which the battery is in an undertemperature state. The value of this parameter must be greater than the value of the **Undertemperature limit** parameter.

Qualification time undertemperature (s) — Undertemperature qualification time
2 (default) | positive scalar

Time required to qualify the undertemperature error, in seconds.

Disqualification time undertemperature (s) — Undertemperature disqualification time
0 (default) | nonnegative scalar

Time required to disqualify the undertemperature error, in seconds. If you set this parameter to 0, the block does not disqualify the error.

Qualification time overtemperature (s) — Overtemperature qualification time
2 (default) | positive scalar

Time required to qualify the overtemperature error, in seconds.

Disqualification time overtemperature (s) — Overtemperature disqualification time
0 (default) | nonnegative scalar

Time required to disqualify the overtemperature error, in seconds. If you set this parameter to `0`, the block does not disqualify the error.

Sample time (-1 for inherited) — Block sample time
`-1` (default) | 0 | positive integer

Time between consecutive block executions. During execution, the block produces outputs and, if appropriate, updates its internal state. For more information, see “What Is Sample Time?” and “Specify Sample Time”.

For inherited discrete-time operation, specify this parameter as `-1`. For discrete-time operation, specify this parameter as a positive integer. For continuous-time operation, specify this parameter as `0`.

If this block is in a masked subsystem or a variant subsystem that allows you to switch between continuous operation and discrete operation, promote the sample time parameter. Promoting the sample time parameter ensures correct switching between the continuous and discrete implementations of the block. For more information, see “Promote Block Parameters on a Mask”.

Version History

Introduced in R2022b

Extended Capabilities

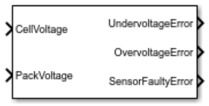
C/C++ Code Generation
Generate C and C++ code using Simulink® Coder™.

See Also

Battery Current Monitoring | Battery Voltage Monitoring | Fault Qualification

Battery Voltage Monitoring

Monitor for battery voltage



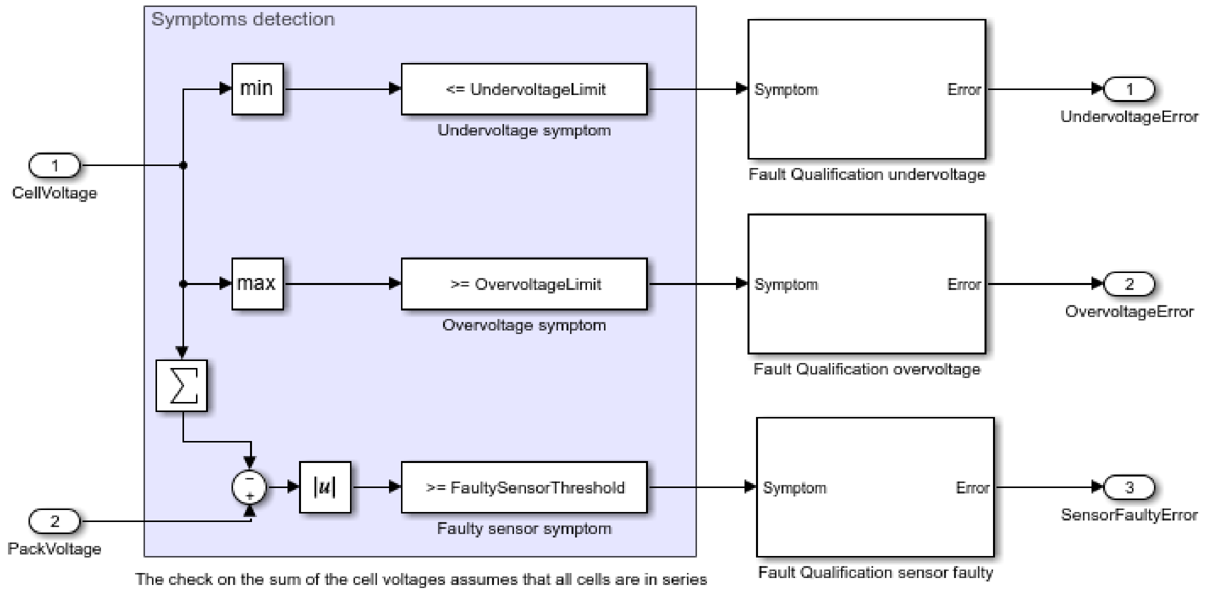
Libraries:
Simscape / Battery / BMS / Protection

Description

This block monitors the voltage of a battery. Voltage protection is necessary in electrical circuits. A battery that is exposed to extremely high or extremely low voltages can enter a dangerously unstable state.

You can switch between continuous and discrete implementations of the block by using the **Sample time (-1 for inherited)** parameter. To configure the block for continuous time, set the **Sample time (-1 for inherited)** parameter to `0`. To configure the block for discrete time, set the **Sample time (-1 for inherited)** parameter to a positive, nonzero value, or to `-1` to inherit the sample time from an upstream block.

This diagram shows the structure of the block:



Equations

This block computes the undervoltage symptom with this equation:

$$Undervoltage\,Symptom = min(CellVoltage) \leq \ UndervoltageLimit$$

This block computes the overvoltage symptom with this equation:

$$Overvoltage\,Symptom = \max(CellVoltage) \geq \ OvervoltageLimit$$

This block computes the faulty sensor symptom with this equation:

$$FaultySensor\,Symptom = \left| PackVoltage - \sum_{i-1}^{n} CellVoltage^{i} \right| \geq \ FaultySensorThreshold$$

The check on the sum of the cell voltages assumes that all the cells are in series.

The Battery Voltage Monitoring block then passes these symptoms to a Fault Qualification block, which qualifies the errors.

Ports

Input

CellVoltage — Cell voltage
scalar | vector

Cell voltage, in volt, specified as a scalar for a single cell or a vector for multiple cells.

PackVoltage — Battery pack voltage
scalar

Battery pack voltage, in volt, specified as a scalar.

Output

UndervoltageError — Indication of undervoltage error
0 | 1

Indication of an undervoltage error. If this output is equal to 1, the battery is in an undervoltage state.

OvervoltageError — Indication of overvoltage error
0 | 1

Indication of an overvoltage error. If this output is equal to 1, the battery is in an overvoltage state.

SensoryFaultyError — Indication of sensor faulty error
0 | 1

Indication of a sensor faulty error. If this output is equal to 1, the sensor is faulty.

Parameters

Undervoltage limit (V) — Undervoltage limit
2.5 (default) | positive scalar

Voltage limit under which the battery is in an undervoltage state, in volt.

Overvoltage limit (V) — Overvoltage limit
`4.5` (default) | positive scalar

Voltage limit over which the battery is in an overvoltage state, in volt. The value of this parameter must be greater than the value of the **Undervoltage limit (V)** parameter.

Voltage sensor fault threshold (V) — Threshold for voltage sensor fault
`0.25` (default) | positive scalar

Voltage threshold over which the sensor is faulty, in volt.

Qualification time undervoltage (s) — Undervoltage qualification time
`2` (default) | positive scalar

Time required to qualify the undervoltage error, in seconds.

Disqualification time undervoltage (s) — Undervoltage disqualification time
`0` (default) | nonnegative scalar

Time required to disqualify the undervoltage error, in seconds. If you set this parameter to `0`, the block does not disqualify the error.

Qualification time overvoltage (s) — Overvoltage qualification time
`2` (default) | positive scalar

Time required to qualify the overvoltage error, in seconds.

Disqualification time overvoltage (s) — Overvoltage disqualification time
`0` (default) | nonnegative scalar

Time required to disqualify the overvoltage error, in seconds. If you set this parameter to `0`, the block does not disqualify the error.

Qualification time faulty sensor (s) — Faulty sensor qualification time
`2` (default) | positive scalar

Time required to qualify the faulty sensor error, in seconds.

Disqualification time faulty sensor (s) — Faulty sensor disqualification time
`0` (default) | nonnegative scalar

Time required to disqualify the faulty sensor error, in seconds. If you set this parameter to `0`, the block does not disqualify the error.

Sample time (-1 for inherited) — Block sample time
`-1` (default) | 0 | positive integer

Time between consecutive block executions. During execution, the block produces outputs and, if appropriate, updates its internal state. For more information, see “What Is Sample Time?” and “Specify Sample Time”.

For inherited discrete-time operation, specify this parameter as `-1`. For discrete-time operation, specify this parameter as a positive integer. For continuous-time operation, specify this parameter as `0`.

If this block is in a masked subsystem or a variant subsystem that allows you to switch between continuous operation and discrete operation, promote the sample time parameter. Promoting the sample time parameter ensures correct switching between the continuous and discrete implementations of the block. For more information, see “Promote Block Parameters on a Mask”.

Version History

Introduced in R2022b

Extended Capabilities

C/C++ Code Generation

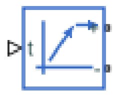
Generate C and C++ code using Simulink® Coder™.

See Also

Battery Current Monitoring | Battery Temperature Monitoring | Fault Qualification

Charger

Ideal battery charger

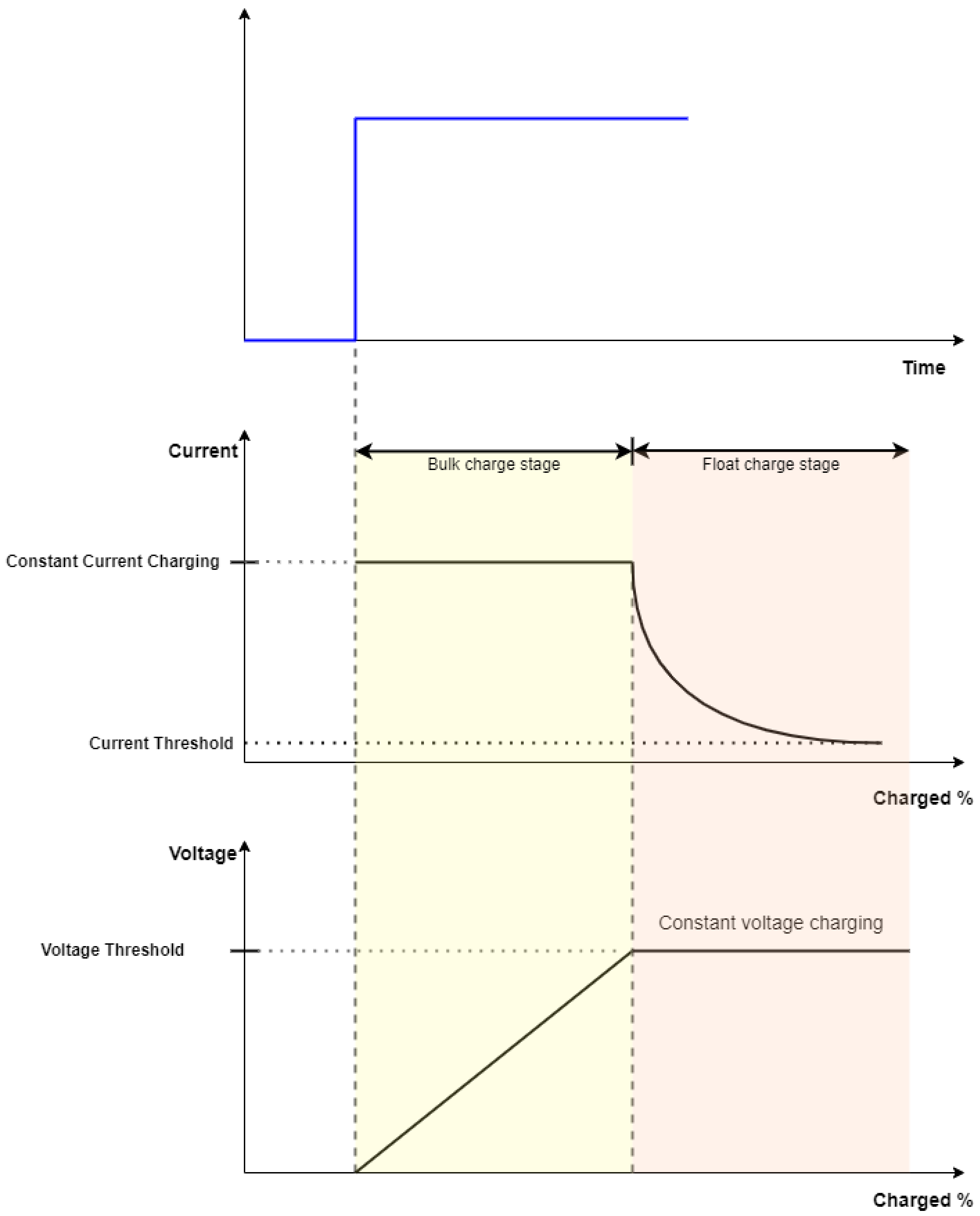


Libraries:
Simscape / Battery / Cyclers

Description

The Charger block implements an ideal battery charger. When the signal at the input port, **t**, changes from a value below `0.5` to a value equal to or greater than `0.5`, this block performs constant current charging at the value of the **Constant charging current** parameter. When the voltage is greater than or equal to the value of the **Voltage threshold** parameter, this block performs constant voltage charging at the value of the **Constant charging voltage** parameter until the current is less than or equal to the value of the **Current threshold** parameter.

This figure shows the behavior of the Charger block when the **t** input crosses `0.5`:



Ports

Input

t — Trigger, unitless
physical signal

Physical signal input port that controls the charging behavior of the block.

Conserving

+ — Positive terminal
electrical

Electrical conserving port associated with the positive terminal of the charger.

- — Negative terminal
electrical

Electrical conserving port associated with the negative terminal of the charger.

Parameters

Constant charging current — Constant charging current
`5` A (default)

Value of the constant charging current at which the block charges a battery.

Voltage threshold — Voltage threshold for constant current charging
`12` V (default)

Threshold at which the block stops performing a constant current charging on the battery.

Constant charging voltage — Constant charging voltage
`12` V (default)

Value of the constant charging voltage at which the block charges a battery.

Current threshold — Current threshold for constant voltage charging
`0.5` A (default)

Threshold at which the block stops performing a constant voltage charging on the battery.

Version History

Introduced in R2022b

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using Simulink® Coder™.

See Also

Discharger | Cycler

Pack

Generated Simscape model of battery pack

Description

The Pack block is a custom generated subsystem model of a battery pack. You can create this Simscape subsystem by using the `buildBattery` function with a `Pack` object as an input argument. The Pack subsystem block is inside the Simulink library that you specify in the `LibraryName` argument of the `buildBattery` function.

Note To allow for structural changes and further customization, when you add the Pack masked library block from the parent library to a Simulink® model, the linked block does not contain the link or path to the parent library block. If you modify any of the battery structural properties, such as `NumSeriesAssemblies` or `NumParallelCells`, you must rebuild the original parent library and copy the block in your model again.

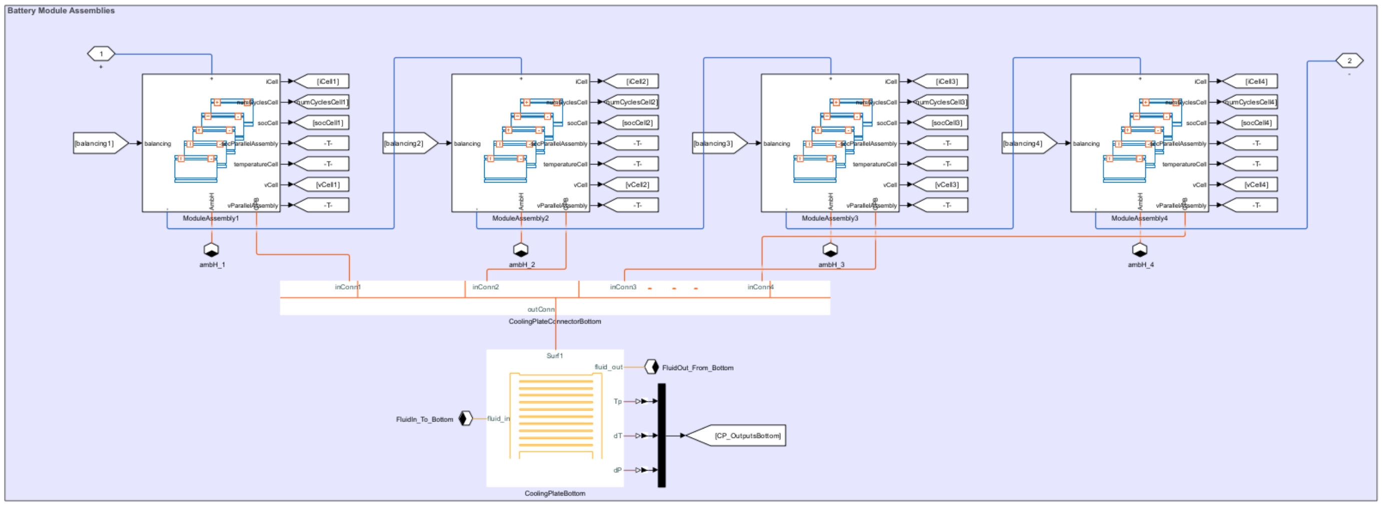
For more information about parent and linked blocks, see “Linked Blocks”.

The Pack subsystem model comprises four major sections.

- The Battery Module Assemblies section displays all the ModuleAssembly subsystems defined in the `Pack` object. This section contains all the series and parallel electrical circuit connections between the module assemblies, as well as the additional connections to the main electrical terminals and the ModuleAssembly-to-thermal boundary conditions. If you require additional non-ModuleAssembly resistances, then the function adds and connects two additional electrical resistors here.
- The Output signals section groups the output of each ModuleAssembly subsystem into a single multiplexed output signal for the Pack subsystem.
- The Thermal Boundary Conditions section contains the thermal boundary conditions for each ModuleAssembly subsystem. The `buildBattery` function adds a connector port from the thermal domain and routes it to every ModuleAssembly subsystem according to the specified thermal boundary conditions. This section is optional and only appears if you define an ambient or coolant thermal path in the `Pack` object.
- The Balancing Signals section contains the routing of the cell balancing control signal from the battery management system to every ModuleAssembly subsystem. Selector blocks route the correct elements of the control signal to their corresponding ParallelAssembly blocks.
- The Coolant Thermal Liquid section contains the thermal liquid network manifold for the coolant that circulates inside the battery cooling plates. This section is optional and only appears if you define a cooling plate in the `Pack` object.

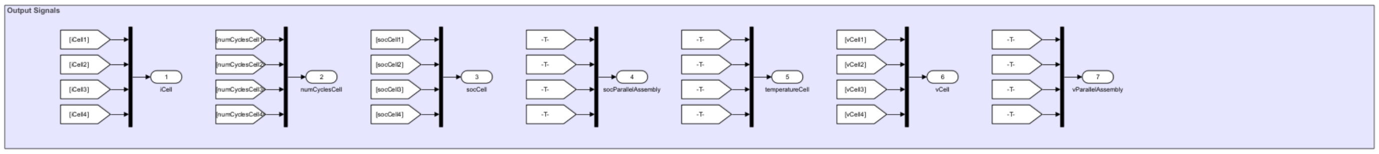
Battery Module Assemblies

This figure shows the Battery Module Assemblies section for a Pack subsystem made of four battery module assemblies.



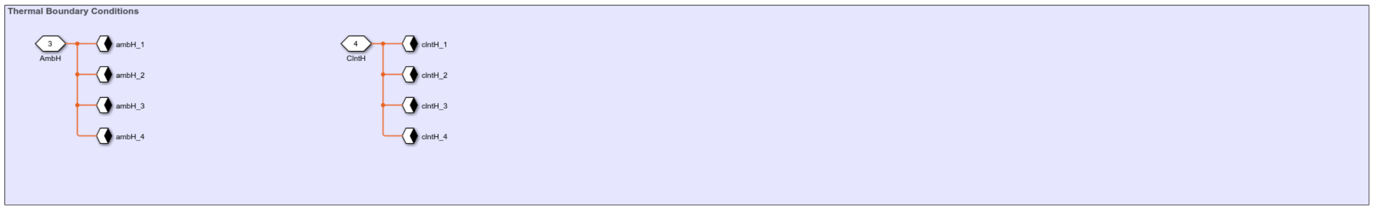
Output Signals

This figure shows the Output Signals section for a Pack subsystem made of four battery module assemblies.



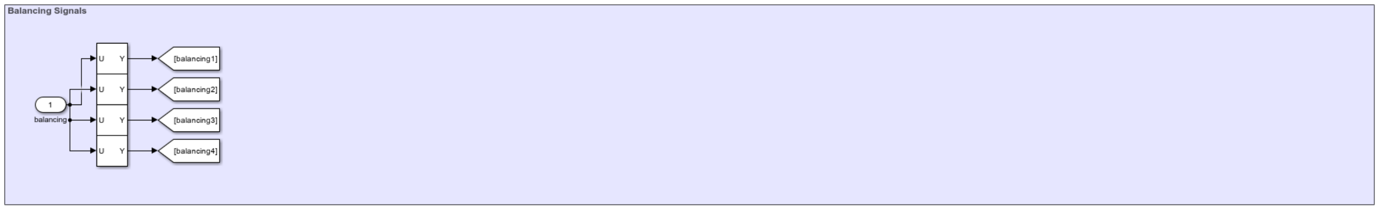
Thermal Boundary Conditions

This figure shows the Thermal Boundary Conditions section for a Pack subsystem made of four battery module assemblies.



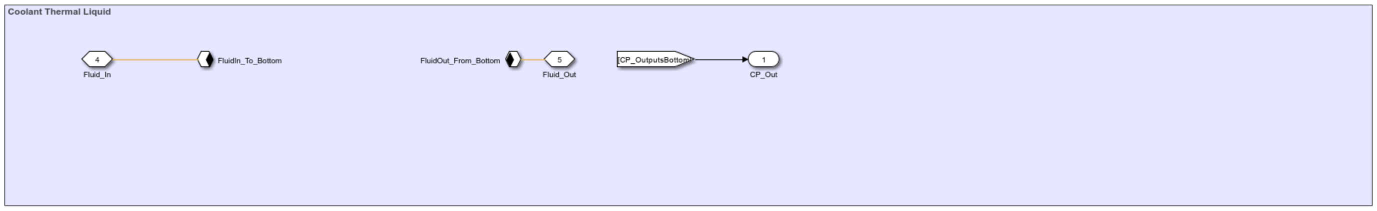
Balancing Signals

This figure shows the Balancing Signals section for a Pack subsystem made of four battery module assemblies.



Coolant Thermal Liquid

This figure shows the Coolant Thermal Liquid section for a Pack subsystem made of four battery module assemblies.



Ports

Input

balancing — Balancing switch control, unitless
scalar

Switch control port for the balancing strategy, specified as a scalar.

Dependencies

To enable this port, set the `BalancingStrategy` property of the `Pack` object to `"Passive"`.

Output

Note The names of the output ports depend on the block you have specified in the CellModelOptions property of the underlying `Cell` object that constitutes this pack.

The names of the output ports in this page refer to a `Cell` object with the Battery (Table-Based) block as the cell component model block. If you specify a custom block for the cell component model, the names of the output ports might differ.

---

iCell — Cell-level current, Ampere
scalar | vector

Current at the cell level. The original signal units from the Module block are Amperes. The size of this signal is equal to the total number of electrical models in the system.

numCyclesCell — Cell-level number of cycles, unitless
scalar | vector

Cell-level number of cycles related to battery aging. The original signal units from the Module block are unitless. The size of this signal is equal to the total number of electrical models in the system.

socCell — Cell-level state of charge, unitless
scalar | vector

State of charge at the cell level. The original signal units from the Module block are unitless and the value typically varies between 0 and 1. The size of this signal is equal to the total number of electrical models in the system.

socParallelAssembly — State of charge at parallel assembly level, unitless
scalar | vector

State of charge at the parallel assembly level. The original signal units from the Module block are unitless and the value typically varies between 0 and 1. The size of this signal is equal to the total number of parallel assemblies in the module assembly.

temperatureCell — Cell-level temperature, kelvin
scalar | vector

Temperature at the cell level. The original signal units from the Module block are kelvin. The size of this signal is equal to the total number of electrical models in the system.

Dependencies

To enable this port, enable the thermal port of the cells that comprise the pack. To model the thermal port of the cells, in the `CellModelBlock` property of the cells, set the **thermal_port** parameter of the `BlockParameters` property to `"Model"`.

vCell — Cell-level voltages, volts
scalar | vector

Voltages at the cell level. The original signal units from the Module block are volts. The size of this signal is equal to the total number of electrical models in the system.

vParallelAssembly — Parallel assembly-level voltages, volts
scalar | vector

Voltages at the parallel assembly level. The original signal units from the Module block are volts. The size of this signal is equal to the total number of parallel assemblies in the pack. Use this signal as an input to the battery management system (BMS) blocks of Simscape Battery.

CP_Out — Outputs of each cooling plate in model
vector of physical signals

Vector that contains the plate temperature, the fluid pressure drop, and the fluid temperature change of each cooling plate in the model. The size of this port depends on the number of cooling plates in the model.

Dependencies

To enable this port, set the `CoolingPlate` property of the `Pack` object to either `"Top"` or `"Bottom"`.

Conserving

+ — Positive terminal
electrical

Electrical conserving port associated with the positive terminal of the battery pack.

- — Negative terminal
electrical

Electrical conserving port associated with the negative terminal of the battery pack.

AmbH — Ambient thermal port
thermal

Thermal conserving port associated with the ambient thermal path.

Dependencies

To enable this port, set the `AmbientThermalPath` property of the `Pack` object to `"CellBasedThermalResistance"`.

ClntH — Coolant thermal port
thermal

Thermal conserving port associated with the coolant thermal path.

Dependencies

To enable this port, in the `Pack` object, set the `CoolantThermalPath` property to `"CellBasedThermalResistance"` and the `CoolingPlate` property to `"None"`.

Fluid_In — Fluid in cooling plate
thermal fluid

Thermal fluid conserving port associated with the fluid that comes in each cooling plate of the model. The size of this port depends on the number of cooling plates in the model.

Dependencies

To enable this port, set the `CoolingPlate` property of the `Pack` object to either `"Top"` or `"Bottom"`.

Fluid_Out — Fluid out of cooling plate
thermal fluid

Thermal fluid conserving port associated with the fluid that comes out of each cooling plate in the model. The size of this port depends on the number of cooling plates in the model.

Dependencies

To enable this port, set the `CoolingPlate` property of the `Pack` object to either `"Top"` or `"Bottom"`.

Version History

Introduced in R2022b

See Also

Apps
Battery Builder

Simscape Blocks
arrayOfThermalNodesConnector | ParallelAssembly (Generated Block) | Module (Generated Block) | ModuleAssembly (Generated Block)

Objects
`Pack`

Functions
`buildBattery`

Topics
"Manage Battery Run-Time Parameters with Centralized Script"

Module

Generated Simscape model of battery module

Description

The Module block represents a custom generated system model of a battery module. You can create this Simscape block by using the `buildBattery` function with a `Module` object as an input argument. The Module generated block is inside the Simulink library that you specify in the `LibraryName` argument of the `buildBattery` function.

Note To allow for structural changes and further customization, when you add the Module masked library block from the parent library to a Simulink model, the linked block does not contain the link or path to the parent library block. If you modify any of the battery structural properties, such as `NumSeriesAssemblies` or `NumParallelCells`, you must rebuild the original parent library and copy the block in your model again.

For more information about parent and linked blocks, see “Linked Blocks”.

Variables

To set the priority and initial target values for the block variables prior to simulation, use the **Initial Targets** section in the block dialog box or Property Inspector. For more information, see “Set Priority and Initial Target for Block Variables”.

For a detailed list of the variables in the ParallelAssembly block, see the “Initial Targets (Variables)” on page 1-99 section.

Nominal values provide a way to specify the expected magnitude of a variable in a model. Using system scaling based on nominal values increases the simulation robustness. Nominal values can come from different sources, one of which is the **Nominal Values** section in the block dialog box or Property Inspector. For more information, see “System Scaling by Nominal Values”.

Ports

Input

CB — Cell balancing, unitless
physical signal

Cell balancing port, specified as a physical signal.

Dependencies

To enable this port, set the `BalancingStrategy` property of the `Module` object to `"Passive"`. Alternatively, set the `BalancingStrategy` property to `"Passive"` inside the parent object of which the `Module` object is a subcomponent.

Conserving

+ — Positive terminal
electrical

Electrical conserving port associated with the positive terminal of the battery module.

- — Negative terminal
electrical

Electrical conserving port associated with the negative terminal of the battery module.

AH — Ambient thermal port
thermal

Thermal conserving port associated with the ambient thermal path.

Dependencies

To enable this port, set the `AmbientThermalPath` property of the `Module` object to `"CellBasedThermalResistance"`. Alternatively, set the `AmbientThermalPath` property to `"CellBasedThermalResistance"` inside the parent object of which the `Module` object is a subcomponent.

CH — Coolant thermal port
thermal

Thermal conserving port associated with the coolant thermal path.

Dependencies

To enable this port, set the `CoolantThermalPath` property of the `Module` object to `"CellBasedThermalResistance"`. Alternatively, set the `CoolantThermalPath` property to `"CellBasedThermalResistance"` inside the parent object of which the `Module` object is a subcomponent.

Note If you set the `CoolingPlate` property to either `"Top"` or `"Bottom"` inside the `Module` object or inside the top-level parent object of which the `Module` object is a subcomponent, then this port is not enabled.

CPT — Cooling plate top port
thermal array-of-nodes

Thermal conserving port used to connect the top side of the module with a cooling plate. The size of this port depends on the number of models in the system. If you set the `ModelResolution` property to `“Lumped”`, this port outputs a scalar. If you set the `ModelResolution` property to `“Detailed”`, this port outputs a vector of size equal to the number of cells in the system. If you set the `ModelResolution` property to `“Grouped”`, this port outputs a vector of size depending on the grouping strategy.

Dependencies

To enable this port, set the `CoolingPlate` property of the `Module` object to `"Top"`. Alternatively, set the `CoolingPlate` property to `"Top"` inside the parent object of which the `Module` object is a subcomponent.

CPB — Cooling plate bottom port
thermal array-of-nodes

Thermal conserving port used to connect the bottom side of the module with a cooling plate. The size of this port depends on the number of models in the system. If you set the `ModelResolution` property to `“Lumped”`, this port outputs a scalar. If you set the `ModelResolution` property to `“Detailed”`, this port outputs a vector of size equal to the number of cells in the system. If you set the `ModelResolution` property to `“Grouped”`, this port outputs a vector of size depending on the grouping strategy.

Dependencies

To enable this port, set the `CoolingPlate` property of the `Module` object to `"Bottom"`. Alternatively, set the `CoolingPlate` property to `"Bottom"` inside the parent object of which the `Module` object is a subcomponent.

Parameters

Note The names of these parameters and variables depend on the block you have specified in the CellModelOptions property of the underlying `Cell` object that constitutes this module.

The names of the parameters and variables in this page refer to a `Cell` object with the Battery (Table-Based) block as the cell component model block. If you specify a custom block for the cell component model, the names of the parameters and variables might differ.

The parameters of this block need to be set for the cell-level and depend on how you define the `Module` object that you use to generate this custom model. The block parameters are divided into these sections:

- Main
- Dynamics
- Fade
- Calendar Aging
- Thermal
- Cell Balancing

The visibility of each section also depends on the generating `Module` object.

For a full list of the parameters in the **Main**, **Dynamics**, **Fade**, and **Calender Aging** sections, see the Battery (Table-Based) block.

Thermal

Thermal mass — Thermal mass
`100 J/K` (default) | positive scalar

Energy required to raise the temperature of the thermal port by one degree.

Cell level coolant thermal path resistance — Coolant thermal path resistance at cell level
`1.2 K/W` (default) | positive scalar | row vector of positive scalars

Resistance of the coolant thermal path at cell level. The size of this parameter is equal to the total number of electrical models in the system.

Dependencies

To enable this parameter, set the `CoolantThermalPath` property of the `Module` object to `"CellBasedThermalResistance"`. Alternatively, set the `CoolantThermalPath` property to `"CellBasedThermalResistance"` inside the parent object of which the `Module` object is a subcomponent.

Cell level ambient thermal path resistance — Ambient thermal path resistance at cell level
`25 K/W` (default) | positive scalar | row vector of positive scalars

Resistance of the ambient thermal path at cell level. The size of this parameter is equal to the total number of electrical models in the system.

Dependencies

To enable this parameter, set the `CoolantThermalPath` property of the `Module` object to `"CellBasedThermalResistance"`. Alternatively, set the `CoolantThermalPath` property to `"CellBasedThermalResistance"` inside the parent object of which the `Module` object is a subcomponent.

Inter-cell thermal path resistance — Thermal path resistance between cells
`1 K/W` (default) | scalar | row vector of scalars

Resistance of the thermal path between the cells inside the module. The size of this parameter must be equal to the number of inter-cell thermal connections that this block models.

Dependencies

To enable this parameter, set the `InterCellThermalPath` property of the `Module` object to `"On"`. Alternatively, set the `InterCellThermalPath` property to `"On"` inside the parent object of which the `Module` object is a subcomponent.

Inter-parallel assembly thermal path resistance — Thermal path resistance between parallel assemblies
`1 K/W` (default) | scalar | row vector of scalars

Resistance of the thermal path between the parallel assemblies inside the module. The size of this parameter must be equal to the number of inter-parallel assembly thermal connections that this block models.

Dependencies

To enable this parameter, set the `InterCellThermalPath` property of the `Module` object to `"On"`. Alternatively, set the `InterCellThermalPath` property to `"On"` inside the parent object of which the `Module` object is a subcomponent.

Inter-cell radiation heat transfer area — Transfer area of radiation heat between cells
`1e-3 m^2` (default) | scalar | row vector of scalars

Area of the radiation heat transfer between the cells inside the module. The size of this parameter must be equal to the number of inter-cell thermal connections that this block models.

Dependencies

To enable this parameter, set the `InterCellRadiativeThermalPath` property of the `Module` object to `"On"`. Alternatively, set the `InterCellRadiativeThermalPath` property to `"On"` inside the parent object of which the `Module` object is a subcomponent.

Inter-cell radiation heat transfer coefficient — Coefficient of radiation heat transfer between cells
`1e-6 W/(K^4*m^2)` (default) | scalar | row vector of scalars

Coefficient of the radiation heat transfer between the cells inside the module. The size of this parameter must be equal to the number of inter-cell thermal connections that this block models.

Dependencies

To enable this parameter, set the `InterCellRadiativeThermalPath` property of the `Module` object to `"On"`. Alternatively, set the `InterCellRadiativeThermalPath` property to `"On"` inside the parent object of which the `Module` object is a subcomponent.

Inter-parallel assembly area for radiation heat transfer — Area of radiation heat transfer between parallel assemblies
`1e-3 m^2` (default) | scalar | row vector of scalars

Area of the radiation heat transfer between the parallel assemblies inside the module. The size of this parameter must be equal to the number of inter-parallel assemblies thermal connections that this block models.

Dependencies

To enable this parameter, set the `InterCellRadiativeThermalPath` property of the `Module` object to `"On"`. Alternatively, set the `InterCellRadiativeThermalPath` property to `"On"` inside the parent object of which the `Module` object is a subcomponent.

Inter-parallel assembly coefficient for radiation heat transfer — Coefficient of radiation heat transfer between parallel assemblies
`1e-6 W/(K^4*m^2)` (default) | scalar | row vector of scalars

Coefficient of the radiation heat transfer between the parallel assemblies inside the module. The size of this parameter must be equal to the number of inter-parallel assemblies thermal connections that this block models.

Dependencies

To enable this parameter, set the `InterCellRadiativeThermalPath` property of the `Module` object to `"On"`. Alternatively, set the `InterCellRadiativeThermalPath` property to `"On"` inside the parent object of which the `Module` object is a subcomponent.

Parallel Assembly

Non-cell electrical resistance — Parallel assembly non-cell resistance
`1 / 1000 Ohm` (default) | positive scalar

Non-cell electrical resistance of each parallel assembly in the module.

Dependencies

To enable this parameter, set the `NonCellResistance` property of the `Module` object to `"On"`. Alternatively, set the `NonCellResistance` property to `"On"` inside the parent object of which the `Module` object is a subcomponent.

Module

Non-cell electrical resistance — Module non-cell resistance
`1 / 1000` `Ohm` (default) | positive scalar

Non-cell electrical resistance of the module.

Dependencies

To enable this parameter, set the `NonCellResistance` property of the `Module` object to `"On"`. Alternatively, set the `NonCellResistance` property to `"On"` inside the parent object of which the `Module` object is a subcomponent.

Cell Balancing

Cell balancing switch closed resistance — Closed resistance of cell balancing switch
`0.01` `Ohm` (default) | positive scalar

Closed resistance of the cell balancing switch.

Cell balancing switch open conductance — Open conductance of cell balancing switch
`1e-8` `1/Ohm` (default) | positive scalar

Open conductance of the cell balancing switch.

Cell balancing switch operation threshold — Threshold for cell balancing switch operation
`0.5` (default) | positive scalar

Threshold to activate the cell balancing switch operation.

Cell balancing shunt resistance — Resistance of cell balancing shunt
`50` `Ohm` (default) | positive scalar

Resistance of the cell balancing shunt.

Initial Targets (Variables)

The size of these variables is equal to the total number of electrical models in the system. For example, in a Module block with 2 parallel assemblies and 2 series cells, the size of the variables depends on the `ModelResolution` property of the `Module` object and is equal to `4` for `Detailed`, `1` for `Lumped`, or `3` when the `SeriesGrouping` property is equal to `[1 1]` and the `ParallelGrouping` property is equal to `[1 2]`.

Cell current (positive in) — Cell-level current, iCell
`0` `A` (default) | scalar | vector

Current at the cell level. The size of this variable is equal to the total number of electrical models in the system.

Cell terminal voltage — Cell-level voltages, vCell
`0` V (default) | scalar | vector

Voltages at the cell level. The size of this signal is equal to the total number of electrical models in the system.

Cell state of charge — Cell-level state of charge, socCell
`1` (default) | scalar | vector

State of charge at cell level. The value of this variable typically varies between 0 and 1. The size of this signal is equal to the total number of electrical models in the system.

Cell discharge cycles — Cell-level number of cycles, numCyclesCell
`0` (default) | scalar | vector

Cell-level number of cycles related to battery aging. The size of this signal is equal to the total number of electrical models in the system.

Cell temperature — Cell-level temperature, temperatureCell
`298.15` K (default) | positive scalar

Temperature at the cell level. The size of this signal is equal to the total number of electrical models in the system.

Parallel Assembly Voltage — Parallel assembly-level voltages, vParallelAssembly
`repmat(0,4,1)` V (default) | scalar | vector

Voltages at the parallel assembly level. The size of this signal is equal to the total number of parallel assemblies in the module.

Parallel Assembly state of charge — State of charge at parallel assembly level, socParallelAssembly
`repmat(1,4,1)` (default) | scalar | vector

State of charge at the parallel assembly level. The size of this signal is equal to the total number of parallel assemblies in the module.

Version History

Introduced in R2022b

See Also

Apps
Battery Builder

Simscape Blocks
arrayOfThermalNodesConnector | ParallelAssembly (Generated Block) | ModuleAssembly (Generated Block) | Pack (Generated Block)

Objects
`Module`

Functions
`buildBattery`

Topics
"Manage Battery Run-Time Parameters with Centralized Script"

ModuleAssembly (Generated Block)

Generated Simscape model of battery module assembly

Description

The ModuleAssembly block is a custom generated subsystem model of a battery module assembly. You can create this Simscape subsystem by using the `buildBattery` function with a `ModuleAssembly` object as an input argument. The ModuleAssembly subsystem block is inside the Simulink library that you specify in the `LibraryName` argument of the `buildBattery` function.

Note To allow for structural changes and further customization, when you add the ModuleAssembly (Generated Block) masked library block from the parent library to a Simulink model, the linked block does not contain the link or path to the parent library block. If you modify any of the battery structural properties, such as `NumSeriesAssemblies` or `NumParallelCells`, you must rebuild the original parent library and copy the block in your model again.

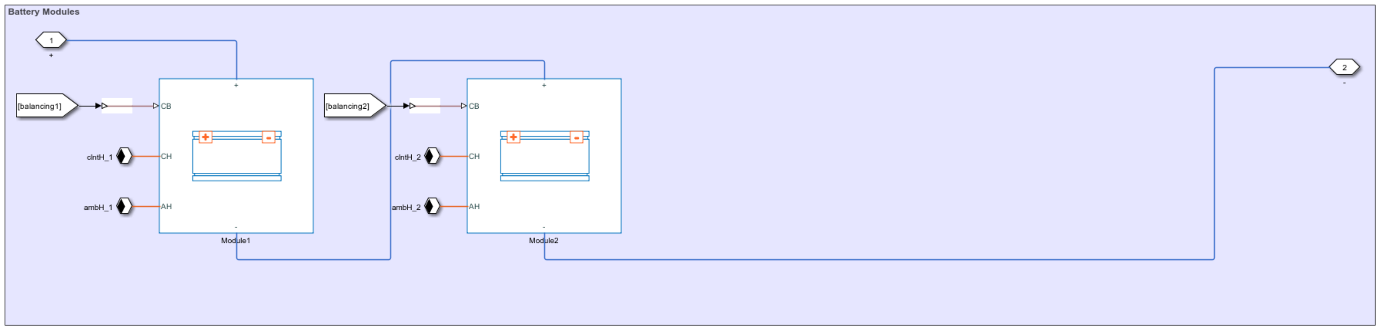
For more information about parent and linked blocks, see “Linked Blocks”.

The ModuleAssembly subsystem model comprises four major sections.

- The Battery Modules section displays all the Module subsystems defined in the `Module` object. This section contains all the series and parallel electrical circuit connections between the modules, as well as the additional connections to the main electrical terminals and the module-to-thermal boundary conditions. If you require additional non-module resistances, then the function adds and connects two additional electrical resistors here.
- The Output signals section groups the output of each Module block into a single multiplexed output signal for the ModuleAssembly subsystem.
- The Thermal Boundary Conditions section contains the thermal boundary conditions for each Module block. The `buildBattery` function adds a connector port from the thermal domain and routes it to every Module subsystem according to the specified thermal boundary conditions. This section is optional and only appears if you define an ambient or coolant thermal path in the `ModuleAssembly` object.
- The Balancing Signals section contains the routing of the cell balancing control signal from the battery management system to every Module subsystem. Selector blocks route the correct elements of the control signal to their corresponding ParallelAssembly blocks inside the modules.
- The Coolant Thermal Liquid section contains the thermal liquid network manifold for the coolant that circulates inside the battery cooling plates. This section is optional and only appears if you define a cooling plate in the `ModuleAssembly` object.

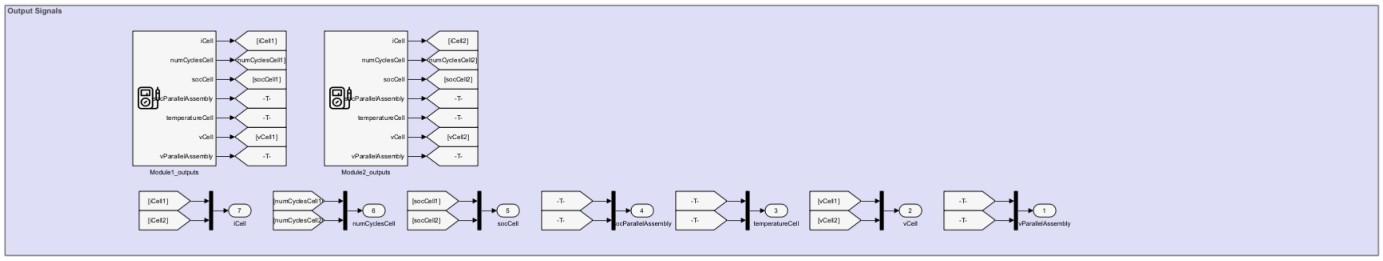
Battery Modules

This figure shows the Battery Modules section for a ModuleAssembly subsystem made of two battery modules.



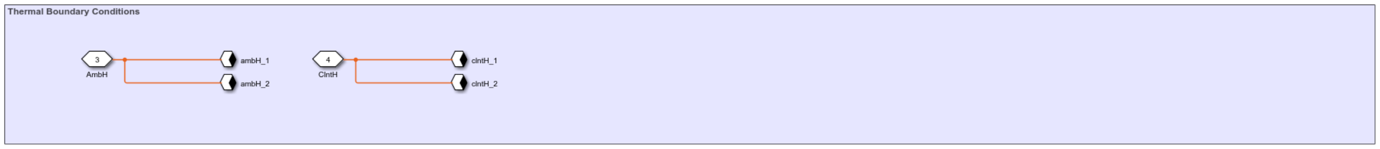
Output Signals

This figure shows the Output Signals section for a ModuleAssembly subsystem made of two battery modules.



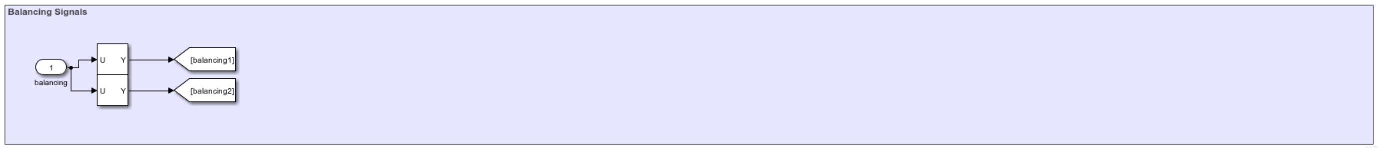
Thermal Boundary Conditions

This figure shows the Thermal Boundary Conditions section for a ModuleAssembly subsystem made of two battery modules.



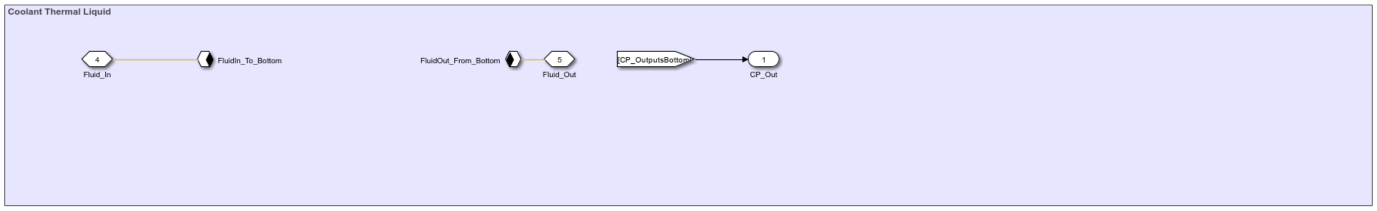
Balancing Signals

This figure shows the Balancing Signals section for a ModuleAssembly subsystem made of two battery modules.



Coolant Thermal Liquid

This figure shows the Coolant Thermal Liquid section for a ModuleAssembly subsystem made of two battery modules.



Ports

Input

balancing — Balancing switch control, unitless
scalar

Switch control port for the balancing strategy, specified as a scalar.

Dependencies

To enable this port, set the `BalancingStrategy` property of the `ModuleAssembly` object to `"Passive"`. Alternatively, set the `BalancingStrategy` property to `"Passive"` inside the parent object of which the `ModuleAssembly` object is a subcomponent.

Output

Note The names of the output ports depend on the block you have specified in the CellModelOptions property of the underlying `Cell` object that constitutes this module assembly.

The names of the output ports in this page refer to a `Cell` object with the Battery (Table-Based) block as the cell component model block. If you specify a custom block for the cell component model, the names of the output ports might differ.

iCell — Cell-level current, Ampere
scalar | vector

Current at the cell level. The original signal units from the Module block are Amperes. The size of this signal is equal to the total number of electrical models in the system.

numCyclesCell — Cell-level number of cycles, unitless
scalar | vector

Cell-level number of cycles related to battery aging. The original signal units from the Module block are unitless. The size of this signal is equal to the total number of electrical models in the system.

socCell — Cell-level state of charge, unitless
scalar | vector

State of charge at cell level. The original signal units from the Module block are unitless and the value typically varies between 0 and 1. The size of this signal is equal to the total number of electrical models in the system.

socParallelAssembly — State of charge at parallel assembly level, unitless
scalar | vector

State of charge at the parallel assembly level. The original signal units from the Module block are unitless and the value typically varies between 0 and 1. The size of this signal is equal to the total number of parallel assemblies in the module assembly.

temperatureCell — Cell-level temperature, kelvin
scalar | vector

Temperature at the cell level. The original signal units from the Module block are Kelvin. The size of this signal is equal to the total number of electrical models in the system.

Dependencies

To enable this port, enable the thermal port of the cells that comprise the pack. To model the thermal port of the cells, in the `CellModelBlock` property of the cells, set the **thermal_port** parameter of the `BlockParameters` property to `"Model"`.

vCell — Cell-level voltages, volts
scalar | vector

Voltages at the cell level. The original signal units from the Module block are volts. The size of this signal is equal to the total number of electrical models in the system.

vParallelAssembly — Parallel assembly-level voltages, volts
scalar | vector

Voltages at the parallel assembly level. The original signal units from the Module block are volts. The size of this signal is equal to the total number of parallel assemblies in the module assembly. Use this signal as an input to the battery management system (BMS) blocks of Simscape Battery.

CP_Out — Outputs of each cooling plate in model
vector of physical signals

Vector that contains the plate temperature, the fluid pressure drop, and the fluid temperature change of each cooling plate in the model. The size of this port depends on the number of cooling plates in the model.

Dependencies

To enable this port:

- The `ModuleAssembly` object must not be a subcomponent of a `Pack` object.
- Set the `CoolingPlate` property of the `ModuleAssembly` object to either `"Top"` or `"Bottom"` .

Conserving

+ — Positive terminal
electrical

Electrical conserving port associated with the positive terminal of the battery module assembly.

- — Negative terminal
electrical

Electrical conserving port associated with the negative terminal of the battery module assembly.

AmbH — Ambient thermal port
thermal

Thermal conserving port associated with the ambient thermal path.

Dependencies

To enable this port, set the `AmbientThermalPath` property of the `ModuleAssembly` object to `"CellBasedThermalResistance"`. Alternatively, set the `AmbientThermalPath` property to `"CellBasedThermalResistance"` inside the top-level parent object of which the `ModuleAssembly` object is a subcomponent.

ClntH — Coolant thermal port
thermal

Thermal conserving port associated with the coolant thermal path.

Dependencies

To enable this port, set the `CoolantThermalPort` property of the `ModuleAssembly` object to `"CellBasedThermalResistance"`. Alternatively, set the `CoolantThermalPath` property to `"CellBasedThermalResistance"` inside the top-level parent object of which the `ModuleAssembly` object is a subcomponent.

Note If you set the `CoolingPlate` property to either `"Top"` or `"Bottom"` inside the `ModuleAssembly` object or inside the top-level parent object of which the `ModuleAssembly` object is a subcomponent, then this port is not enabled.

CPT — Cooling plate top port
thermal array-of-nodes

Thermal conserving port used to connect the top side of the module assembly with a cooling plate. The size of this port depends on the total number of battery models in the system.

Dependencies

To enable this port:

- The `ModuleAssembly` object must be a subcomponent of a `Pack` object.
- Set the `CoolingPlate` property of the `ModuleAssembly` or the parent `Pack` object to `"Top"` .

CPB — Cooling plate bottom port
thermal array-of-nodes

Thermal conserving port used to connect the bottom side of the module assembly with a cooling plate. The size of this port depends on the total number of battery models in the system.

Dependencies

To enable this port:

- The `ModuleAssembly` object must be a subcomponent of a `Pack` object.
- Set the `CoolingPlate` property of the `ModuleAssembly` or the parent `Pack` object to `"Bottom"` .

Fluid_In — Fluid in cooling plate
thermal fluid

Thermal fluid conserving port associated with the fluid that comes in each cooling plate of the model. The size of this port depends on the number of cooling plates in the model.

Dependencies

To enable this port:

- The `ModuleAssembly` object must not be a subcomponent of a `Pack` object.
- Set the `CoolingPlate` property of the `ModuleAssembly` object to either `"Top"` or `"Bottom"` .

Fluid_Out — Fluid out of cooling plate
thermal fluid

Thermal fluid conserving port associated with the fluid that comes out of each cooling plate in the model. The size of this port depends on the number of cooling plates in the model.

Dependencies

To enable this port:

- The `ModuleAssembly` object must not be a subcomponent of a `Pack` object.
- Set the `CoolingPlate` property of the `ModuleAssembly` object to either `"Top"` or `"Bottom"` .

Version History

Introduced in R2022b

See Also

Apps
Battery Builder

Simscape Blocks
arrayOfThermalNodesConnector | ParallelAssembly (Generated Block) | Module (Generated Block) | Pack (Generated Block)

Objects
`ModuleAssembly`

Functions
`buildBattery`

Topics
“Manage Battery Run-Time Parameters with Centralized Script”

ParallelAssembly

Generated Simscape model of battery parallel assembly

Description

The ParallelAssembly block represents a custom generated system model of a battery parallel assembly. You can create this Simscape block by using the `buildBattery` function with a `ParallelAssembly` object as an input argument. The ParallelAssembly generated block is inside the Simulink library that you specify in the `LibraryName` argument of the `buildBattery` function.

Note To allow for structural changes and further customization, when you add the ParallelAssembly masked library block from the parent library to a Simulink model, the linked block does not contain the link or path to the parent library block. If you modify any of the battery structural properties, such as `NumSeriesAssemblies` or `NumParallelCells`, you must rebuild the original parent library and copy the block in your model again.

For more information about parent and linked blocks, see “Linked Blocks”.

Variables

To set the priority and initial target values for the block variables prior to simulation, use the **Initial Targets** section in the block dialog box or Property Inspector. For more information, see “Set Priority and Initial Target for Block Variables”.

For a detailed list of the variables in the ParallelAssembly block, see the “Initial Targets (Variables)” on page 1-112 section.

Nominal values provide a way to specify the expected magnitude of a variable in a model. Using system scaling based on nominal values increases the simulation robustness. Nominal values can come from different sources, one of which is the **Nominal Values** section in the block dialog box or Property Inspector. For more information, see “System Scaling by Nominal Values”.

Ports

Input

CB — Cell balancing, unitless
physical signal

Cell balancing port, specified as a physical signal.

Dependencies

To enable this port, set the `BalancingStrategy` property of the `ParallelAssembly` object to `"Passive"`. Alternatively, set the `BalancingStrategy` property to `"Passive"` inside the parent object of which the `ParallelAssembly` object is a subcomponent.

Conserving

+ — Positive terminal
electrical

Electrical conserving port associated with the positive terminal of the battery parallel assembly.

- — Negative terminal
electrical

Electrical conserving port associated with the negative terminal of the battery parallel assembly.

AH — Ambient thermal port
thermal

Thermal conserving port associated with the ambient thermal path.

Dependencies

To enable this port, set the `AmbientThermalPath` property of the `ParallelAssembly` object to `"CellBasedThermalResistance"`. Alternatively, set the `AmbientThermalPath` property to `"CellBasedThermalResistance"` inside the parent object of which the `ParallelAssembly` object is a subcomponent.

CH — Coolant thermal port
thermal

Thermal conserving port associated with the coolant thermal path.

Dependencies

To enable this port, set the `CoolantThermalPath` property of the `ParallelAssembly` object to `"CellBasedThermalResistance"`. Alternatively, set the `CoolantThermalPath` property to `"CellBasedThermalResistance"` inside the parent object of which the `ParallelAssembly` object is a subcomponent.

Note If you set the `CoolingPlate` property to either `"Top"` or `"Bottom"` inside the `ParallelAssembly` object or inside the top-level parent object of which the `ParallelAssembly` object is a subcomponent, then this port is not enabled.

ICH — Inter-cell thermal port
thermal

Thermal conserving port associated with the inter-cell thermal path.

Dependencies

To enable this port, set either the `InterCellThermalPath` or the `InterCellRadiativeThermalPath` properties of the `ParallelAssembly` object to `"on"`.

CPT — Cooling plate top port
thermal array-of-nodes

Thermal conserving port used to connect the top side of the module with a cooling plate. The size of this port depends on the number of models in the system. If you set the `ModelResolution` property to `“Lumped”`, this port outputs a scalar. If you set the `ModelResolution` property to `“Detailed”`, this port outputs a vector of size equal to the number of cells in the system. If you set the `ModelResolution` property to `“Grouped”`, this port outputs a vector of size depending on the grouping strategy.

Dependencies

To enable this port, set the `CoolingPlate` property of the `ParallelAssembly` object to `"Top"`. Alternatively, set the `CoolingPlate` property to `"Top"` inside the parent object of which the `ParallelAssembly` object is a subcomponent.

CPB — Cooling plate bottom port
thermal array-of-nodes

Thermal conserving port used to connect the bottom side of the module with a cooling plate. The size of this port depends on the number of models in the system. If you set the `ModelResolution` property to `“Lumped”`, this port outputs a scalar. If you set the `ModelResolution` property to `“Detailed”`, this port outputs a vector of size equal to the number of cells in the system. If you set the `ModelResolution` property to `“Grouped”`, this port outputs a vector of size depending on the grouping strategy.

Dependencies

To enable this port, set the `CoolingPlate` property of the `ParallelAssembly` object to `"Bottom"`. Alternatively, set the `CoolingPlate` property to `"Bottom"` inside the parent object of which the `ParallelAssembly` object is a subcomponent.

Parameters

Note The names of these parameters and variables depend on the block you have specified in the CellModelOptions property of the underlying `Cell` object that constitutes this parallel assembly.

The names of the parameters and variables in this page refer to a `Cell` object with the Battery (Table-Based) block as the cell component model block. If you specify a custom block for the cell component model, the names of the parameters and variables might differ.

The parameters of this block need to be set for the cell-level and depend on how you define the `ParallelAssembly` object that you use to generate this custom model. The block parameters are divided into these sections:

- Main
- Dynamics
- Fade
- Calendar Aging
- Thermal
- Cell Balancing

The visibility of each section also depends on the generating `ParallelAssembly` object.

For a full list of the parameters in the **Main**, **Dynamics**, **Fade**, and **Calender Aging** sections, see the Battery (Table-Based) block.

Thermal

Thermal mass — Thermal mass
`100 J/K` (default) | positive scalar

Energy required to raise the temperature of the thermal port by one degree.

Cell level coolant thermal path resistance — Coolant thermal path resistance at cell level
`1.2 K/W` (default) | positive scalar | row vector of positive scalars

Resistance of the coolant thermal path at the cell level.

Dependencies

To enable this parameter, set the `CoolantThermalPath` property of the `ParallelAssembly` object to `"CellBasedThermalResistance"`. Alternatively, set the `CoolantThermalPath` property to `"CellBasedThermalResistance"` inside the parent object of which the `ParallelAssembly` object is a subcomponent.

Cell level ambient thermal path resistance — Ambient thermal path resistance at cell level
`25 K/W` (default) | positive scalar | row vector of positive scalars

Resistance of the ambient thermal path at the cell level. The size of this parameter is equal to the total number of electrical models in the system.

Dependencies

To enable this parameter, set the `CoolantThermalPath` property of the `ParallelAssembly` object to `"CellBasedThermalResistance"`. Alternatively, set the `CoolantThermalPath` property to `"CellBasedThermalResistance"` inside the parent object of which the `ParallelAssembly` object is a subcomponent.

Inter-cell thermal path resistance — Thermal path resistance between cells
`1 K/W` (default) | scalar | row vector of scalars

Resistance of the thermal path between the cells inside the parallel assembly. The size of this parameter must be equal to the number of inter-cell thermal connections that this block models.

Dependencies

To enable this parameter, set the `InterCellThermalPath` property of the `ParallelAssembly` object to `"On"`. Alternatively, set the `InterCellThermalPath` property to `"On"` inside the parent object of which the `ParallelAssembly` object is a subcomponent.

Inter-cell radiation heat transfer area — Area of radiation heat transfer between cells
`1e-3 m^2` (default) | scalar | row vector of scalars

Area of the radiation heat transfer between the cells inside the parallel assembly. The size of this parameter must be equal to the number of inter-cell thermal connections that this block models.

Dependencies

To enable this parameter, set the `InterCellRadiativeThermalPath` property of the `ParallelAssembly` object to `"On"`. Alternatively, set the `InterCellRadiativeThermalPath` property to `"On"` inside the parent object of which the `ParallelAssembly` object is a subcomponent.

Inter-cell radiation heat transfer coefficient — Coefficient of radiation heat transfer between cells
1e-6 W/(K^4*m^2) (default) | scalar | row vector of scalars

Coefficient of the radiation heat transfer between the cells inside the parallel assembly. The size of this parameter must be equal to the number of inter-cell thermal connections that this block models.

Dependencies

To enable this parameter, set the InterCellRadiativeThermalPath property of the ParallelAssembly object to "On". Alternatively, set the InterCellRadiativeThermalPath property to "On" inside the parent object of which the ParallelAssembly object is a subcomponent.

Parallel Assembly

Non-cell electrical resistance — Parallel assembly non-cell resistance
1 / 1000 Ohm (default) | positive scalar

Non-cell electrical resistance of each parallel assembly in the module.

Dependencies

To enable this parameter, set the NonCellResistance property of the ParallelAssembly object to "On". Alternatively, set the NonCellResistance property to "On" inside the parent object of which the ParallelAssembly object is a subcomponent.

Cell Balancing

Cell balancing switch closed resistance — Closed resistance of cell balancing switch
0.01 Ohm (default) | positive scalar

Closed resistance of the cell balancing switch.

Cell balancing switch open conductance — Open conductance of cell balancing switch
1e-8 1/Ohm (default) | positive scalar

Open conductance of the cell balancing switch.

Cell balancing switch operation threshold — Threshold for cell balancing switch operation
0.5 (default) | positive scalar

Threshold to activate the cell balancing switch operation.

Cell balancing shunt resistance — Resistance of cell balancing shunt
50 Ohm (default) | positive scalar

Resistance of the cell balancing shunt.

Initial Targets (Variables)

The size of these variables is equal to the total number of electrical models in the system. For example, in a Module block with 2 parallel assemblies and 2 series cells, the size of the variables depends on the ModelResolution property of the Module object and is equal to 4 for Detailed, 1 for Lumped, or 3 when the SeriesGrouping property is equal to [1 1] and the ParallelGrouping property is equal to [1 2].

Cell current (positive in) — Cell-level current, iCell
`0` A (default) | scalar | vector

Current at the cell level. The size of this variable is equal to the total number of electrical models in the system.

Cell terminal voltage — Cell-level voltages, vCell
`0` V (default) | scalar | vector

Voltages at the cell level. The size of this signal is equal to the total number of electrical models in the system.

Cell state of charge — Cell-level state of charge, socCell
`1` (default) | scalar | vector

State of charge at cell level. The value of this variable typically varies between 0 and 1. The size of this signal is equal to the total number of electrical models in the system.

Cell discharge cycles — Cell-level number of cycles, numCyclesCell
`0` (default) | scalar | vector

Cell-level number of cycles related to battery aging. The size of this signal is equal to the total number of electrical models in the system.

Cell temperature — Cell-level temperature, temperatureCell
`298.15` K (default) | positive scalar

Temperature at the cell level. The size of this signal is equal to the total number of electrical models in the system.

Parallel Assembly Voltage — Parallel assembly-level voltages, vParallelAssembly
`0` V (default) | scalar | vector

Voltages at the parallel assembly level. The size of this signal is equal to the total number of electrical models in the system.

Parallel Assembly state of charge — State of charge at parallel assembly level, socParallelAssembly
`1` (default) | scalar | vector

State of charge at the parallel assembly level. The size of this signal is equal to the total number of electrical models in the system.

Version History
Introduced in R2022b

See Also

Apps
Battery Builder

Simscape Blocks
arrayOfThermalNodesConnector | Module (Generated Block) | ModuleAssembly (Generated Block) | Pack (Generated Block)

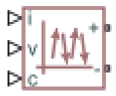
Objects
`ParallelAssembly`

Functions
`buildBattery`

Topics
"Manage Battery Run-Time Parameters with Centralized Script"

Cycler

Ideal galvanostat, potentiostat, or cycler



Libraries:
Simscape / Battery / Cyclers

Description

The Cycler block implements an ideal galvanostat, potentiostat, or a cycler.

To configure this block as a galvanostat, set the **Configuration** parameter to `Galvanostat, specify current`. The Cycler block then represents an ideal current source or sink that is powerful enough to maintain the specified current through it regardless of the voltage across it. The output current is the value at the physical signal port, **i**.

To configure this block as a potentiostat, set the **Configuration** parameter to `Potentiostat, specify voltage`. The Cycler block then represents an ideal voltage source that is powerful enough to maintain the specified voltage at its output regardless of the current passing through it. The output voltage is the value at the physical signal port, **v**.

To configure this block as a cycler, set the **Configuration** parameter to `Cycler, specify current or voltage`. When the value at the control physical signal port, **c**, is equal to `0`, the cycler turns off. When the value at the control physical signal port, **c**, is equal to `1`, this block represents a galvanostat. When the value at the control physical signal port, **c**, is equal to `2`, this block represents a potentiostat.

This table shows the overall behavior of the Cycler block:

Configuration				
`Galvanostat, specify current`	`Potentiostat, specify voltage`	`Cycler, specify current or voltage`		
		Value at input port **c**		
		0	1	2
The Cycler block represents an ideal current source.	The Cycler block represents an ideal voltage source.	The Cycler block turns off.	The Cycler block represents a galvanostat.	The Cycler block represents a potentiostat.

Ports

Input

i — Current, A
physical signal

Physical signal input port associated with the current that passes through the cycler.

Dependencies

To enable this port, set the **Configuration** parameter to either `Galvanostat, specify current` or `Cycler, specify current or voltage`.

v — Voltage, V
physical signal

Physical signal input port associated with the voltage across the cycler.

Dependencies

To enable this port, set the **Configuration** parameter to either `Potentiostat, specify voltage` or `Cycler, specify current or voltage`.

c — Control, unitless
physical signal

Physical signal input port that controls the behavior of the cycler.

Dependencies

To enable this port, set the **Configuration** parameter to `Cycler, specify current or voltage`.

Conserving

+ — Positive terminal
electrical

Electrical conserving port associated with the positive terminal of the cycler.

- — Negative terminal
electrical

Electrical conserving port associated with the negative terminal of the cycler.

Parameters

Current sign convention — Sign convention option for current
`Positive for device discharge (IEC 62660)` (default) | `Positive for device charge`

Convention option for the sign of the current.

Configuration — Block configuration
`Cycler, specify current or voltage` (default) | `Galvanostat, specify current` | `Potentiostat, specify voltage`

Option to configure the behavior of the block.

Version History

Introduced in R2022b

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using Simulink® Coder™.

See Also

Charger | Discharger

Discharger

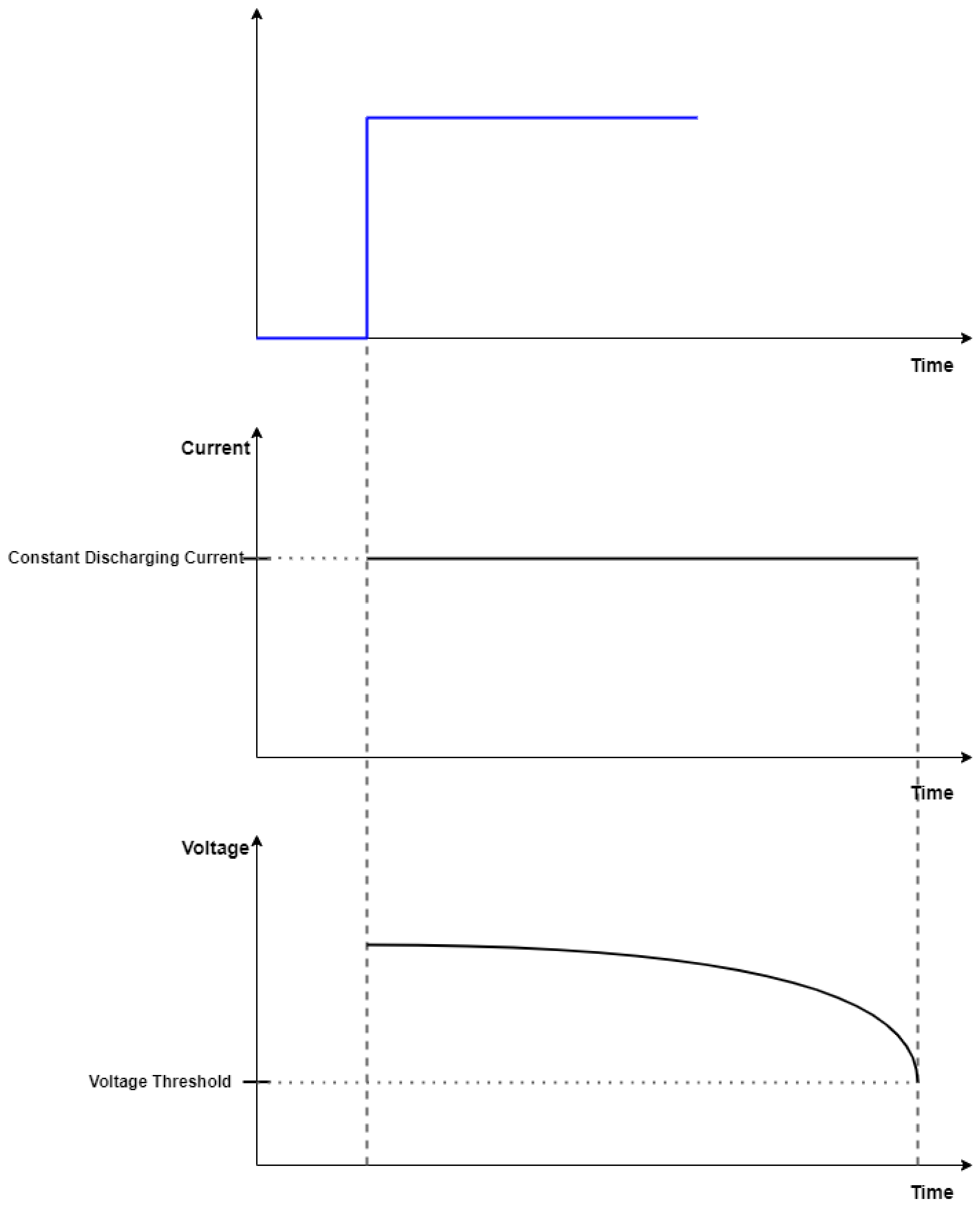
Ideal battery discharger

Libraries:
Simscape / Battery / Cyclers

Description

This block implements an ideal discharger. When the signal at the input port, **t**, changes from a value below `0.5` to a value equal to or greater than `0.5`, the block performs constant current discharging at the value of the **Constant discharging current** parameter until the voltage is less than or equal to the value of the **Voltage threshold** parameter.

This figure shows the behavior of the Discharger block when the **t** input crosses `0.5`:



Ports

Input

t — Trigger, unitless
physical signal

Physical signal input port that controls the discharging behavior of the block.

Conserving

+ — Positive terminal
electrical

Electrical conserving port associated with the positive terminal of the discharger.

- — Negative terminal
electrical

Electrical conserving port associated with the negative terminal of the discharger.

Parameters

Constant discharging current — Constant discharging current
`5 A` (default)

Value of the constant discharging current at which the block discharges a battery.

Voltage threshold — Voltage threshold for constant discharging current
`10 V` (default)

Threshold at which the block stops performing constant current discharging on the battery.

Version History
Introduced in R2022b

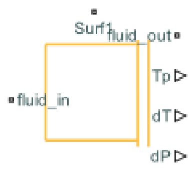
Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using Simulink® Coder™.

See Also
Charger | Cycler

Edge Cooling

Cooling plate with edge cooling



Libraries:
Simscape / Battery / Thermal

Description

This block models a battery cooling plate with edge cooling. Use the `buildBattery` function to create a Simscape model of a battery and connect it to the one or both sides of the cooling plate. You can cool any of the four edges of the cooling plate. The **fluid_in** and **fluid_out** ports are thermal fluid ports. The output ports are the plate temperature, **Tp**, the fluid pressure drop, **dP**, and the fluid temperature change, **dT**.

Ports

Output

Tp — Plate temperature
physical signal

Temperature of the cooling plate.

dT — Fluid temperature change
physical signal

Temperature change of the fluid.

dP — Fluid pressure drop
physical signal

Pressure drop of the fluid.

Conserving

fluid_in — Fluid in cooling plate
thermal fluid

Thermal fluid conserving port associated with the fluid that comes in the cooling plate.

fluid_out — Fluid out of cooling plate
thermal fluid

Thermal fluid conserving port associated with the fluid that comes out of the cooling plate.

Surf1 — Surface 1 of cooling plate
thermal

Thermal conserving port associated with the surface 1 of the cooling plate. This port connects the array of thermal nodes between the cooling plate and battery pack or module.

Surf2 — Surface 2 of cooling plate
thermal

Thermal conserving port associated with the surface 2 of the cooling plate. This port connects the array of thermal nodes between the cooling plate and battery pack or module.

Parameters

Interface

Battery Connectivity — Connectivity of battery
`Single sided` (default) | `Double sided`

Option to choose the connectivity of the battery.

Number of battery thermal nodes (surface 1) — Number of battery thermal nodes on surface 1
`1` (default)

Number of battery thermal nodes on surface 1 of the cooling plate.

Dimension of battery thermal nodes (surface 1) — Dimension of battery thermal nodes on surface 1
`ones(1, 2)` (default)

Dimension of battery thermal nodes on surface 1 of the cooling plate.

Coordinates of battery thermal nodes (surface 1) — Coordinates of battery thermal nodes on surface 1
`ones(1, 2)` (default)

Coordinates of battery thermal nodes on surface 1 of the cooling plate.

Number of battery thermal nodes (surface 2) — Number of battery thermal nodes on surface 2
`1` (default)

Number of battery thermal nodes on surface 2 of the cooling plate.

Dependencies

To enable this parameter, set **Battery Connectivity** to `Double sided`.

Dimension of battery thermal nodes (surface 2) — Dimension of battery thermal nodes on surface 2
`ones(1, 2)` (default)

Dimension of battery thermal nodes on surface 2 of the cooling plate.

Dependencies

To enable this parameter, set **Battery Connectivity** to `Double sided`.

Coordinates of battery thermal nodes (surface 2) — Coordinates of battery thermal nodes on surface 2
`ones(1, 2)` (default)

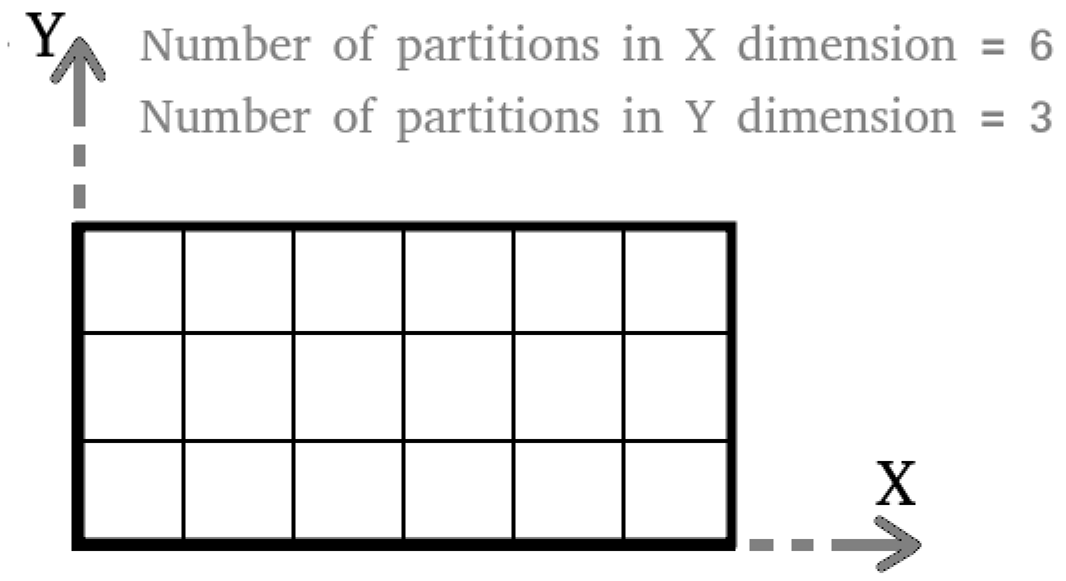
Coordinates of battery thermal nodes on surface 2 of the cooling plate.

Dependencies

To enable this parameter, set **Battery Connectivity** to `Double sided`.

Number of partitions in X dimension for the cooling plate — Number of partitions in X dimension for the cooling plate
2 (default)

Number of partitions in X dimension for the cooling plate.


Y
Number of partitions in X dimension = 6
Number of partitions in Y dimension = 3
X

Number of partitions in Y dimension for the cooling plate — Number of partitions in Y dimension for the cooling plate
5 (default)

Number of partitions in Y dimension for the cooling plate.

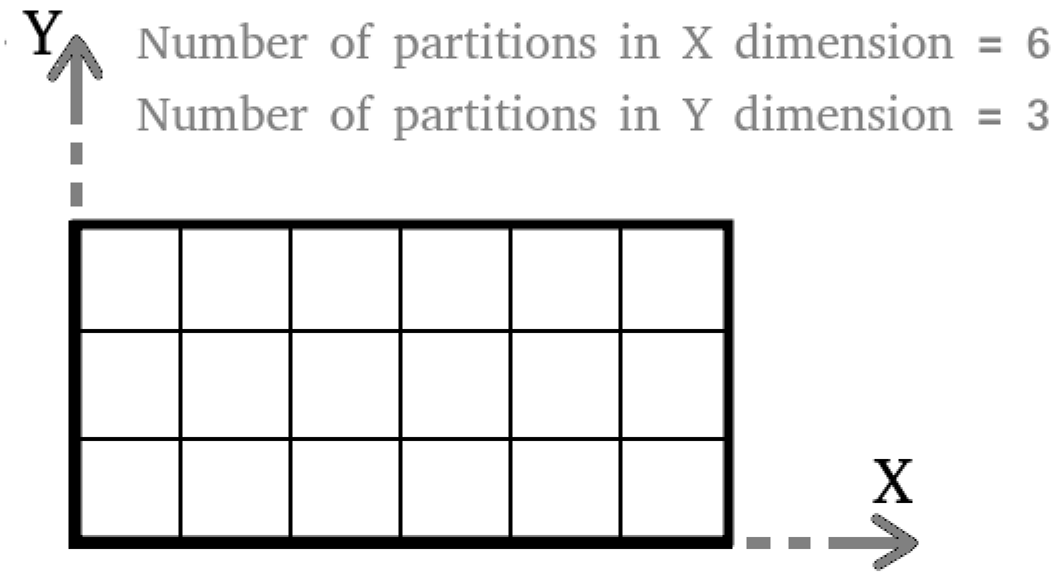

Y
Number of partitions in X dimension = 6
Number of partitions in Y dimension = 3
X

Plate Material

Thickness of cooling plate material — Thickness of material of cooling plate
`2e-3 m` (default)

Thickness of the material of the cooling plate.

Thermal conductivity of cooling plate material — Thermal conductivity of material of cooling plate
`20 W/(K*m)` (default)

Thermal conductivity of the material of the cooling plate.

Density of cooling plate material — Density of material of cooling plate
`2500 kg/m^3` (default)

Density of the material of the cooling plate.

Specific heat of cooling plate material — Specific heat of material of cooling plate
`447 J/(K*kg` (default)

Specific heat of the material of the cooling plate.

Initial temperature of the cooling plate and the coolant fluid — Initial temperature of cooling plate and coolant fluid
`300 K` (default)

Initial temperature of the cooling plate and of the coolant fluid.

Fluid Properties

Reynolds number upper limit for laminar flow — Upper limit of Reynolds number for laminar flow
`2000` (default)

Upper limit of Reynolds number for laminar flow.

Reynolds number lower limit for turbulent flow — Lower limit of Reynolds number for turbulent flow
`4000` (default)

Lower limit of Reynolds number for turbulent flow.

Nusselt number for laminar flow heat transfer — Nusselt number for laminar flow heat transfer
`3.66` (default)

Nusselt number for laminar flow heat transfer.

Aggregate equivalent length of local resistances — Aggregate equivalent length of local resistances
`1e-3 m` (default)

Aggregate equivalent length of local resistances.

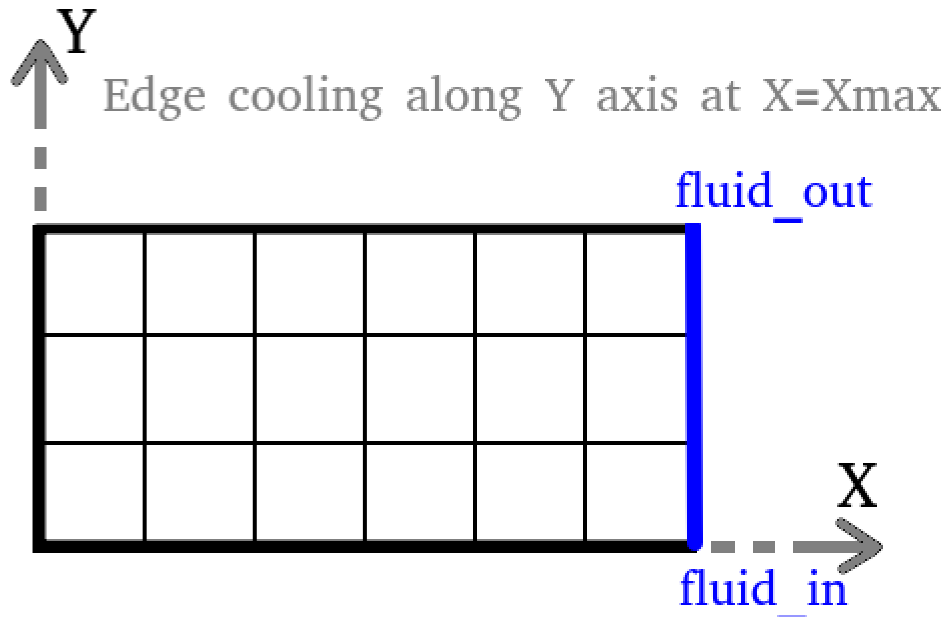
Laminar friction constant for Darcy friction factor — Laminar friction constant for Darcy friction factor
`64` (default)

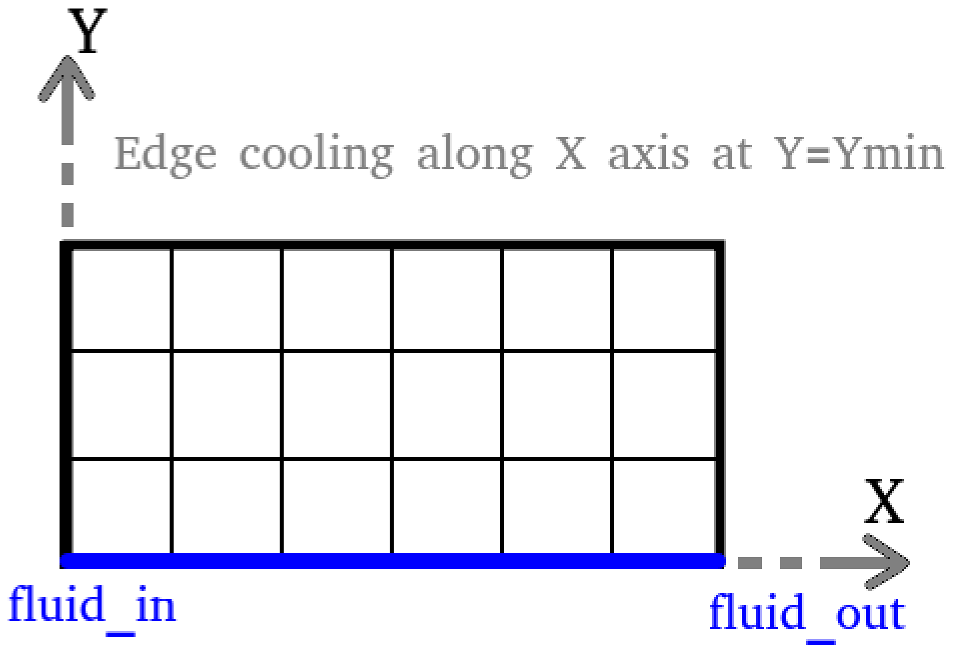
Laminar friction constant for the Darcy friction factor.

Design

Select edge to be cooled — Cooling edge
`Edge cooling along Y axis at X = Xmin` (default) | `Edge cooling along X axis at Y = Ymin` | `Edge cooling along Y axis at X = Xmax` | `Edge cooling along X axis at Y = Ymax`

Option to choose which edge of the cooling plate you want to cool.





Cooling channel hydraulic diameter — Hydraulic diameter of cooling channel
`5e-3 m` (default)

Hydraulic diameter of the cooling channel.

Cooling channel cross-sectional area — Cross-sectional area of cooling channel
`2e-5 m^2` (default)

Cross-sectional area of the cooling channel.

Coolant channel roughness — Roughness of the coolant channel
`1.5e-05` `m` (default)

Roughness of the coolant channel.

Version History

Introduced in R2022b

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using Simulink® Coder™.

See Also

Simscape Blocks
arrayOfThermalNodesConnector | Parallel Channels | U-shaped Channels

Objects
`Cell` | `ParallelAssembly` | `Module` | `ModuleAssembly` | `Pack`

Functions
`buildBattery`

Topics
"Connect Cooling Plate to Battery Blocks"

Fault Qualification

Fault qualification algorithm

Libraries:
Simscape / Battery / BMS / Protection

Description

This block implements a fault qualification algorithm. The input and output ports are logic signals. To prevent error disqualification, set the **Disqualification time** parameter to `0`.

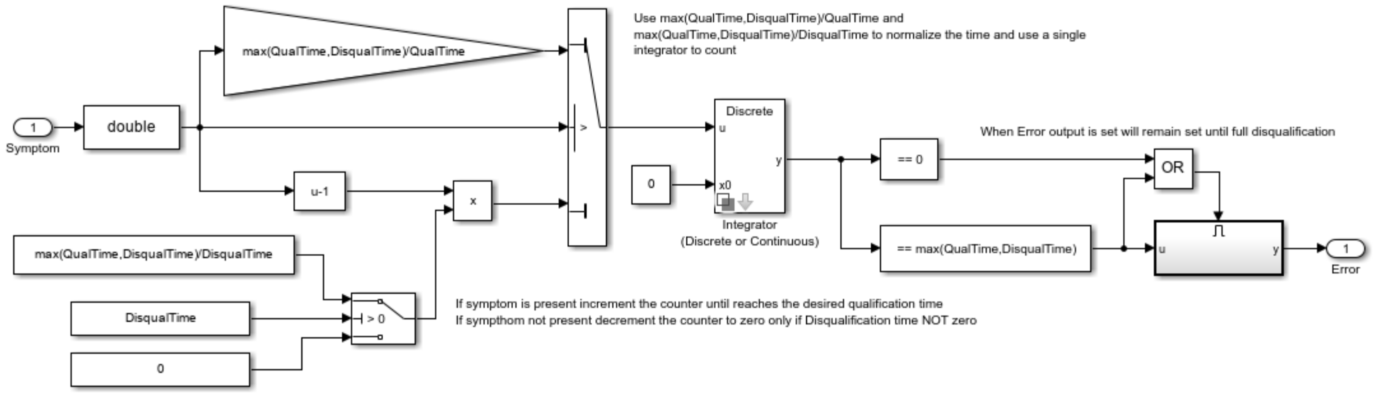
This block supports both single-precision and double-precision floating-point simulation.

Note To enable single-precision floating-point simulation, the data type of all inputs and parameters, except for the **Sample time (-1 for inherited)** parameter, must be `single`.

You can switch between continuous and discrete implementations of the block by using the **Sample time (-1 for inherited)** parameter. To configure the block for continuous time, set the **Sample time (-1 for inherited)** parameter to `0`. To configure the block for discrete time, set the **Sample time (-1 for inherited)** parameter to a positive, nonzero value, or to `-1` to inherit the sample time from an upstream block.

Note Continuous-time implementation of this block works only in double-precision floating-point simulation. If you provide single-precision floating-point parameters and inputs, this block casts them to double-precision floating-point values to prevent errors.

This diagram shows the structure of the block:


1
Symptom
double
max(QualTime,DisqualTime)/QualTime
u-1
x
>
max(QualTime,DisqualTime)/DisqualTime
DisqualTime
> 0
0
Discrete
u
y
x0
Integrator
(Discrete or Continuous)
0
== 0
== max(QualTime,DisqualTime)
OR
u
y
1
Error

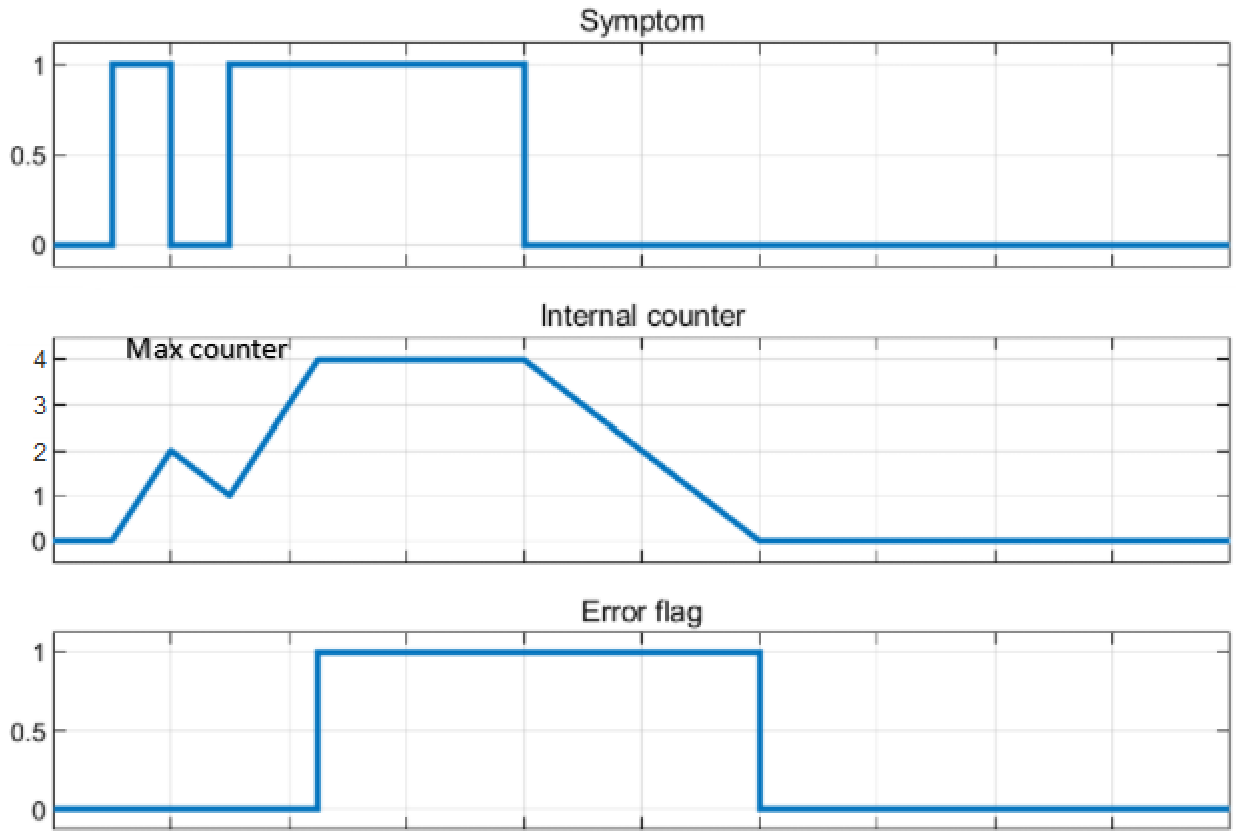
Use max(QualTime,DisqualTime)/QualTime and max(QualTime,DisqualTime)/DisqualTime to normalize the time and use a single integrator to count

When Error output is set will remain set until full disqualification

If symptom is present increment the counter until reaches the desired qualification time
If sympthom not present decrement the counter to zero only if Disqualification time NOT zero

Equations

This figure shows how the internal counter and the **Error** output port of this block work:



The fault qualification takes a symptom as input. The input is `1` when a symptom is present, such as an overvoltage error, and `0` when the symptom is not present. When the symptom is present, the block increments an internal counter. When the counter reaches the maximum value corresponding to the desired qualification time, the output port is `1`.

You must provide a disqualification time greater than `0` to allow disqualification. When the disqualification time is not `0` and the symptom is not present, the block decrements the counter until it reaches `0`. When the counter is `0`, the output port is `0`.

Ports

Input

Symptom — Symptom flag
`0` | `1`

Indication that the battery has a symptom to qualify. If this input is equal to `1`, the battery has a symptom.

Output

Error — Error flag
0 | 1

Indication that the battery has an error. If this output is equal to 1, the battery has an error.

Parameters

Qualification time (s) — Error qualification time
2 (default) | positive scalar

Time required to qualify the error, in seconds.

Disqualification time (s) — Error disqualification time
4 (default) | nonnegative scalar

Time required to disqualify the error, in seconds. If you set this parameter to 0, the block does not disqualify the error.

Sample time (-1 for inherited) — Block sample time
-1 (default) | 0 | positive integer

Time between consecutive block executions. During execution, the block produces outputs and, if appropriate, updates its internal state. For more information, see “What Is Sample Time?” and “Specify Sample Time”.

For inherited discrete-time operation, specify this parameter as -1. For discrete-time operation, specify this parameter as a positive integer. For continuous-time operation, specify this parameter as 0.

If this block is in a masked subsystem or a variant subsystem that allows you to switch between continuous operation and discrete operation, promote the sample time parameter. Promoting the sample time parameter ensures correct switching between the continuous and discrete implementations of the block. For more information, see “Promote Block Parameters on a Mask”.

Version History

Introduced in R2022b

Extended Capabilities

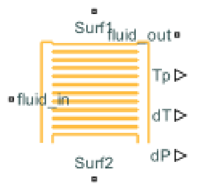
C/C++ Code Generation
Generate C and C++ code using Simulink® Coder™.

See Also

Battery Current Monitoring | Battery Temperature Monitoring | Battery Voltage Monitoring

Parallel Channels

Cooling plate with multiple parallel channels



Libraries:
Simscape / Battery / Thermal

Description

This block models a battery cooling plate with multiple channels and a pair of distributor channels for inlet and outlet flow. Use the `buildBattery` function to create a Simscape model of a battery and connect it to one or both sides of the cooling plate. You can cool any of the four edges of the cooling plate. The **fluid_in** and **fluid_out** ports are thermal fluid ports. The output ports are the plate temperature, **Tp**, the fluid pressure drop, **dP**, and the fluid temperature change, **dT**.

Ports

Output

Tp — Plate temperature
physical signal

Temperature of the cooling plate.

dT — Fluid temperature change
physical signal

Temperature change of the fluid.

dP — Fluid pressure drop
physical signal

Pressure drop of the fluid.

Conserving

fluid_in — Fluid in cooling plate
thermal fluid

Thermal fluid conserving port associated with the fluid that comes in the cooling plate.

fluid_out — Fluid out of cooling plate
thermal fluid

Thermal fluid conserving port associated with the fluid that comes out of the cooling plate.

Surf1 — Surface 1 of cooling plate
thermal

Thermal conserving port associated with the surface 1 of the cooling plate. This port connects the array of thermal nodes between the cooling plate and battery pack or module.

Surf2 — Surface 2 of cooling plate
thermal

Thermal conserving port associated with the surface 2 of the cooling plate. This port connects the array of thermal nodes between the cooling plate and battery pack or module.

Parameters

Interface

Battery Connectivity — Connectivity of battery
`Single sided` (default) | `Double sided`

Option to choose the connectivity of the battery.

Number of battery thermal nodes (surface 1) — Number of battery thermal nodes on surface 1
`1` (default)

Number of battery thermal nodes on surface 1 of the cooling plate.

Dimension of battery thermal nodes (surface 1) — Dimension of battery thermal nodes on surface 1
`ones(1, 2)` (default)

Dimension of battery thermal nodes on surface 1 of the cooling plate.

Coordinates of battery thermal nodes (surface 1) — Coordinates of battery thermal nodes on surface 1
`ones(1, 2)` (default)

Coordinates of battery thermal nodes on surface 1 of the cooling plate.

Number of battery thermal nodes (surface 2) — Number of battery thermal nodes on surface 2
`1` (default)

Number of battery thermal nodes on surface 2 of the cooling plate.

Dependencies

To enable this parameter, set **Battery Connectivity** to `Double sided`.

Dimension of battery thermal nodes (surface 2) — Dimension of battery thermal nodes on surface 2
`ones(1, 2)` (default)

Dimension of battery thermal nodes on surface 2 of the cooling plate.

Dependencies

To enable this parameter, set **Battery Connectivity** to `Double sided`.

Coordinates of battery thermal nodes (surface 2) — Coordinates of battery thermal nodes on surface 2
`ones(1, 2)` (default)

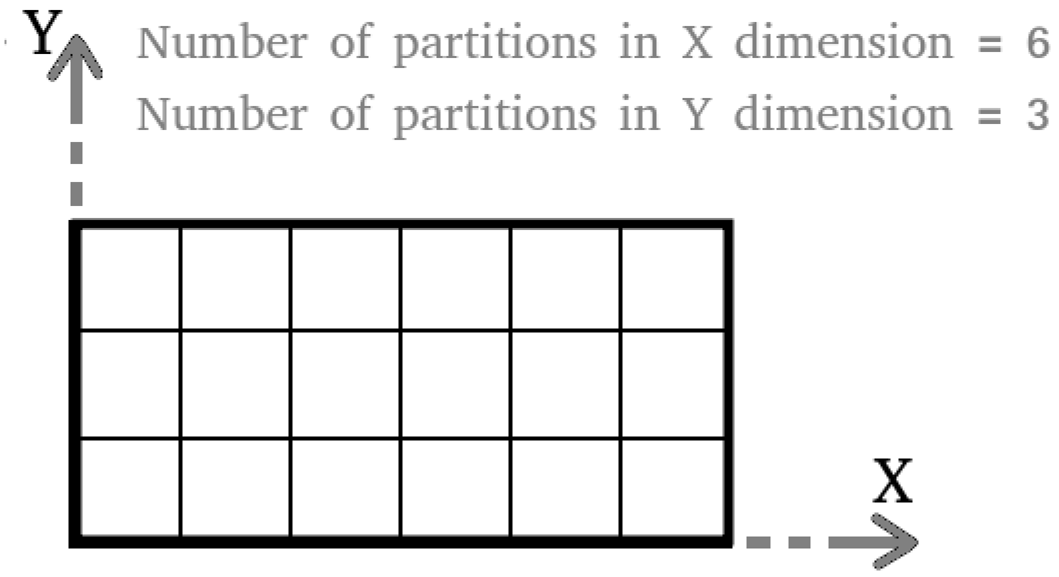
Coordinates of battery thermal nodes on surface 2 of the cooling plate.

Dependencies

To enable this parameter, set **Battery Connectivity** to `Double sided`.

Number of partitions in X dimension for the cooling plate — Number of partitions in X dimension for the cooling plate
2 (default)

Number of partitions in X dimension for the cooling plate.


Y
Number of partitions in X dimension = 6
Number of partitions in Y dimension = 3
X

Number of partitions in Y dimension for the cooling plate — Number of partitions in Y dimension for the cooling plate
5 (default)

Number of partitions in Y dimension for the cooling plate.

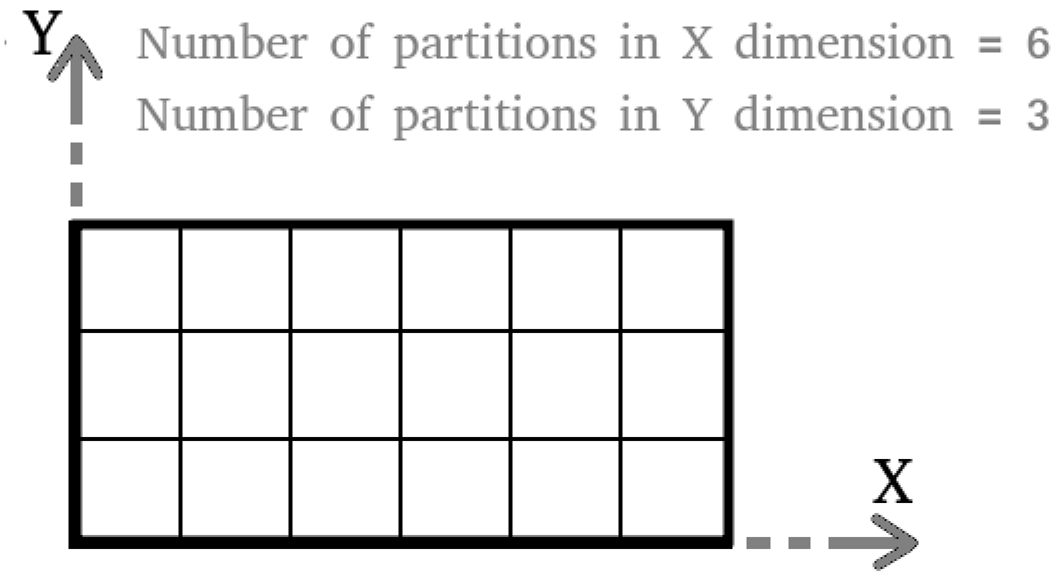

Y
Number of partitions in X dimension = 6
Number of partitions in Y dimension = 3
X

Plate Material

Thickness of cooling plate material — Thickness of material of cooling plate
`2e-3` `m` (default)

Thickness of the material of the cooling plate.

Thermal conductivity of cooling plate material — Thermal conductivity of material of cooling plate
`20` `W/(K*m)` (default)

Thermal conductivity of the material of the cooling plate.

Density of cooling plate material — Density of material of cooling plate
`2500` `kg/m^3` (default)

Density of the material of the cooling plate.

Specific heat of cooling plate material — Specific heat of material of cooling plate
`447` `J/(K*kg` (default)

Specific heat of the material of the cooling plate.

Initial temperature of the cooling plate and the coolant fluid — Initial temperature of cooling plate and coolant fluid
`300` `K` (default)

Initial temperature of the cooling plate and of the coolant fluid.

Fluid Properties

Reynolds number upper limit for laminar flow — Upper limit of Reynolds number for laminar flow
`2000` (default)

Upper limit of Reynolds number for laminar flow.

Reynolds number lower limit for turbulent flow — Lower limit of Reynolds number for turbulent flow
`4000` (default)

Lower limit of Reynolds number for turbulent flow.

Nusselt number for laminar flow heat transfer — Nusselt number for laminar flow heat transfer
`3.66` (default)

Nusselt number for laminar flow heat transfer.

Aggregate equivalent length of local resistances — Aggregate equivalent length of local resistances
`1e-3 m` (default)

Aggregate equivalent length of local resistances.

Laminar friction constant for Darcy friction factor — Laminar friction constant for Darcy friction factor
`64` (default)

Laminar friction constant for the Darcy friction factor.

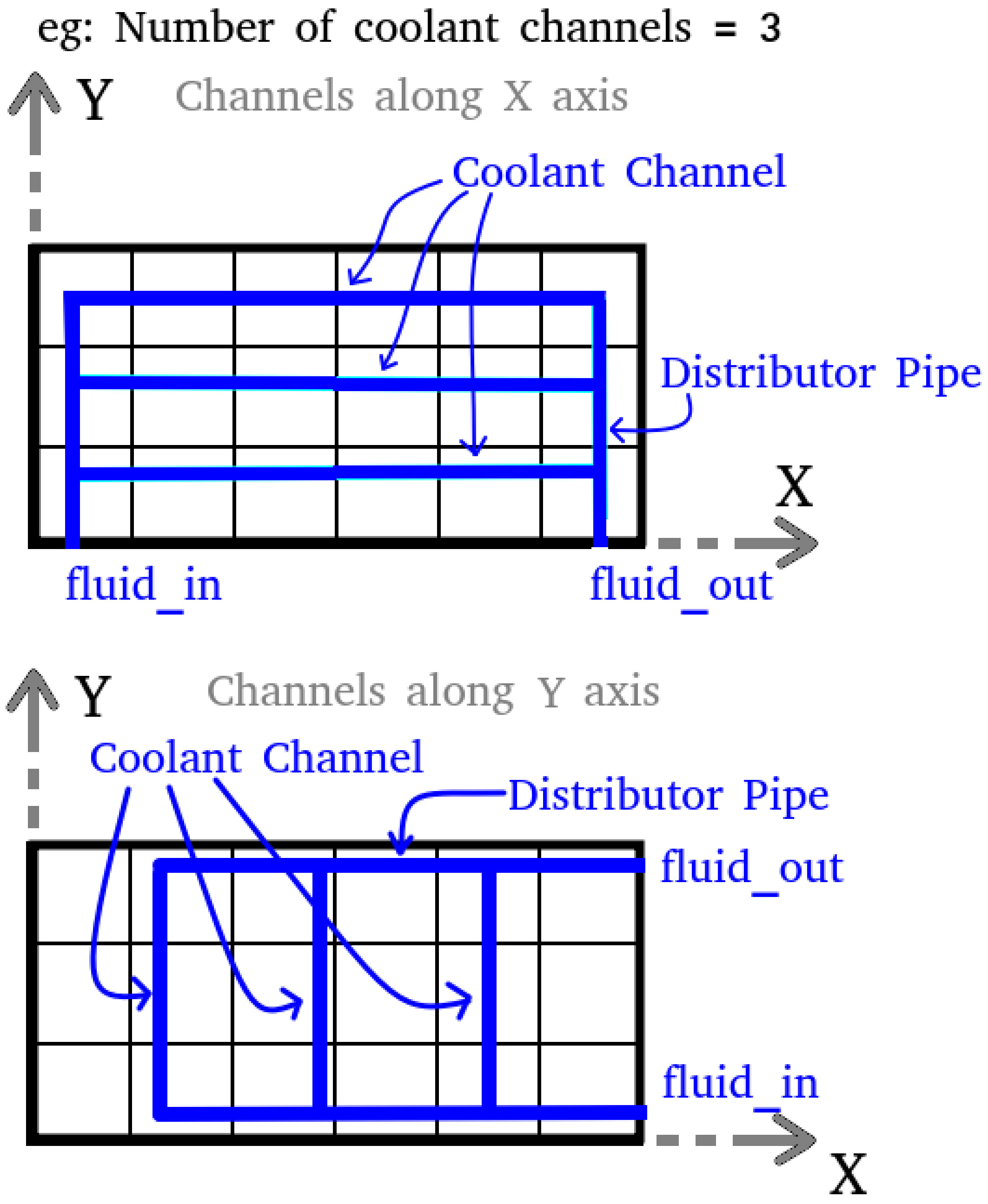
Design

Number of coolant channels — Number of coolant channels
`4` (default)

Number of the coolant channels.

Select channel orientation direction — Option for direction of channel orientation
`Channel along X axis` (default) | `Channel along Y axis`

Option to choose the axis direction of the channel orientation.



Cooling channel hydraulic diameter — Hydraulic diameter of cooling channel
`0.005` m (default)

Hydraulic diameter of the cooling channel.

Distributor pipe diameter — Diameter of distributor pipe
`0.005` m (default)

Diameter of the distributor pipe.

Coolant channel and distributor roughness — Roughness of coolant channel and distributor
`1.5e-05` m (default)

Roughness of the coolant channel and of the distributor.

Version History

Introduced in R2022b

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using Simulink® Coder™.

See Also

Simscape Blocks
arrayOfThermalNodesConnector | Edge Cooling | U-shaped Channels

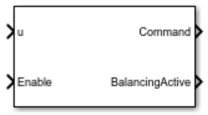
Objects
`Cell` | `ParallelAssembly` | `Module` | `ModuleAssembly` | `Pack`

Functions
`buildBattery`

Topics
“Connect Cooling Plate to Battery Blocks”

Passive Cell Balancing

Passive battery cell balancing algorithm



Libraries:
Simscape / Battery / BMS / Cell Balancing

Description

This block implements a passive battery cell balancing algorithm. The passive cell balancing technique keeps a similar state-of-charge value in all cells by dissipating the excess charge in a bleed resistor.

The value at the **u** input port can be cell voltages or cell state-of-charge (SOC) values. The **Enable** input port acts as a trigger for the balancing procedure. When **Enable** is `1`, the block starts balancing the battery until the battery is fully balanced.

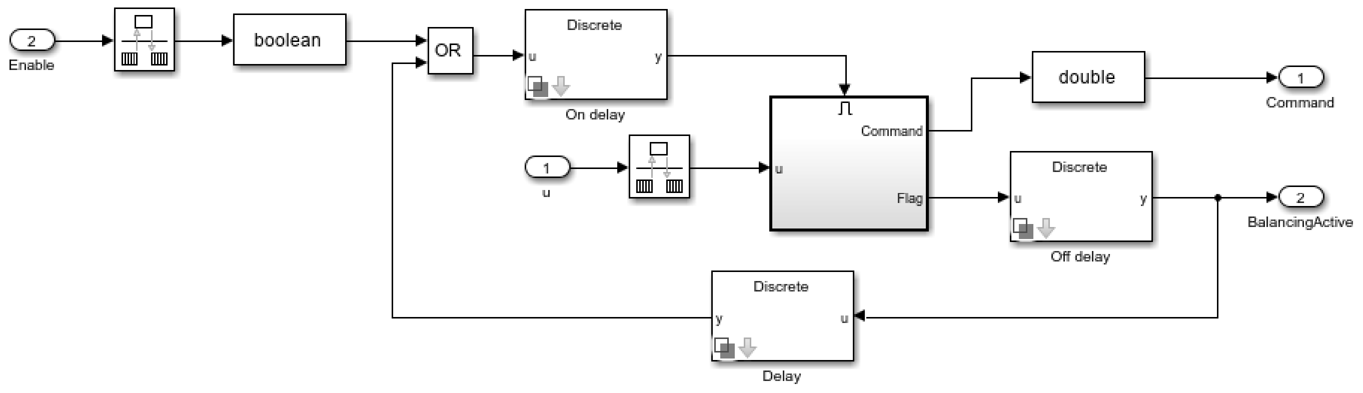
This block supports both single-precision and double-precision floating-point simulation.

Note To enable single-precision floating-point simulation, the data type of all inputs and parameters, except for the **Sample time (-1 for inherited)** parameter, must be `single`.

You can switch between continuous and discrete implementations of the block by using the **Sample time (-1 for inherited)** parameter. To configure the block for continuous time, set the **Sample time (-1 for inherited)** parameter to `0`. To configure the block for discrete time, set the **Sample time (-1 for inherited)** parameter to a positive, nonzero value, or to `-1` to inherit the sample time from an upstream block.

Note Continuous-time implementation of this block works only in double-precision floating-point simulation. If you provide single-precision floating-point parameters and inputs, this block casts them to double-precision floating-point values to prevent errors.

This diagram shows the structure of the block:



Equations

When the value at the **Enable** input port is `1`, the block starts balancing the battery until the battery is fully balanced. To balance the battery, the algorithm discharges some of the cells until their voltage or state-of-charge is equal to the voltage or state-of-charge of the cell with the lowest value.

The **Command** output port specifies the balancing command as a vector of elements equal to 0 or 1. This equation determines each element of **Command**:

$$Command = \begin{cases} 0, \ if \ u - min(u) < Threshold \\ 1, \ if \ u - min(u) \geq Threshold \end{cases}$$

To specify the data type of the **Command** output port, use the **Command Data Type** parameter.

The **BalancingActive** output port is true if any cell of the battery exceeds the threshold value.

Ports

Input

u — Cell voltages or cell SOC values
vector

Cell voltages or cell SOC values, specified as a vector.

Note The input port **u** expects cell voltages from a realistic sensor. To emulate the sensor and solve algebraic loops, if you are feeding simulation outputs from a model, add a unit delay, transfer function, or any other filter.

Enable — Whether to enable cell balancing
scalar

Whether to enable passive cell balancing, specified as `1` (enabled) or `0` (disabled).

Output

Command — Balancing command
vector

Balancing command, returned as a vector of entries equal to `0` or `1`. The size of this vector is equal to the size of the **u** input.

BalancingActive — Whether the balancing procedure is active
scalar

Whether the balancing procedure is active, returned as a scalar.

Parameters

Main

Threshold for balancing — Balancing threshold
`0.2` (default) | scalar

Threshold that the block uses for balancing.

ON delay time (s) — Time delay before balancing activation
`1` (default) | scalar

Time delay before activating the cell balancing, in seconds.

OFF delay time (s) — Time delay after balancing completion
`1` (default) | scalar

Time delay after the block completes the balancing procedure, in seconds.

Sample time (-1 for inherited) — Block sample time
`-1` (default) | 0 | positive integer

Time between consecutive block executions. During execution, the block produces outputs and, if appropriate, updates its internal state. For more information, see "What Is Sample Time?" and "Specify Sample Time".

For inherited discrete-time operation, specify this parameter as `-1`. For discrete-time operation, specify this parameter as a positive integer. For continuous-time operation, specify this parameter as `0`.

If this block is in a masked subsystem or a variant subsystem that allows you to switch between continuous operation and discrete operation, promote the sample time parameter. Promoting the sample time parameter ensures correct switching between the continuous and discrete implementations of the block. For more information, see "Promote Block Parameters on a Mask".

Signal Attributes

Command Data Type — Data type of Command output
`double` (default) | `single` | `boolean` | `<data type expression>`

Specify the data type of the **Command** output. You can specify the type directly or you can express it as a data type object such as `Simulink.NumericType`.

Click the **Show data type assistant** button >> to display the **Data Type Assistant**, which helps you set the data type attributes. For more information, see "Specify Data Types Using Data Type Assistant".

Version History

Introduced in R2022b

References

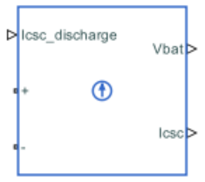
[1] Daowd, Mohamed, Noshin Omar, Peter Van Den Bossche, and Joeri Van Mierlo. "Passive and active battery balancing comparison based on MATLAB simulation." *2011 IEEE Vehicle Power and Propulsion Conference* (October 2011): 1-7. https://doi.org/10.1109/VPPC.2011.6043010.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using Simulink® Coder™.

Passive Interface

Passive interface between battery and cell supervisory circuit



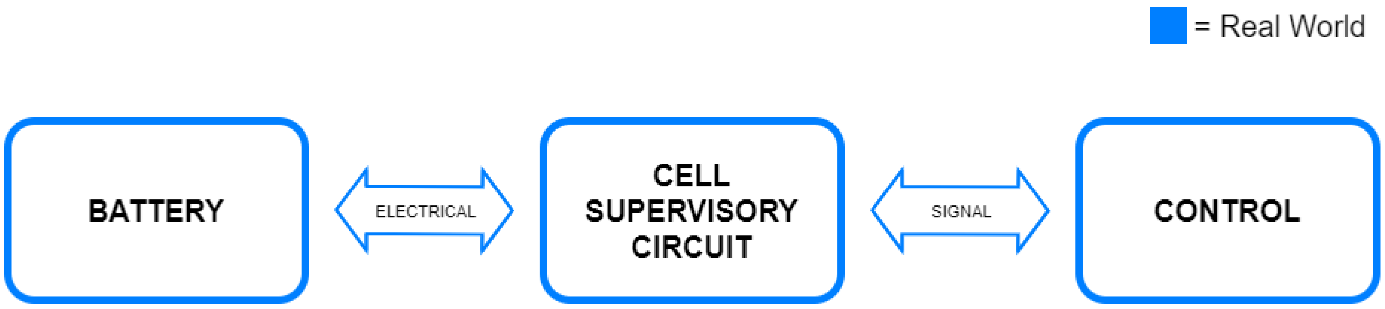
Libraries:
Simscape / Battery / HIL

Description

The Passive Interface block provides an abstracted passive interface between a battery and a cell supervisory circuit. Use this block with desktop simulations and hardware-in-the-loop battery emulation hardware.

A cell supervisory circuit is an electronic circuit that connects to battery cells or parallel assemblies. You can use this circuit to normalize voltages or states-of-charge (SOC) inside a module or pack. In the real world, you implement this equipment between your battery and the battery management system (BMS) control.

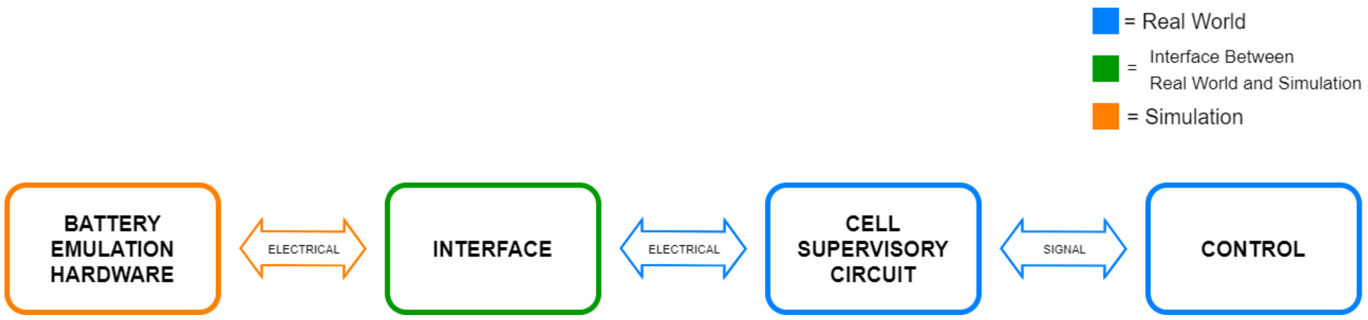
In this figure, the items in blue represent real-world equipment, voltages, and currents.



To avoid damaging your battery and to test your cell balancing algorithms under standard operational conditions and fault conditions, you can use hardware-in-the-loop battery emulation hardware. An abstracted interface facilitates connecting, setting, and configuring the passive balancing BMS to your emulated battery.

In this figure:

- The blue color represents real-world equipment, voltages, and currents.
- The green color represents the interface between the real-world equipment, voltages, and currents, as well as the simulated equipment, voltages, and currents.
- The orange color represents the simulated equipment, voltages, and currents.



The Passive Interface block provides the abstract representation of power electronic equipment without the need to model it in detail. For example, if you have a battery pack with different levels of temperature or voltage within its constituent parts, you can use this block to check if the BMS is correctly balancing your pack after a variety of operational scenarios.

SOC Optional Ports

Use the optional SOC ports of the Passive Interface block to avoid implementing an SOC estimation algorithm in the BMS, to validate and verify the SOC estimation algorithm against the simulated SOC, or to represent a real-world battery.

- If, in your model, the battery has an SOC output port, expose the SOC input port of this block and connect the ports appropriately. To expose the SOC input port of this block, set the **Expose SOC input port** parameter to `Yes`.
- To avoid implementing an SOC estimation algorithm in the BMS or to validate and verify the SOC estimation algorithm against the simulated SOC, expose the SOC output port of this block and connect it to your BMS. To expose the SOC output port of this block, set the **Expose SOC output port** parameter to `Yes`.
- To represent a real-world battery, do not expose the SOC ports of this block.

Thermal and Temperature Optional Ports

Use the optional thermal and temperature ports of the Passive Interface block to represent a battery with a temperature sensor or to validate and verify the temperature estimation algorithm against the simulated temperature.

- If, in your model, the battery has a temperature sensor, expose the thermal port **H** of this block and connect it to the thermal network. To expose the thermal port of this block, set the **Thermal port** parameter to `Model`.
- To validate and verify the temperature estimation algorithm against the simulated temperature, expose the thermal port **H** and the temperature measurement port **T** of this block and connect them to the thermal network and your BMS, respectively. To expose the temperature measurement port of this block, set the **Expose temperature measurement port** parameter to `Yes`.
- If, in your model, the battery does not include thermal effects or does not have a temperature sensor, do not expose the thermal and temperature ports of this block.

Ports

Input

Icsc_discharge — Discharge current from cell supervisory circuit, A
physical signal

Discharge current from the cell supervisory circuit, in ampere.

SOC — Battery state-of-charge, unitless
physical signal

State-of-charge of the battery, in percent.

Dependencies

To enable this port, set the **Expose SOC input port** parameter to `Yes`.

Output

Vbat — Battery voltage, V
physical signal

Voltage across the battery, in volt.

Icsc — Cell supervisory circuit current, A
physical signal

Current flowing through the cell supervisory circuit, in ampere.

The value at this port is always numerically identical to the value at the **Icsc_discharge** port. This port simply provides consistency with the Active Interface block or with any hardware that requires it.

Dependencies

To enable this port, set the **Expose cell supervisory circuit current measurement port** parameter to `Yes`.

SOC — Battery state-of-charge, unitless
physical signal

State-of-charge of the battery, in percent.

Dependencies

To enable this port, set the **Expose SOC output port** parameter to `Yes`.

T — Temperature measurement, K
physical signal

Temperature of the battery, in kelvin.

Dependencies

To enable this port, set the **Expose temperature measurement port** parameter to `Yes`.

Conserving

+ — Positive terminal
electrical

Electrical conserving port associated with the positive terminal of the interface.

- — Negative terminal
electrical

Electrical conserving port associated with the negative terminal of the interface.

H — Thermal port
thermal

Thermal conserving port.

Dependencies

To enable this port, set the **Thermal port** parameter to `Model`.

Parameters

Expose cell supervisory circuit current measurement port — Measurement port visibility for cell supervisory circuit current
`Yes` (default) | `No`

Option to expose the port that measures the current from the cell supervisory circuit.

Expose SOC input port — SOC input port visibility
`No` (default) | `Yes`

Option to expose the battery SOC input port.

Expose SOC output port — SOC output port visibility
`Yes` (default) | `No`

Option to expose the battery SOC output port.

Dependencies

To enable this parameter, set **Expose SOC input port** to `Yes`.

Thermal port — Thermal port visibility
`Omit` (default) | `Model`

Option to model the thermal port of the passive interface.

Expose temperature measurement port — Temperature measurement port visibility
`Yes` (default) | `No`

Option to expose the port that measures the temperature of the battery.

Dependencies

To enable this parameter, set **Thermal port** to `Model`.

Version History

Introduced in R2022b

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using Simulink® Coder™.

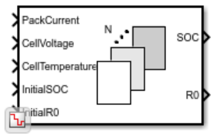
See Also

Active Interface

Topics
“Real-Time Model Preparation Workflow”
“Real-Time Simulation Workflow”
“Hardware-in-the-Loop Simulation Workflow”

SOC Estimator (Adaptive Kalman Filter)

State-of-charge and terminal resistance estimator with adaptive Kalman filter



Libraries:
Simscape / Battery / BMS / Estimators

Description

This block implements an estimator that calculates the state-of-charge (SOC) and the terminal resistance of a battery by using the Kalman filter algorithms. The terminal resistance, R_0, is a state of the estimator.

The SOC is defined as the ratio of the released capacity $C_{releasable}$ to the rated capacity C_{rated}. Manufacturers provide the value of the rated capacity of each battery, which represents the maximum amount of charge in the battery:

$$SOC = \frac{C_{releasable}}{C_{rated}}.$$

This block supports both single-precision and double-precision floating-point simulation.

Note To enable single-precision floating-point simulation, the data type of all inputs and parameters, except for the **Sample time (-1 for inherited)** parameter, must be `single`.

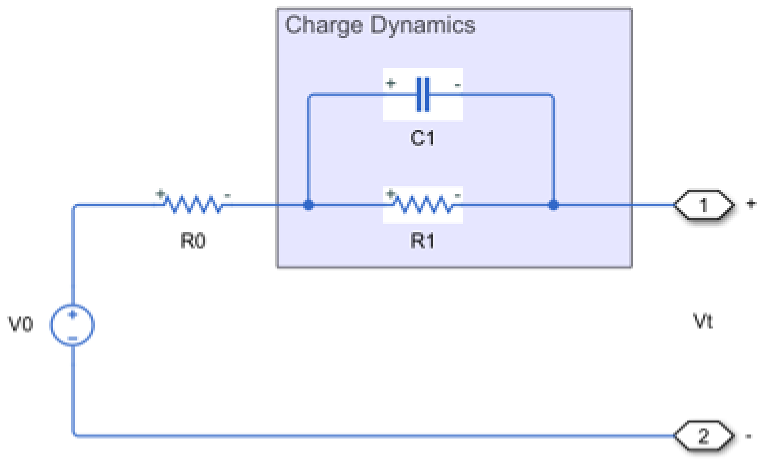
For continuous-time simulation, set the **Filter type** parameter to `Extended Kalman-Bucy filter` or `Unscented Kalman-Bucy filter`.

Note Continuous-time implementation of this block works only in double-precision floating-point simulation. If you provide single-precision floating-point parameters and inputs, this block casts them to double-precision floating-point values to prevent errors.

For discrete-time simulation, set the **Filter type** parameter to `Extended Kalman filter` or `Unscented Kalman filter` and the **Sample time (-1 for inherited)** parameter to a positive value or `-1`.

Equations

This figure shows the equivalent circuit for a battery with one time-constant dynamics:



The equations for the equivalent circuit, with the terminal resistance R_0 as an additional state, are:

$$\frac{dSOC}{dt} = -\frac{i}{3600AH(T)}$$

$$\frac{dV_1}{dt} = \frac{i}{C_1(SOC, T)} - \frac{V_1}{R_1(SOC, T)C_1(SOC, T)}$$

$$\frac{dR_0}{dt} = 0$$

$$V_t = V_0(SOC, T) - iR_0 - V_1$$

where

- *SOC* is the state-of-charge.
- *i* is the current.
- V_0 is the no-load voltage.
- V_t is the terminal voltage.
- *AH* is the ampere-hour rating.
- R_1 is the first polarization resistance.
- C_1 is the parallel RC capacitance.
- *T* is the temperature.

A time constant, τ_1, for the parallel section relates the first polarization resistance R_1 and the parallel RC capacitance C_1 using the relationship $C_1 = \tau_1/R_1$.

For the Kalman filter algorithms, the block uses this state and these process and observation nonlinear functions:

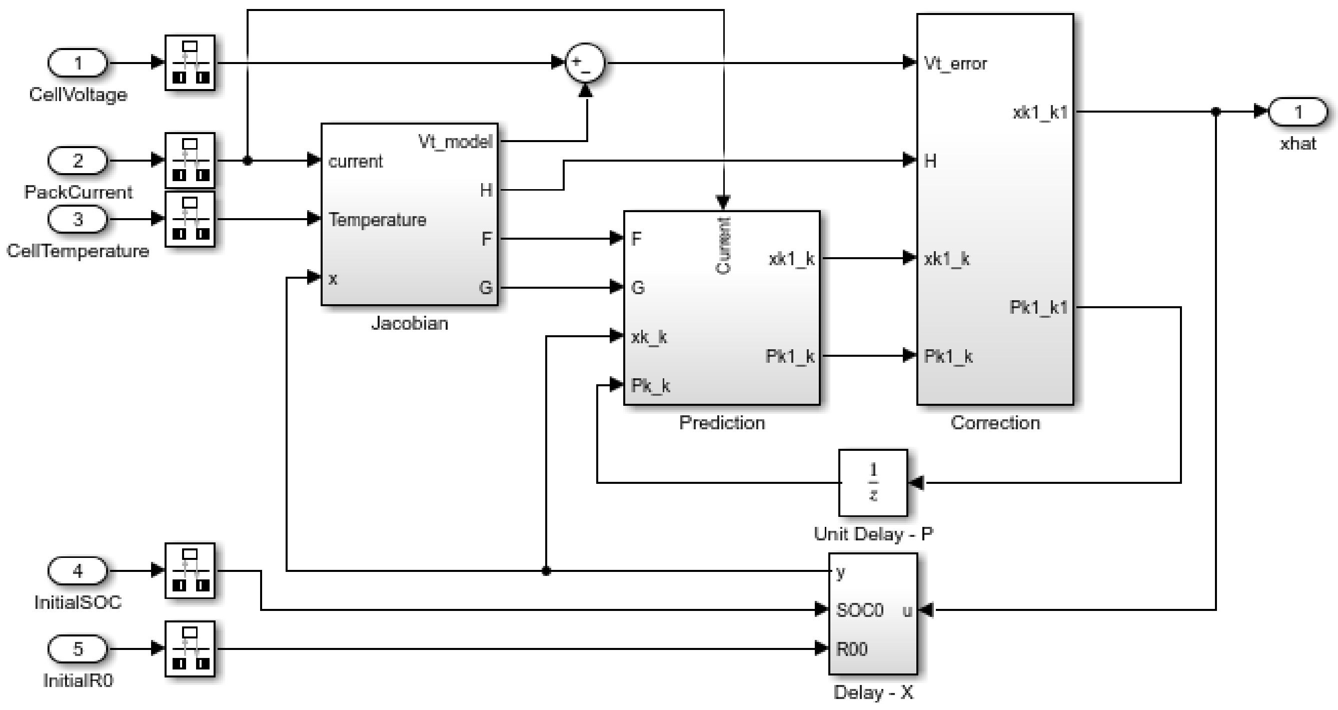
$$x = [SOC\ V_1\ R_0]^T$$

$$f(x,i) = \begin{bmatrix} -\dfrac{i}{3600AH(T)} \\ \dfrac{i}{C_1(SOC,T)} - \dfrac{V_1}{R_1(SOC,T)C_1(SOC,T)} \\ 0 \end{bmatrix}$$

$$h(x,i) = V_0(SOC,T) - iR_0 - V_1$$

Extended Kalman Filter

This diagram shows the structure of the extended Kalman filter (EKF):



The EKF technique relies on a linearization at every time step to approximate the nonlinear system. To linearize at every time step, the algorithm computes these Jacobians online:

$$F = \frac{\partial f}{\partial x}$$

$$H = \frac{\partial h}{\partial x}$$

The EKF is a discrete-time algorithm. After the discretization, the Jacobians for the SOC estimation of the battery are:

$$F_d = \begin{bmatrix} 1 & 0 & 0 \\ 0 & e^{\frac{-T_S}{R_1 C_1}} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$H_d = \begin{bmatrix} \frac{\partial V_{OC}}{\partial SOC} & -1 & -i \end{bmatrix}$$

where T_S is the sample time.

The EKF algorithm comprises these phases:

- **Initialization**
  - $\hat{x}(0|0)$— State estimate at time step 0 using measurements at time step 0.
  - $\hat{P}(0|0)$— State estimation error covariance matrix at time step 0 using measurements at time step 0.
- **Prediction**
  - Project the states ahead (*a priori*):

    $$\hat{\mathbf{x}}(k+1|k) = f(\hat{\mathbf{x}}(k|k), i)\,.$$
  - Project the error covariance ahead:

    $$\hat{\mathbf{P}}(k+1|k) = \mathbf{F}_d(k)\hat{\mathbf{P}}(k|k)\mathbf{F}_d^T(k) + \mathbf{Q},$$

    where $\mathbf{Q}$ is the covariance of the process noise.
- **Correction**
  - Compute the Kalman gain:

    $$\mathbf{K}(k+1) = \hat{\mathbf{P}}(k+1|k)\mathbf{H}_d^T(k)(\mathbf{H}_d(k)\hat{\mathbf{P}}(k+1|k)\mathbf{H}_d^T(k) + \mathbf{R})^{-1},$$

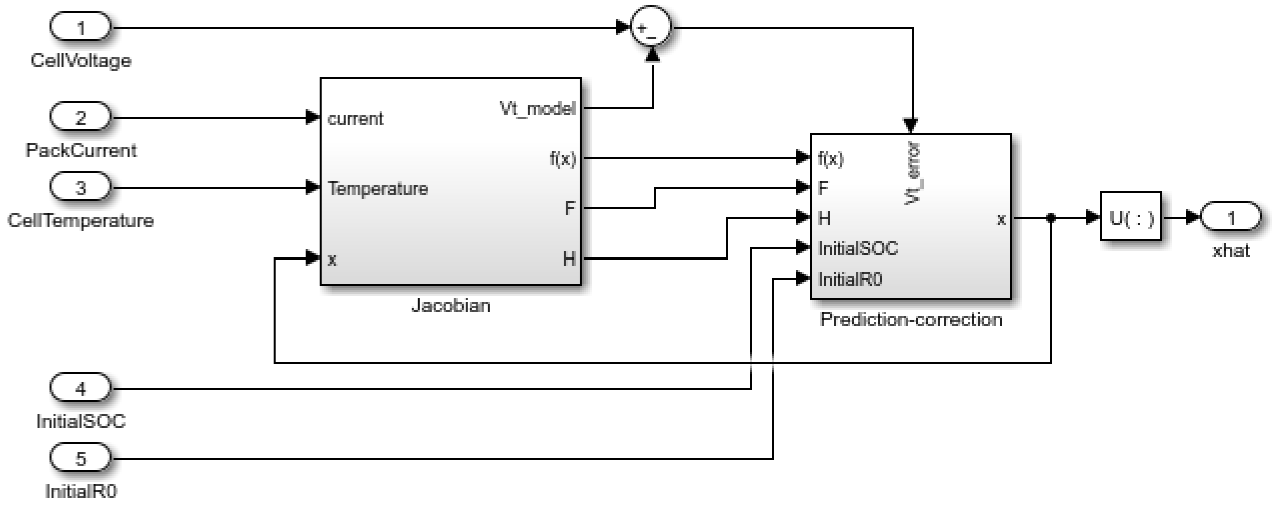
    where R is the covariance of the measurement noise.
  - Update the estimate with the measurement *y(k)* (*a posteriori*):

    $$\hat{\mathbf{x}}(k+1|k+1) = \hat{\mathbf{x}}(k+1|k) + \mathbf{K}(k+1)\left(V_t(k) - h(\hat{\mathbf{x}}(k|k), i)\right).$$
  - Update the error covariance:

    $$\hat{\mathbf{P}}(k+1|k+1) = (\mathbf{I} - \mathbf{K}(k+1)\mathbf{H}_d)\hat{\mathbf{P}}(k+1|k)\,.$$

Extended Kalman-Bucy Filter

This diagram shows the structure of the extended Kalman-Bucy filter (EKBF):



The EKBF is the continuous-time variant of the Kalman filter. In continuous-time, the prediction and correction steps are coupled. The EKBF algorithm comprises these phases:

- **Initialization**
  - $\widehat{\mathbf{x}}(t_0)$— State estimate at time t_0.
  - $\widehat{\mathbf{P}}(t_0)$— State estimation error covariance matrix at time t_0.
- **Prediction-Correction EKBF algorithm**

$$\mathbf{K}(t) = \mathbf{P}(t)\mathbf{H}^T(t)\mathbf{R}^{-1}(t)$$

$$\frac{d\widehat{\mathbf{x}}(t)}{dt} = f\left(\widehat{\mathbf{x}}(t), i(t)\right) + \mathbf{K}(t)\left(V_t(t) - h\left(\widehat{\mathbf{x}}(t), i(t)\right)\right)$$

$$\frac{d\mathbf{P}(t)}{dt} = \mathbf{F}(t)\mathbf{P}(t) + \mathbf{P}(t)\mathbf{F}^T(t) + \mathbf{Q}(t) - \mathbf{K}(t)\mathbf{H}(t)\mathbf{P}(t)$$

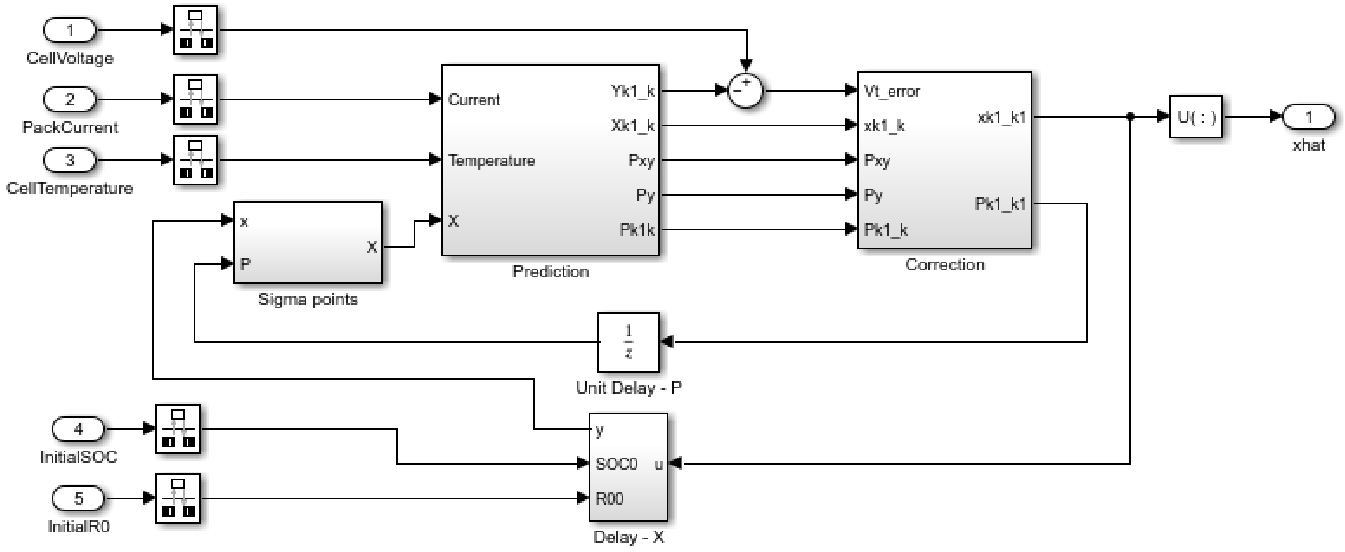
where:

$$F(t) = \frac{\partial f}{\partial x}$$

$$H(t) = \frac{\partial h}{\partial x}$$

Unscented Kalman Filter

This diagram shows the structure of the unscented Kalman filter (UKF):



The EKF locally approximates nonlinear functions with the linear equations obtained from the Taylor expansion by using only the first term of the expansion. In a highly nonlinear system, these solutions are not very accurate.

The UKF uses nonlinear transformations on a set of sigma points that the algorithm chooses deterministically. This technique is called unscented transformation. The mean and the covariance matrix of the transformed points are accurate to the second order of the Taylor series expansion.

The UKF algorithm follows these steps:

- **Initialization**
  - $\hat{x}(0|0)$— State estimate at time step 0 using measurements at time step 0.
  - $\hat{P}(0|0)$— State estimation error covariance matrix at time step 0 using measurements at time step 0.
- Generate sigma points and calculate the mean weight and covariance weight for each point.
  - Choose the sigma points, $x^{(i)}(k|k)$

$$\mathbf{x}^{(i)}(k|k) = \begin{cases} \hat{\mathbf{x}}(k+1|k) & i = 1 \\ \hat{\mathbf{x}}(k+1|k) + (\sqrt{(n+\lambda)\mathbf{P}(k|k)})_i & i = 2, \ldots, n+1 \\ \hat{\mathbf{x}}(k+1|k) - (\sqrt{(n+\lambda)\mathbf{P}(k|k)})_i & i = n+2, \ldots, 2n+1 \end{cases}$$

$$\mathbf{W}_m^{(i)} = \begin{cases} \dfrac{\lambda}{n+\lambda} & i = 1 \\ \dfrac{1}{2(n+\lambda)} & i \neq 1 \end{cases}$$

$$\mathbf{W}_c^{(i)} = \begin{cases} \dfrac{\lambda}{n+\lambda} + \left(1 - \alpha^2 + \beta\right) & i = 1 \\ \dfrac{1}{2(n+\lambda)} & i \neq 1 \end{cases}$$

where

- n is the dimension of the state vector x.
- $\lambda = \alpha^2(n + \kappa) - n$, $\alpha \in [0, 1]$ describes the distance between the sigma point and the mean point. In a normal distribution, $\beta = 2$ and $\kappa = 0$.
- $(\sqrt{(n + \lambda)\mathbf{P}})_i$ is the i-th row or column of $\sqrt{c\mathbf{P}}$. The block calculates the matrix square root by using numerically efficient and stable methods such as the Cholesky decomposition.

- First estimation of the system state matrix:

$$\widehat{\mathbf{x}}^{(i)}(k+1|k) = f\left(\widehat{\mathbf{x}}^{(i)}(k|k), i(k)\right)$$

$$\widehat{\mathbf{x}}(k+1|k) = \sum_{i=1}^{2n+1} \mathbf{W}_m^{(i)}\widehat{\mathbf{x}}^{(i)}(k+1|k)$$

- First estimation of the covariance matrix of the state variables:

$$\mathbf{P}(k+1|k) = \sum_{i=1}^{2n+1} \mathbf{W}_c^{(i)}\left(\widehat{\mathbf{x}}^{(i)}(k+1|k) - \widehat{\mathbf{x}}(k+1|k)\right) \cdot \left(\widehat{\mathbf{x}}^{(i)}(k+1|k) - \widehat{\mathbf{x}}(k+1|k)\right)^T + \mathbf{Q}$$

- Estimation of the measured variables:

$$V_t^{(i)}(k+1|k) = h\left(\widehat{\mathbf{x}}^{(i)}(k+1|k), i(k)\right)$$

$$\widehat{V}_t(k+1|k) = \sum_{i=1}^{2n+1} \mathbf{W}_m^{(i)}\widehat{V}_t^{(i)}(k+1|k)$$

- Estimation of the covariance of the measurement (P_y) and covariance between the measurement and the state (P_{xy}):

$$\mathbf{P}_y = \sum_{i=1}^{2n+1} \mathbf{W}_c^{(i)}\left(\widehat{V}_t^{(i)}(k+1|k) - \widehat{V}_t(k+1|k)\right) \cdot \left(\widehat{V}_t^{(i)}(k+1|k) - \widehat{V}_t(k+1|k)\right)^T + R$$

$$\mathbf{P}_{xy} = \sum_{i=1}^{2n+1} \left(\widehat{\mathbf{x}}^{(i)}(k+1|k) - \widehat{\mathbf{x}}(k+1|k)\right) \cdot \left(\widehat{V}_t^{(i)}(k+1|k) - \widehat{V}_t(k+1|k)\right)^T$$

- Kalman filter gain:

$$\mathbf{K}(k+1) = \mathbf{P}_{xy}\mathbf{P}_y^{-1}$$

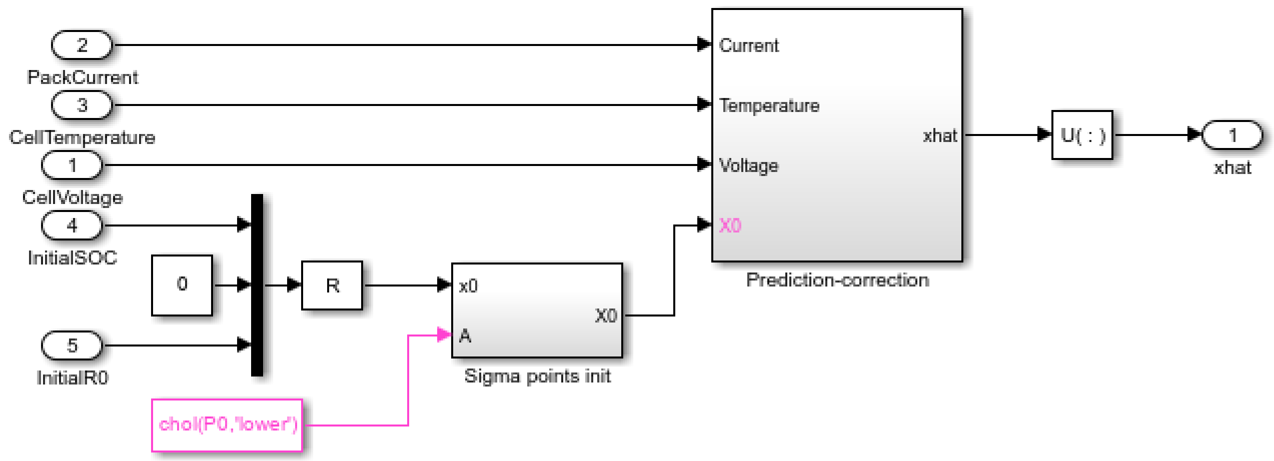
- Second update of the state matrix and of the covariance of the state variables:

$$\widehat{\mathbf{x}}(k+1|k+1) = \widehat{\mathbf{x}}(k+1|k) + \mathbf{K}(k+1)\left(V_t(k+1) - \widehat{V}_t(k+1|k)\right)$$

$$\mathbf{P}(k+1|k+1) = \mathbf{P}(k+1|k) - \mathbf{K}(k+1)\mathbf{P}_y\mathbf{K}^T(k+1)$$

Unscented Kalman-Bucy Filter

This diagram shows the structure of the unscented Kalman-Bucy filter (UKBF):



The derived continuous-time filtering equations of the UKBF are similar to the EKBF equations.

Because the UKF uses matrix square roots in its sigma points, the algorithm obtains the square-root version of the UKBF by formulating the filter as a differential equation for the sigma points. The equations for the square-root UKBF are:

$$\mathbf{K}(t) = \mathbf{X}(t)\mathbf{W}h^T(\mathbf{X}(t), t)\mathbf{R}^{-1}(t)$$

$$\mathbf{M}(t) = \mathbf{A}^{-1}(t)\left[\mathbf{X}(t)\mathbf{W}f^T(\mathbf{X}(t), t) + f(\mathbf{X}(t), t)\mathbf{W}\mathbf{X}^T(t) + \mathbf{Q}(t) - \mathbf{K}(t)\mathbf{R}(t)\mathbf{K}^T(t)\right]\mathbf{A}^{-T}(t)$$

$$\frac{d\mathbf{X}_i(t)}{dt} = f(\mathbf{X}(t), t)\mathbf{w}_m + \mathbf{K}(t)(V_t(t) - h(\mathbf{X}(t), t)\mathbf{w}_m) + \sqrt{c}[0\ \ \mathbf{A}(t)\Phi(\mathbf{M}(t))\ \ -\mathbf{A}(t)\Phi(\mathbf{M}(t))]_i$$

where

- $\mathbf{X}(t) = [m(t)\ \dots\ m(t)] + \sqrt{c}[\mathbf{0}\ \ \mathbf{A}(t)\ \ -\mathbf{A}(t)]$ is the sigma-point matrix.
- $\Phi_{ij}(\mathbf{M}(t)) = \begin{cases} \mathbf{M}_{ij}(t), & i > j \\ 0.5\mathbf{M}_{ij}(t), & i = j \\ 0, & i < j \end{cases}$ is a function that returns the lower diagonal part of the argument .
- $\mathbf{w}_m = \left[\mathbf{W}_m^{(1)}\ \cdots\ \mathbf{W}_m^{(2n+1)}\right]^T$
- $\mathbf{W} = (1 - [\mathbf{w}_m\ \cdots\ \mathbf{w}_m])diag\left(\mathbf{W}_c^{(1)}\ \cdots\ \mathbf{W}_c^{(2n+1)}\right)(I - [\mathbf{w}_m\ \cdots\ \mathbf{w}_m])^T$
- $c = \alpha^2(n + \kappa)$
- $[0\ \ \mathbf{A}(t)\Phi(\mathbf{M}(t))\ \ -\mathbf{A}(t)\Phi(\mathbf{M}(t))]_i$ is the i-th column of the argument matrix.

Assumptions and Limitations

- Process and sensor noises are independent, zero-mean, Gaussian noises.
- Battery nominal capacity does not consider aging.

Ports

Input

PackCurrent — Battery pack current
scalar

Battery pack current, in ampere, specified as a scalar.

CellVoltage — Cell voltage
scalar | vector

Cell voltage, in volt, specified as a scalar for a single cell or a vector for multiple cells. The size of this input port must be equal to the size of the **CellTemperature**, **InitialSOC**, and **InitialR0** input ports.

CellTemperature — Cell temperature
scalar | vector

Cell temperature, specified as a scalar for a single cell or a vector for multiple cells. The size of this input port must be equal to the size of the **CellVoltage**, **InitialSOC**, and **InitialR0** input ports.

InitialSOC — Initial state-of-charge
scalar | vector

Initial state-of-charge, specified as a scalar or vector of entries in the range [0, 1]. The size of this input port must be equal to the size of the **CellVoltage**, **CellTemperature**, and **InitialR0** input ports.

InitialR0 — Initial terminal resistance
scalar | vector

Initial terminal resistance, specified as a scalar or a vector. The size of this input port must be equal to the size of the **CellVoltage**, **CellTemperature**, and **InitialSOC** input ports.

Output

SOC — State-of-charge
scalar | vector

State-of-charge of the battery, returned as a scalar or a vector. The size of this output port is equal to the size of the **CellVoltage**, **CellTemperature**, **InitialSOC**, and **InitialR0** input ports.

R0 — Terminal resistance
scalar | vector

Terminal resistance, returned as a scalar or a vector. The size of this output port is equal to the size of the **CellVoltage**, **CellTemperature**, **InitialSOC**, and **InitialR0** input ports.

Parameters

Main

Filter type — Kalman filter type
`Extended Kalman filter` (default) | `Extended Kalman-Bucy filter` | `Unscented Kalman filter` | `Unscented Kalman-Bucy filter`

Type of Kalman filter that this block uses to estimate the battery state-of-charge and the terminal resistance.

Alpha — Alpha coefficient
`1` (default) | positive scalar between `0` and `1`

Coefficient that controls the spread of the sigma points. The block uses this parameter in its implementation of the equations for the unscented Kalman filter and the Unscented Kalman-Bucy filter.

Dependencies

To enable this parameter, set **Filter type** to `Unscented Kalman filter` or `Unscented Kalman-Bucy filter`.

Beta — Beta coefficient
`2` (default) | nonnegative scalar

Coefficient related to the distribution. The block uses this parameter in its implementation of the unscented Kalman filter and the Unscented Kalman-Bucy filter.

Dependencies

To enable this parameter, set **Filter type** to `Unscented Kalman filter` or `Unscented Kalman-Bucy filter`.

Kappa — Kappa coefficient
`0` (default) | nonnegative scalar between `0` and `3`

Coefficient that controls the spread of the sigma points. The block uses this parameter in its implementation of the equations for the unscented Kalman filter and the Unscented Kalman-Bucy filter.

Dependencies

To enable this parameter, set **Filter type** to `Unscented Kalman filter` or `Unscented Kalman-Bucy filter`.

Covariance of the process noise, Q — Covariance of process noise
`[1e-6 0 0; 0 1e-6 0;0 0 1e-6]` (default) | matrix of scalars

Covariance matrix of the noise in the states.

Covariance of the measurement noise, R — Covariance of measurement noise
`0.1` (default) | scalar

Covariance matrix of the noise in the measurements.

Initial state error covariance, P0 — Initial state error covariance
`[1e-5 0 0; 0 1 0; 0 0 1e-5]` (default) | matrix of scalars

Covariance matrix of the initial state error. This parameter defines the deviation in the initialization of the state.

Sample time (-1 for inherited) — Block sample time
`-1` (default) | positive scalar

Time between consecutive block executions. During execution, the block produces outputs and, if appropriate, updates its internal state. For more information, see “What Is Sample Time?” and “Specify Sample Time”.

For inherited discrete-time operation, specify this parameter as `-1`. For discrete-time operation, specify this parameter as a positive integer. For continuous-time operation, specify this parameter as `0`.

If this block is in a masked subsystem or a variant subsystem that allows you to switch between continuous operation and discrete operation, promote the sample time parameter. Promoting the sample time parameter ensures correct switching between the continuous and discrete implementations of the block. For more information, see “Promote Block Parameters on a Mask”.

Dependencies

To enable this parameter, set **Filter type** to `Extended Kalman filter` or `Unscented Kalman filter`.

System Model

Vector of state-of-charge values, SOC (-) — SOC breakpoints
`[0, .1, .25, .5, .75, .9, 1]` (default) | vector of nonnegative scalars between `0` and `1`

Vector of the state-of-charge breakpoints defining the points at which you specify lookup data. The entries of this vector must be in strictly ascending order. The block calculates the state-of-charge value with respect to the nominal battery capacity specified in the **Cell capacity, AH (A*Hr)** parameter. The SOC is the ratio of the available battery charge $q_{battery}$ and the nominal battery capacity $q_{nom}(T,n)$. You must make sure that, for each temperature, an SOC of 1 represents the respective battery charge capacity specified in the **Cell capacity, AH (A*Hr)** parameter when you model a fresh battery with a number of cycles N equal to 1 and $\delta_{AH}(n = 1, T_{fade}) = 0$.

$$SOC = \frac{q_{battery}}{q_{nom}(T, n)}$$

$$for\, N = 1\ and\, \delta_{AH}(n, T_{fade}) = 0,\ \ q_{nom}(T, n) = AH\,.$$

Vector of temperatures, T — *T* breakpoints
`[278, 293, 313]` (default) | vector of positive scalars

Vector of temperature breakpoints defining the points at which you specify lookup data. This vector must be strictly ascending and greater than `0 K`. The physical unit of this parameter must be the same as the physical unit of the **CellTemperature** input port.

First polarization resistance, R1(SOC,T), (ohm) — First RC resistance at temperature breakpoints
`[.0109, .0029, .0013; .0069, .0024, .0012; .0047, .0026, .0013; .0034, .0016, .001; .0033, .0023, .0014; .0033, .0018, .0011; .0028, .0017, .0011]` (default) | matrix of positive scalars

Lookup data, in ohm, for the first parallel RC resistance at the specified SOC and temperature breakpoints. The number of rows of this matrix is equal to the size of the **Vector of state-of-charge values, SOC (-)** parameter. The number of columns of this matrix is equal to the size of the **Vector of temperatures, T** parameter.

First time constant, tau1(SOC,T), (s) — First RC time constant at temperature breakpoints
`[20, 36, 39; 31, 45, 39; 109, 105, 61; 36, 29, 26; 59, 77, 67; 40, 33, 29; 25, 39, 33]` (default) | matrix of positive scalars

Lookup data, in second, for the first parallel RC time constant at the specified SOC and temperature breakpoints. The number of rows of this matrix is equal to the size of the **Vector of state-of-charge values, SOC (-)** parameter. The number of columns of this matrix is equal to the size of the **Vector of temperatures, T** parameter.

No-load voltage, V0(SOC,T), (V) — V_0 lookup table
`[3.49, 3.5, 3.51; 3.55, 3.57, 3.56; 3.62, 3.63, 3.64; 3.71, 3.71, 3.72; 3.91, 3.93, 3.94; 4.07, 4.08, 4.08; 4.19, 4.19, 4.19]` (default) | matrix of nonnegative scalars

Lookup data, in volt, for open-circuit voltages across the fundamental battery model at the specified SOC. The number of rows of this matrix is equal to the size of the **Vector of state-of-charge values, SOC (-)** parameter. The number of columns of this matrix is equal to the size of the **Vector of temperatures, T** parameter.

Cell capacity, AH (A*Hr) — Battery capacity when fully charged
`27` (default) | nonnegative scalar

Cell capacity of the battery, in A*Hr . The block calculates the state of charge by dividing the accumulated charge by this value. The block calculates the accumulated charge by integrating the battery current.

Version History

Introduced in R2022b

References

[1] Grewal, Mohinder S., and Angus P. Andrews. *Kalman Filtering: Theory and Practice Using MATLAB.* 2nd ed. New York: Wiley, 2001.

[2] Van der Merwe, R., and E. A. Wan, *The Square-Root Unscented Kalman Filter for State and Parameter-Estimation.* 2001 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings (Cat. No.01CH37221), vol. 6, IEEE, 2001, pp. 3461–64.

[3] Sarkka, Simo *On Unscented Kalman Filtering for State Estimation of Continuous-Time Nonlinear Systems.* IEEE Transactions on Automatic Control, vol. 52, no. 9, pp. 1631-1641, Sept. 2007, doi: 10.1109/TAC.2007.904453.

[4] Huria, Tarun, Massimo Ceraolo, Javier Gazzarri, and Robyn Jackey. *Simplified Extended Kalman Filter Observer for SOC Estimation of Commercial Power-Oriented LFP Lithium Battery Cells.*2013, pp. 2013-01–1544.

Extended Capabilities

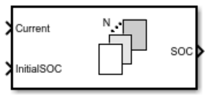
C/C++ Code Generation
Generate C and C++ code using Simulink® Coder™.

See Also

SOC Estimator (Coulomb Counting) | SOC Estimator (Coulomb Counting, Variable Capacity) | SOC Estimator (Kalman Filter) | SOH Estimator

SOC Estimator (Coulomb Counting)

State-of-charge estimator with Coulomb counting



Libraries:
Simscape / Battery / BMS / Estimators

Description

This block implements an estimator that calculates the state-of-charge (SOC) of a battery by using the Coulomb counting method.

The SOC is defined as the ratio of the released capacity $C_{releasable}$ to the rated capacity C_{rated}. Manufacturers provide the value of the rated capacity of each battery, which represents the maximum amount of charge in the battery:

$$SOC = \frac{C_{releasable}}{C_{rated}}.$$

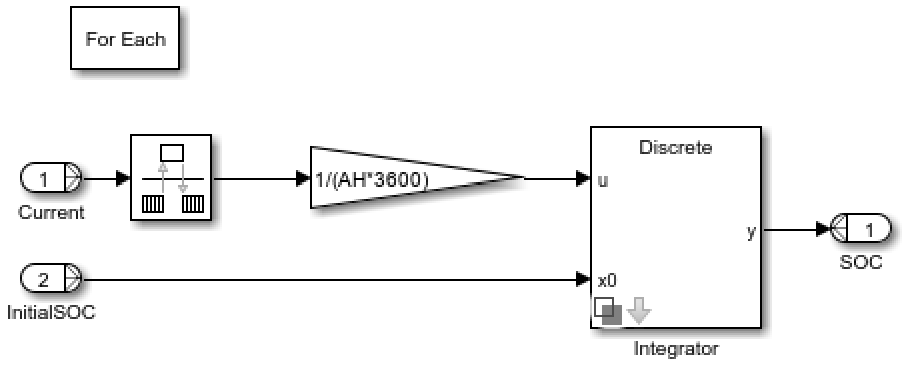
This block supports both single-precision and double-precision floating-point simulation.

Note To enable single-precision floating-point simulation, the data type of all inputs and parameters, except for the **Sample time (-1 for inherited)** parameter, must be `single`.

You can switch between continuous and discrete implementations of the block by using the **Sample time (-1 for inherited)** parameter. To configure the block for continuous time, set the **Sample time (-1 for inherited)** parameter to `0`. To configure the block for discrete time, set the **Sample time (-1 for inherited)** parameter to a positive, nonzero value, or to `-1` to inherit the sample time from an upstream block.

Note Continuous-time implementation of this block works only in double-precision floating-point simulation. If you provide single-precision floating-point parameters and inputs, this block casts them to double-precision floating-point values to prevent errors.

This diagram shows the structure of the block:



Equations

To compute the SOC of the battery, the SOC Estimator (Coulomb Counting) block counts the Ampere hour and current integration:

$$SOC = SOC(t_0) + \frac{1}{C_{rated}} \int_{t_0}^{t_0 + \tau} I_{batt}$$

where C_{rated} is the nominal battery capacity and I_{batt} is the battery current.

Assumptions and Limitations

The nominal capacity of the battery does not consider aging.

Ports

Input

Current — Battery current
scalar | vector

Battery current, in ampere, specified as a scalar for a single cell or a vector for multiple cells.

InitialSOC — Initial state-of-charge
scalar | vector

Initial state-of-charge, specified as a scalar or vector of entries in the range [0, 1]. The size of this input port must be equal to the size of the **Current** input port.

Output

SOC — Battery state-of-charge
scalar | vector

State-of-charge of the battery, returned as a scalar or a vector. The size of this output port is equal to the size of the **Current** and **InitialSOC** input ports.

Parameters

Cell capacity, AH (A*Hr) — Battery capacity when fully charged
27 (default) | nonnegative scalar

Cell capacity of the battery, in `A*Hr` . The block calculates the state of charge by dividing the accumulated charge by this value. The block calculates accumulated charge by integrating the battery current.

Sample time (-1 for inherited) — Block sample time
`-1` (default) | 0 | positive integer

Time between consecutive block executions. During execution, the block produces outputs and, if appropriate, updates its internal state. For more information, see “What Is Sample Time?” and “Specify Sample Time”.

For inherited discrete-time operation, specify this parameter as `-1`. For discrete-time operation, specify this parameter as a positive integer. For continuous-time operation, specify this parameter as `0`.

If this block is in a masked subsystem or a variant subsystem that allows you to switch between continuous operation and discrete operation, promote the sample time parameter. Promoting the sample time parameter ensures correct switching between the continuous and discrete implementations of the block. For more information, see “Promote Block Parameters on a Mask”.

Version History
Introduced in R2022b

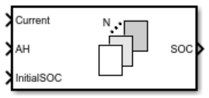
Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using Simulink® Coder™.

See Also
SOC Estimator (Coulomb Counting, Variable Capacity) | SOC Estimator (Adaptive Kalman Filter) | SOC Estimator (Kalman Filter) | SOH Estimator

SOC Estimator (Coulomb Counting, Variable Capacity)

State-of-charge estimator with Coulomb counting and variable capacity



Libraries:
Simscape / Battery / BMS / Estimators

Description

The SOC Estimator (Coulomb Counting, Variable Capacity) block implements an estimator that calculates the state-of-charge (SOC) of a battery by using the Coulomb counting method. The cell capacity of the battery is an input to the block. The block calculates the state of charge by dividing the accumulated charge by the value of this input. The block then calculates the accumulated charge by integrating the battery current.

The SOC is defined as the ratio of the released capacity $C_{releasable}$ to the rated capacity C_{rated}. Manufacturers provide the value of the rated capacity of each battery, which represents the maximum amount of charge in the battery:

$$SOC = \frac{C_{releasable}}{C_{rated}}.$$

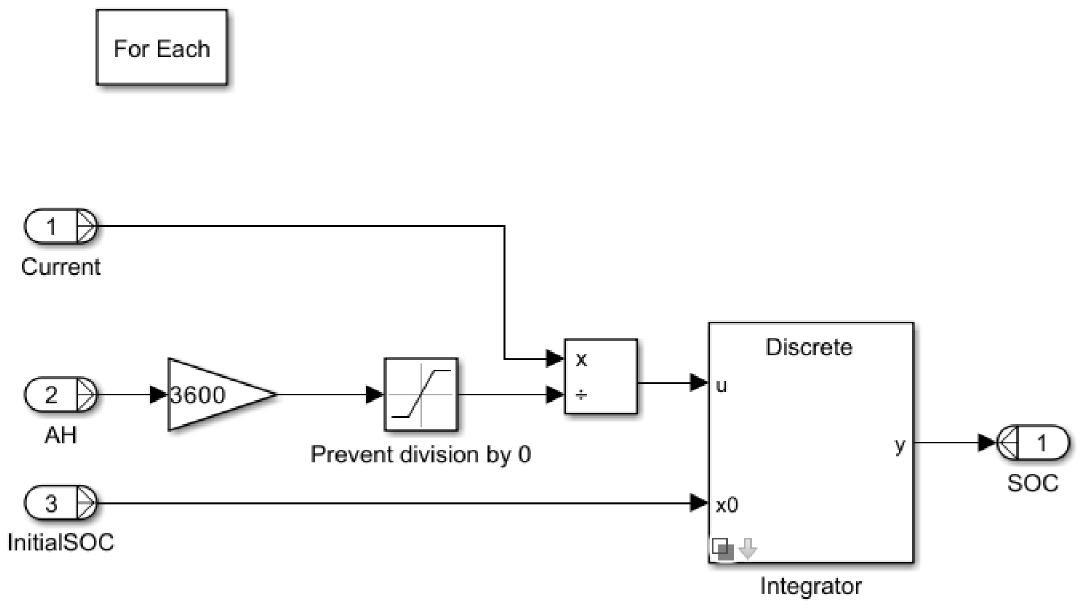
This block supports both single-precision and double-precision floating-point simulation.

Note To enable single-precision floating-point simulation, the data type of all inputs and parameters, except for the **Sample time (-1 for inherited)** parameter, must be `single`.

You can switch between continuous and discrete implementations of the block by using the **Sample time (-1 for inherited)** parameter. To configure the block for continuous time, set the **Sample time (-1 for inherited)** parameter to `0`. To configure the block for discrete time, set the **Sample time (-1 for inherited)** parameter to a positive, nonzero value, or to `-1` to inherit the sample time from an upstream block.

Note Continuous-time implementation of this block works only in double-precision floating-point simulation. If you provide single-precision floating-point parameters and inputs, this block casts them to double-precision floating-point values to prevent errors.

This diagram shows the structure of the block:



Equations

To compute the SOC of the battery, the SOC Estimator (Coulomb Counting, Variable Capacity) block counts the Ampere hour and current integration:

$$SOC = SOC(t_0) + \frac{1}{C_{rated}} \int_{t_0}^{t_0 + \tau} I_{batt}$$

where C_{rated} is the nominal battery capacity and I_{batt} is the battery current.

Assumptions and Limitations

The nominal capacity of the battery does not consider aging.

Ports

Input

Current — Battery current, ampere
scalar | vector

Battery current, in ampere, specified as a scalar for a single cell or a vector for multiple cells.

InitialSOC — Initial state-of-charge, unitless
scalar | vector

Initial state-of-charge, specified as a scalar or vector of entries in the range [0, 1]. The size of this input port must be equal to the size of the **Current** input port.

AH — Battery cell capacity, A*Hr
nonnegative scalar

Cell capacity of the battery, in `A*Hr`, specified as a strictly positive scalar. The block calculates the state of charge by dividing the accumulated charge by this value. The block calculates accumulated charge by integrating the battery current.

Output

SOC — Battery state-of-charge
scalar | vector

State-of-charge of the battery, returned as a scalar or a vector. The size of this output port is equal to the size of the **Current** and **InitialSOC** input ports.

Parameters

Sample time (-1 for inherited) — Block sample time
`-1` (default) | 0 | positive integer

Time between consecutive block executions. During execution, the block produces outputs and, if appropriate, updates its internal state. For more information, see “What Is Sample Time?” and “Specify Sample Time”.

For inherited discrete-time operation, specify this parameter as `-1`. For discrete-time operation, specify this parameter as a positive integer. For continuous-time operation, specify this parameter as `0`.

If this block is in a masked subsystem or a variant subsystem that allows you to switch between continuous operation and discrete operation, promote the sample time parameter. Promoting the sample time parameter ensures correct switching between the continuous and discrete implementations of the block. For more information, see “Promote Block Parameters on a Mask”.

Version History
Introduced in R2023a

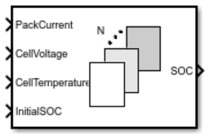
Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using Simulink® Coder™.

See Also
SOC Estimator (Coulomb Counting) | SOC Estimator (Adaptive Kalman Filter) | SOC Estimator (Kalman Filter) | SOH Estimator

SOC Estimator (Kalman Filter)

State-of-charge estimator with Kalman filter



Libraries:
Simscape / Battery / BMS / Estimators

Description

This block implements an estimator that calculates the state-of-charge (SOC) of a battery by using the Kalman filter algorithms.

The SOC is defined as the ratio of the released capacity $C_{releasable}$ to the rated capacity C_{rated}. Manufacturers provide the value of the rated capacity of each battery, which represents the maximum amount of charge in the battery:

$$SOC = \frac{C_{releasable}}{C_{rated}}.$$

This block supports both single-precision and double-precision floating-point simulation.

Note To enable single-precision floating-point simulation, the data type of all inputs and parameters, except for the **Sample time (-1 for inherited)** parameter, must be `single`.

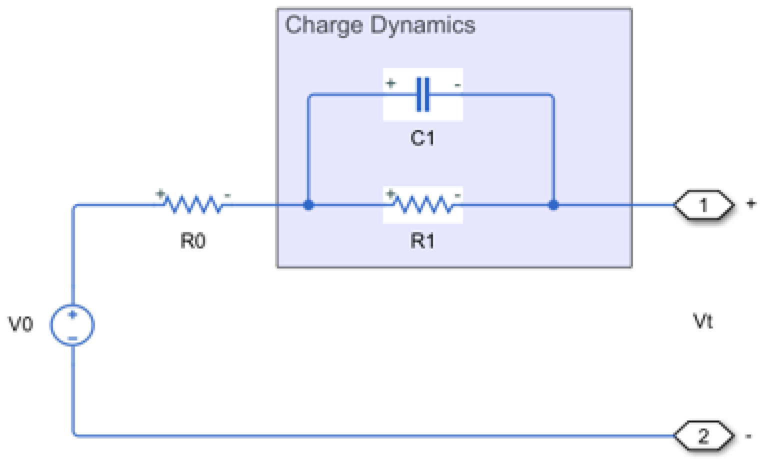
For continuous-time simulation, set the **Filter type** parameter to `Extended Kalman-Bucy filter` or `Unscented Kalman-Bucy filter`.

Note Continuous-time implementation of this block works only in double-precision floating-point simulation. If you provide single-precision floating-point parameters and inputs, this block casts them to double-precision floating-point values to prevent errors.

For discrete-time simulation, set the **Filter type** parameter to `Extended Kalman filter` or `Unscented Kalman filter` and the **Sample time (-1 for inherited)** parameter to a positive value or `-1`.

Equations

This figure shows the equivalent circuit for a battery with one time-constant dynamics:



The equations for the equivalent circuit, with the terminal resistance R_0 as an additional state, are:

$$\frac{dSOC}{dt} = -\frac{i}{3600AH(T)}$$

$$\frac{dV_1}{dt} = \frac{i}{C_1(SOC, T)} - \frac{V_1}{R_1(SOC, T)C_1(SOC, T)}$$

$$V_t = V_0(SOC, T) - iR_0 - V_1$$

where

- SOC is the state-of-charge.
- i is the current.
- V_0 is the no-load voltage.
- V_t is the terminal voltage.
- AH is the ampere-hour rating.
- R_1 is the first polarization resistance.
- C_1 is the parallel RC capacitance.
- T is the temperature.

A time constant, τ_1, for the parallel section relates the first polarization resistance R_1 and the parallel RC capacitance C_1 using the relationship $C_1 = \tau_1/R_1$.

For the Kalman filter algorithms, the block uses this state and these process and observation functions:

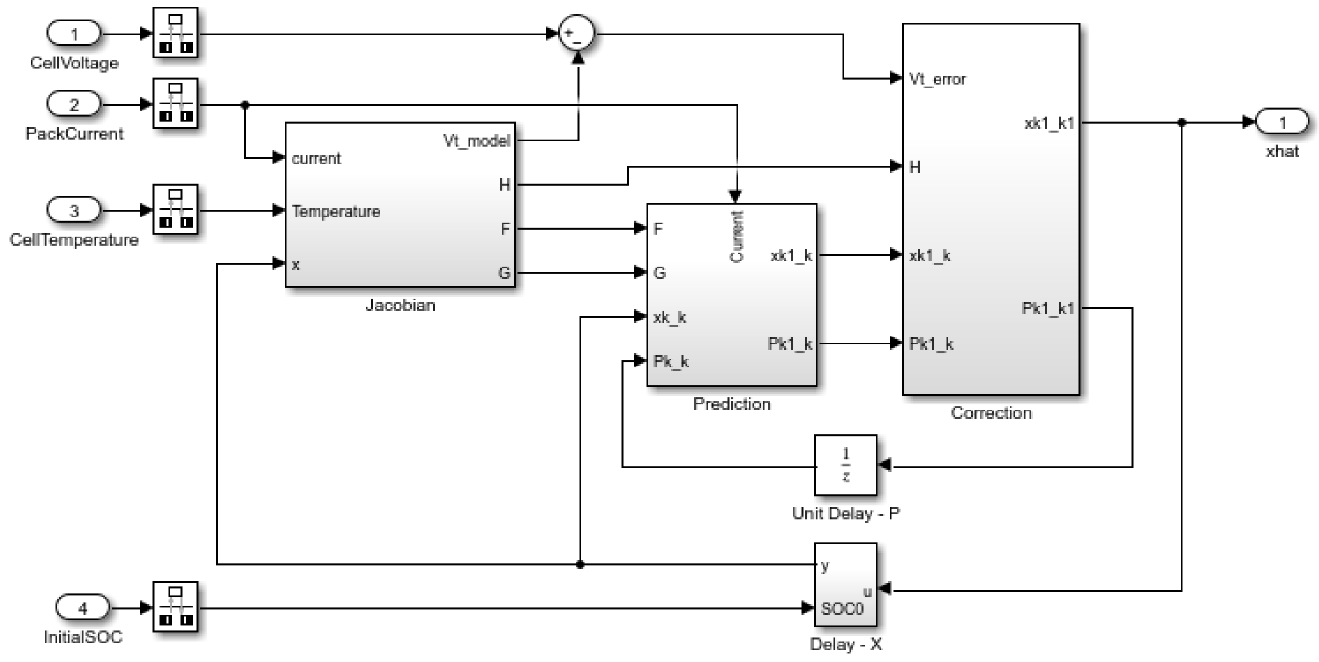
$$x = \left[SOC\ V_1\right]^T$$

$$f(x, i) = \begin{bmatrix} -\dfrac{i}{3600AH(T)} \\ \dfrac{i}{C_1(SOC, T)} - \dfrac{V_1}{R_1(SOC, T)C_1(SOC, T)} \end{bmatrix}$$

$$h(x, i) = V_0(SOC, T) - iR_0 - V_1$$

Extended Kalman Filter

This diagram shows the structure of the extended Kalman filter (EKF):



The EKF technique relies on a linearization at every time step to approximate the nonlinear system. To linearize at every time step, the algorithm computes these Jacobians online:

$$F = \frac{\partial f}{\partial x}$$

$$H = \frac{\partial h}{\partial x}$$

The EKF is a discrete-time algorithm. After the discretization, the Jacobians for the SOC estimation of the battery are:

$$\mathbf{F}_d = \begin{bmatrix} 1 & 0 \\ 0 & e^{\frac{-T_S}{R_1 C_1}} \end{bmatrix}$$

$$\mathbf{H}_d = \begin{bmatrix} \frac{\partial V_{OC}}{\partial SOC} & -1 \end{bmatrix}$$

where T_S is the sample time.

The EKF algorithm comprises these phases:

- **Initialization**
  - $\widehat{x}(0|0)$— State estimate at time step 0 using measurements at time step 0.

  - $\widehat{P}(0\,|\,0)$— State estimation error covariance matrix at time step 0 using measurements at time step 0.
- **Prediction**
  - Project the states ahead (a priori):

    $$\widehat{\mathbf{x}}(k+1|k) = f(\widehat{\mathbf{x}}(k|k), i).$$
  - Project the error covariance ahead:

    $$\widehat{\mathbf{P}}(k+1|k) = \mathbf{F}_d(k)\widehat{\mathbf{P}}(k|k)\mathbf{F}_d^T(k) + \mathbf{Q},$$

    where **Q** is the covariance of the process noise.
- **Correction**
  - Compute the Kalman gain:

    $$\mathbf{K}(k+1) = \widehat{\mathbf{P}}(k+1\,|\,k)\mathbf{H}_d^T(k)(\mathbf{H}_d(k)\widehat{\mathbf{P}}(k+1\,|\,k)\mathbf{H}_d^T(k) + \mathbf{R})^{-1},$$

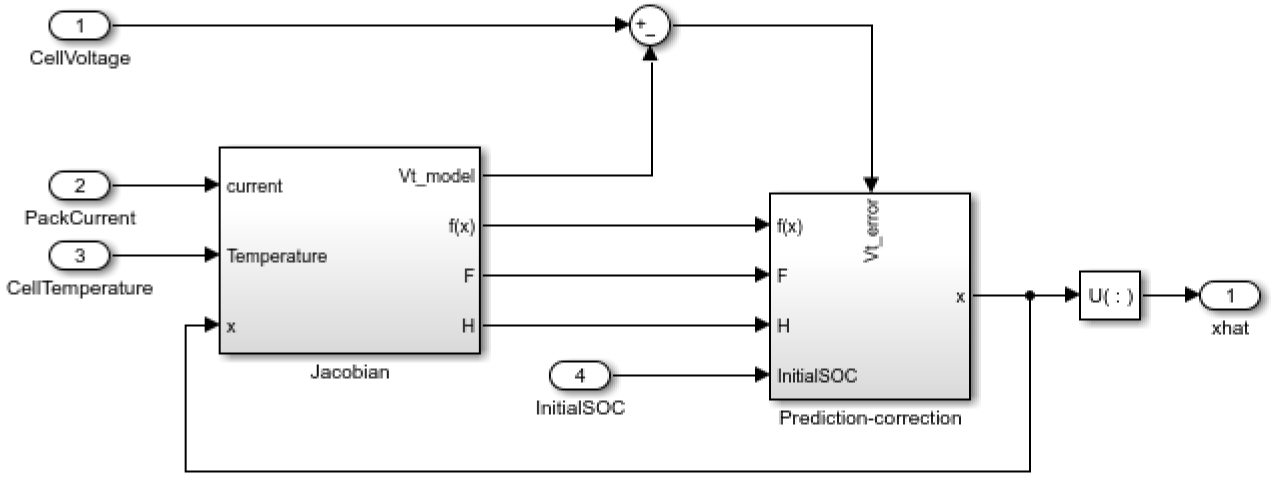
    where R is the covariance of the measurement noise.
  - Update the estimate with the measurement *y(k)* (a posteriori):

    $$\widehat{\mathbf{x}}(k+1|k+1) = \widehat{\mathbf{x}}(k+1|k) + \mathbf{K}(k+1)(V_t(k) - h(\widehat{\mathbf{x}}(k|k), i)).$$
  - Update the error covariance:

    $$\widehat{\mathbf{P}}(k+1|k+1) = (\mathbf{I} - \mathbf{K}(k+1)\mathbf{H}_d)\widehat{\mathbf{P}}(k+1|k).$$

Extended Kalman-Bucy Filter

This diagram shows the structure of the extended Kalman-Bucy filter (EKBF):



The EKBF is the continuous-time variant of the Kalman filter. In continuous-time, the prediction and correction steps are coupled.

- **Initialization**
  - $\widehat{\mathbf{x}}(t_0)$— State estimate at time t_0.
  - $\widehat{\mathbf{P}}(t_0)$— State estimation error covariance matrix at time t_0.
- **Prediction-Correction EKBF algorithm**

$$\mathbf{K}(t) = \mathbf{P}(t)\mathbf{H}^T(t)\mathbf{R}^{-1}(t)$$

$$\frac{d\widehat{\mathbf{x}}(t)}{dt} = f(\widehat{\mathbf{x}}(t), i(t)) + \mathbf{K}(t)(V_t(t) - h(\widehat{\mathbf{x}}(t), i(t)))$$

$$\frac{d\mathbf{P}(t)}{dt} = \mathbf{F}(t)\mathbf{P}(t) + \mathbf{P}(t)\mathbf{F}^T(t) + \mathbf{Q}(t) - \mathbf{K}(t)\mathbf{H}(t)\mathbf{P}(t)$$

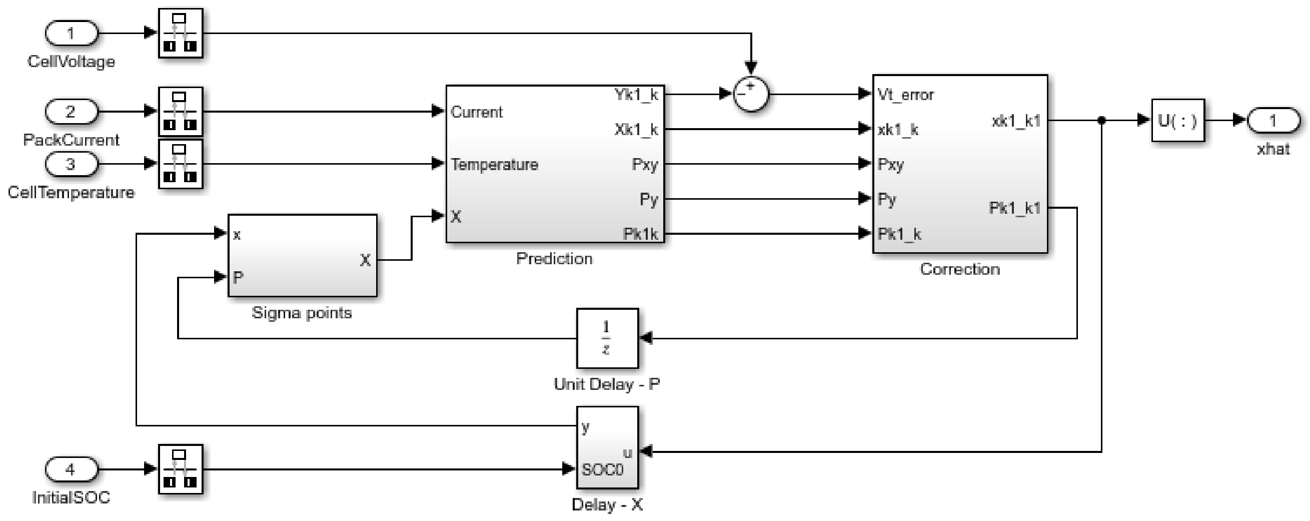
where:

$$F(t) = \frac{\partial f}{\partial x}$$

$$H(t) = \frac{\partial h}{\partial x}$$

Unscented Kalman Filter

This diagram shows the structure of the unscented Kalman filter (UKF):



The EKF locally approximates nonlinear functions with the linear equations obtained from the Taylor expansion by using only the first term of the expansion. In a highly nonlinear system, these solutions are not very accurate.

The UKF uses nonlinear transformations on a set of sigma points that the algorithm chooses deterministically. This technique is called unscented transformation. The mean and the covariance matrix of the transformed points are accurate to the second order of the Taylor series expansion.

The UKF algorithm follows these steps:

- **Initialization**
  - $\widehat{x}(0|0)$— State estimate at time step 0 using measurements at time step 0.
  - $\widehat{P}(0|0)$— State estimation error covariance matrix at time step 0 using measurements at time step 0.
- Generate sigma points and calculate the mean weight and covariance weight for each point.
  - Choose the sigma points $x^{(i)}(k|k)$

$$\mathbf{x}^{(i)}(k|k) = \begin{cases} \widehat{\mathbf{x}}(k+1|k) & i = 1 \\ \widehat{\mathbf{x}}(k+1|k) + (\sqrt{(n+\lambda)\mathbf{P}(k|k)})_i & i = 2, \ldots, n+1 \\ \widehat{\mathbf{x}}(k+1|k) - (\sqrt{(n+\lambda)\mathbf{P}(k|k)})_i & i = n+2, \ldots, 2n+1 \end{cases}$$

$$\mathbf{W}_m^{(i)} = \begin{cases} \frac{\lambda}{n+\lambda} & i = 1 \\ \frac{1}{2(n+\lambda)} & i \neq 1 \end{cases}$$

$$\mathbf{W}_c^{(i)} = \begin{cases} \frac{\lambda}{n+\lambda} + \left(1 - \alpha^2 + \beta\right) & i = 1 \\ \frac{1}{2(n+\lambda)} & i \neq 1 \end{cases}$$

  where

  - n is the dimension of the state vector, x.
  - $\lambda = \alpha^2(n+\kappa) - n$, $\alpha \in [0,1]$ describes the distance between the sigma point and the mean point. In a normal distribution, $\beta = 2$ and $\kappa = 0$.
  - $(\sqrt{(n+\lambda)\mathbf{P}})_i$ is the ith row or column of $\sqrt{c\mathbf{P}}$. The block calculates the matrix square root by using numerically efficient and stable methods such as the Cholesky decomposition.
- First estimation of the system state matrix:

$$\widehat{\mathbf{x}}^{(i)}(k+1|k) = f\left(\widehat{\mathbf{x}}^{(i)}(k|k), i(k)\right)$$

$$\widehat{\mathbf{x}}(k+1|k) = \sum_{i=1}^{2n+1} \mathbf{W}_m^{(i)} \widehat{\mathbf{x}}^{(i)}(k+1|k)$$

- First estimation of the covariance matrix of the state variables:

$$\mathbf{P}(k+1|k) = \sum_{i=1}^{2n+1} \mathbf{W}_c^{(i)} \left(\widehat{\mathbf{x}}^{(i)}(k+1|k) - \widehat{\mathbf{x}}(k+1|k)\right) \cdot \left(\widehat{\mathbf{x}}^{(i)}(k+1|k) - \widehat{\mathbf{x}}(k+1|k)\right)^T + \mathbf{Q}$$

- Estimation of the measured variables:

$$V_t^{(i)}(k+1|k) = h\left(\widehat{\mathbf{x}}^{(i)}(k+1|k), i(k)\right)$$

$$\widehat{V}_t(k+1|k) = \sum_{i=1}^{2n+1} \mathbf{W}_m^{(i)} \widehat{V}_t^{(i)}(k+1|k)$$

- Estimation of the covariance of the measurement (P_y) and covariance between the measurement and the state (P_{xy}):

$$\mathbf{P}_y = \sum_{i=1}^{2n+1} \mathbf{W}_c^{(i)}\left(\widehat{V}_t^{(i)}(k+1|k) - \widehat{V}_t(k+1|k)\right)\cdot\left(\widehat{V}_t^{(i)}(k+1|k) - \widehat{V}_t(k+1|k)\right)^T + R$$

$$\mathbf{P}_{xy} = \sum_{i=1}^{2n+1}\left(\widehat{\mathbf{x}}^{(i)}(k+1|k) - \widehat{\mathbf{x}}(k+1|k)\right)\cdot\left(\widehat{V}_t^{(i)}(k+1|k) - \widehat{V}_t(k+1|k)\right)^T$$

- Kalman filter gain:

$$\mathbf{K}(k+1) = \mathbf{P}_{xy}\mathbf{P}_y^{-1}$$

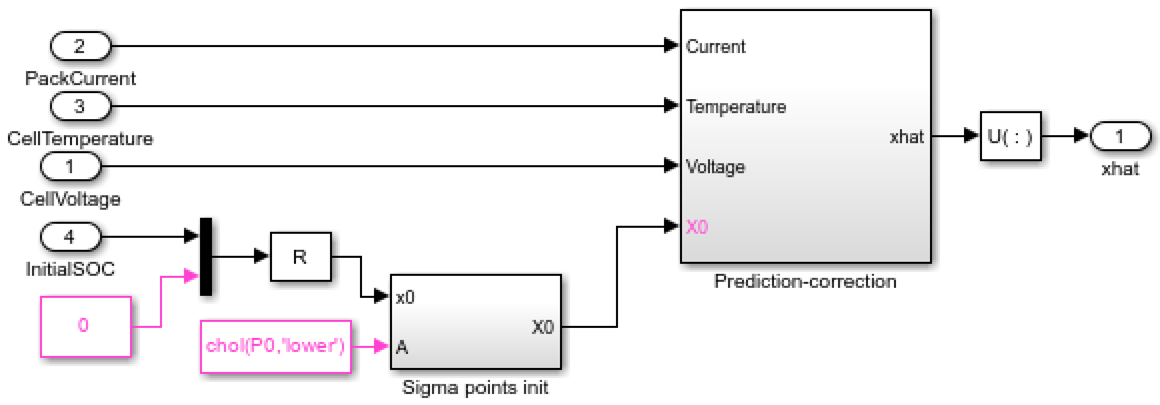
- Second update of the state matrix and of the covariance of the state variables:

$$\widehat{\mathbf{x}}(k+1|k+1) = \widehat{\mathbf{x}}(k+1|k) + \mathbf{K}(k+1)\left(V_t(k+1) - \widehat{V}_t(k+1|k)\right)$$

$$\mathbf{P}(k+1|k+1) = \mathbf{P}(k+1|k) - \mathbf{K}(k+1)\mathbf{P}_y\mathbf{K}^T(k+1)$$

Unscented Kalman-Bucy Filter

This diagram illustrates the overall structure of the unscented Kalman-Bucy filter (UKBF):



The derived continuous-time filtering equations of the UKBF are similar to the EKBF equations.

Because the UKF uses matrix square roots in its sigma points, the algorithm obtains the square-root version of the UKBF by formulating the filter as a differential equation for the sigma points. The equations for the square-root UKBF are:

$$\mathbf{K}(t) = \mathbf{X}(t)\mathbf{W}h^T(\mathbf{X}(t), t)\mathbf{R}^{-1}(t)$$

$$\mathbf{M}(t) = \mathbf{A}^{-1}(t)\left[\mathbf{X}(t)\mathbf{W}f^T(\mathbf{X}(t), t) + f(\mathbf{X}(t), t)\mathbf{W}\mathbf{X}^T(t) + \mathbf{Q}(t) - \mathbf{K}(t)\mathbf{R}(t)\mathbf{K}^T(t)\right]\mathbf{A}^{-T}(t)$$

$$\frac{d\mathbf{X}_i(t)}{dt} = f(\mathbf{X}(t), t)\mathbf{w}_m + \mathbf{K}(t)(V_t(t) - h(\mathbf{X}(t), t)\mathbf{w}_m) + \sqrt{c}[0\ \ \mathbf{A}(t)\Phi(\mathbf{M}(t))\ \ -\mathbf{A}(t)\Phi(\mathbf{M}(t))]_i$$

where

- $\mathbf{X}(t) = [m(t)\ \ldots\ m(t)] + \sqrt{c}[\mathbf{0}\ \ \mathbf{A}(t)\ \ -\mathbf{A}(t)]$ is the sigma-point matrix.
- $\Phi_{ij}(\mathbf{M}(t)) = \begin{cases} \mathbf{M}_{ij}(t), & i > j \\ 0.5\mathbf{M}_{ij}(t), & i = j \\ 0, & i < j \end{cases}$ is a function that returns the lower diagonal part of the argument .

- $\mathbf{w}_m = \left[\mathbf{W}_m^{(1)} \cdots \mathbf{W}_m^{(2n+1)}\right]^T$
- $\mathbf{W} = (1 - [\mathbf{w}_m \cdots \mathbf{w}_m])diag\left(\mathbf{W}_c^{(1)} \cdots \mathbf{W}_c^{(2n+1)}\right)(I - [\mathbf{w}_m \cdots \mathbf{w}_m])^T$
- $c = \alpha^2(n + \kappa)$
- $[0 \;\; \mathbf{A}(t)\Phi(\mathbf{M}(t)) \;\; -\mathbf{A}(t)\Phi(\mathbf{M}(t))]_i$ is the ith column of the argument matrix.

Ports

Input

PackCurrent — Battery pack current
scalar

Battery pack current, in ampere, specified as a scalar.

CellVoltage — Cell voltage
scalar | vector

Cell voltage, in volt, specified as a scalar for a single cell or a vector for multiple cells. The size of this input port must be equal to the size of the **CellTemperature** and **InitialSOC** input ports.

CellTemperature — Cell temperature
scalar | vector

Cell temperature, specified as a scalar for a single cell or a vector for multiple cells. The size of this input port must be equal to the size of the **CellVoltage** and **InitialSOC** input ports.

InitialSOC — Initial state-of-charge
scalar | vector

Initial state-of-charge, specified as a scalar or vector of entries in the range [0, 1]. The size of this input port must be equal to the size of the **CellVoltage** and **CellTemperature** input ports.

Output

SOC — State-of-charge
scalar | vector

State-of-charge of the battery, returned as a scalar or a vector. The size of this output port is equal to the size of the **CellVoltage**, **CellTemperature**, and **InitialSOC** input ports.

Parameters

Main

Filter type — Kalman filter type
`Extended Kalman filter` (default) | `Extended Kalman-Bucy filter` | `Unscented Kalman filter` | `Unscented Kalman-Bucy filter`

Type of Kalman filter that this block uses to estimate the battery state-of-charge.

Alpha — Alpha coefficient
`1` (default) | positive scalar between `0` and `1`

Coefficient that controls the spread of the sigma points. The block uses this parameter in its implementation of the equations for the unscented Kalman filter and the Unscented Kalman-Bucy filter.

Dependencies

To enable this parameter, set **Filter type** to `Unscented Kalman filter` or `Unscented Kalman-Bucy filter`.

Beta — Beta coefficient
2 (default) | nonnegative scalar

Coefficient related to the distribution. The block uses this parameter in its implementation of the unscented Kalman filter and the Unscented Kalman-Bucy filter.

Dependencies

To enable this parameter, set **Filter type** to `Unscented Kalman filter` or `Unscented Kalman-Bucy filter`.

Kappa — Kappa coefficient
`0` (default) | nonnegative scalar between `0` and `3`

Coefficient that controls the spread of the sigma points. The block uses this parameter in its implementation of the equations for the unscented Kalman filter and the Unscented Kalman-Bucy filter.

Dependencies

To enable this parameter, set **Filter type** to `Unscented Kalman filter` or `Unscented Kalman-Bucy filter`.

Covariance of the process noise, Q — Covariance of the process noise
`[1e-6 0; 0 1e-6]` (default) | matrix of scalars

2-by-2 covariance matrix of the noise in the states.

Covariance of the measurement noise, R — Covariance of the measurement noise
`0.1` (default) | scalar

Covariance matrix of the noise in the measurements.

Initial state error covariance, P0 — Initial state error covariance
`[1e-5 0; 0 1]` (default) | matrix of scalars

2-by-2 covariance matrix of the initial state error. This parameter defines the deviation in the initialization of the state.

Sample time (-1 for inherited) — Block sample time
`-1` (default) | positive scalar

Time between consecutive block executions. During execution, the block produces outputs and, if appropriate, updates its internal state. For more information, see “What Is Sample Time?” and “Specify Sample Time”.

For inherited discrete-time operation, specify this parameter as `-1`. For discrete-time operation, specify this parameter as a positive integer. For continuous-time operation, specify this parameter as `0`.

If this block is in a masked subsystem or a variant subsystem that allows you to switch between continuous operation and discrete operation, promote the sample time parameter. Promoting the sample time parameter ensures correct switching between the continuous and discrete implementations of the block. For more information, see “Promote Block Parameters on a Mask”.

Dependencies

To enable this parameter, set **Filter type** to `Extended Kalman filter` or `Unscented Kalman filter`.

System Model

Vector of state-of-charge values, SOC (-) — SOC breakpoints
`[0, .1, .25, .5, .75, .9, 1]` (default) | vector of nonnegative scalars between `0` and `1`

Vector of the state-of-charge breakpoints defining the points at which you specify lookup data. The entries of this vector must be in strictly ascending order. The block calculates the state-of-charge value with respect to the nominal battery capacity specified in the **Cell capacity, AH (A*Hr)** parameter. The SOC is the ratio of the available battery charge $q_{battery}$ and the nominal battery capacity $q_{nom}(T,n)$. You must make sure that, for each temperature, an SOC of 1 represents the respective battery charge capacity specified in the **Cell capacity, AH (A*Hr)** parameter when you model a fresh battery with a number of cycles N equal to 1 and $\delta_{AH}(n = 1, T_{fade}) = 0$.

$$SOC = \frac{q_{battery}}{q_{nom}(T,n)}$$
$$for\, N = 1\; and\, \delta_{AH}(n, T_{fade}) = 0,\;\; q_{nom}(T,n) = AH\,.$$

Vector of temperatures, T — *T* breakpoints
`[278, 293, 313]` (default) | vector of positive scalars

Vector of temperature breakpoints defining the points at which you specify lookup data. This vector must be strictly ascending and greater than `0 K`. The physical unit of this parameter must be the same as the physical unit of the **CellTemperature** input port.

Terminal resistance, R0(SOC,T), (ohm) — R0 lookup table with temperature breakpoint
`[.0117, .0085, .009; .011, .0085, .009; .0114, .0087, .0092; .0107, .0082, .0088; .0107, .0083, .0091; .0113, .0085, .0089; .0116, .0085, .0089]` (default) | matrix of positive scalars

Lookup data for series resistance of the battery at the specified SOC and temperature breakpoints.

First polarization resistance, R1(SOC,T), (ohm) — First RC resistance at temperature breakpoints
`[.0109, .0029, .0013; .0069, .0024, .0012; .0047, .0026, .0013; .0034, .0016, .001; .0033, .0023, .0014; .0033, .0018, .0011; .0028, .0017, .0011]` (default) | matrix of positive scalars

Lookup data, in ohm, for the first parallel RC resistance at the specified SOC and temperature breakpoints. Matrix of resistance values, in ohm. The number of rows of this matrix is equal to the size of the **Vector of state-of-charge values, SOC (-)** parameter. The number of columns of this matrix is equal to the size of the **Vector of temperatures, T** parameter.

First time constant, tau1(SOC,T), (s) — First RC time constant at temperature breakpoints
`[20, 36, 39; 31, 45, 39; 109, 105, 61; 36, 29, 26; 59, 77, 67; 40, 33, 29; 25, 39, 33]` (default) | matrix of positive scalars

Lookup data, in second, for the first parallel RC time constant at the specified SOC and temperature breakpoints. The number of rows of this matrix is equal to the size of the **Vector of state-of-charge values, SOC (-)** parameter. The number of columns of this matrix is equal to the size of the **Vector of temperatures, T** parameter.

No-load voltage, V0(SOC,T), (V) — V_0 lookup table
`[3.49, 3.5, 3.51; 3.55, 3.57, 3.56; 3.62, 3.63, 3.64; 3.71, 3.71, 3.72; 3.91, 3.93, 3.94; 4.07, 4.08, 4.08; 4.19, 4.19, 4.19]` (default) | matrix of nonnegative scalars

Lookup data, in volt, for open-circuit voltages across the fundamental battery model at the specified SOC. The number of rows of this matrix is equal to the size of the **Vector of state-of-charge values, SOC (-)** parameter. The number of columns of this matrix is equal to the size of the **Vector of temperatures, T** parameter.

Cell capacity, AH (A*Hr) — Battery capacity when fully charged
27 (default) | nonnegative scalar

Cell capacity of the battery, in A*Hr . The block calculates the state of charge by dividing the accumulated charge by this value. The block calculates the accumulated charge by integrating the battery current.

Version History

Introduced in R2022b

References

[1] Grewal, Mohinder S., and Angus P. Andrews. *Kalman Filtering: Theory and Practice Using MATLAB.* New York: Wiley Intersci., 2001.

[2] Van der Merwe, R., and E. A. Wan, *The Square-Root Unscented Kalman Filter for State and Parameter-Estimation.* 2001 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings (Cat. No.01CH37221), vol. 6, IEEE, 2001, pp. 3461–64.

[3] Sarkka, Simo *On Unscented Kalman Filtering for State Estimation of Continuous-Time Nonlinear Systems.* IEEE Transactions on Automatic Control, vol. 52, no. 9, pp. 1631-1641, Sept. 2007, doi: 10.1109/TAC.2007.904453.

[4] Huria, Tarun, et al. *Simplified Extended Kalman Filter Observer for SOC Estimation of Commercial Power-Oriented LFP Lithium Battery Cells.*2013, pp. 2013-01–1544.

Extended Capabilities

C/C++ Code Generation

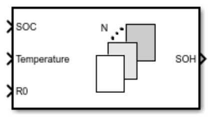
Generate C and C++ code using Simulink® Coder™.

See Also

SOC Estimator (Adaptive Kalman Filter) | SOC Estimator (Coulomb Counting) | SOC Estimator (Coulomb Counting, Variable Capacity) | SOH Estimator

SOH Estimator

State-of-health estimator



Libraries:
Simscape / Battery / BMS / Estimators

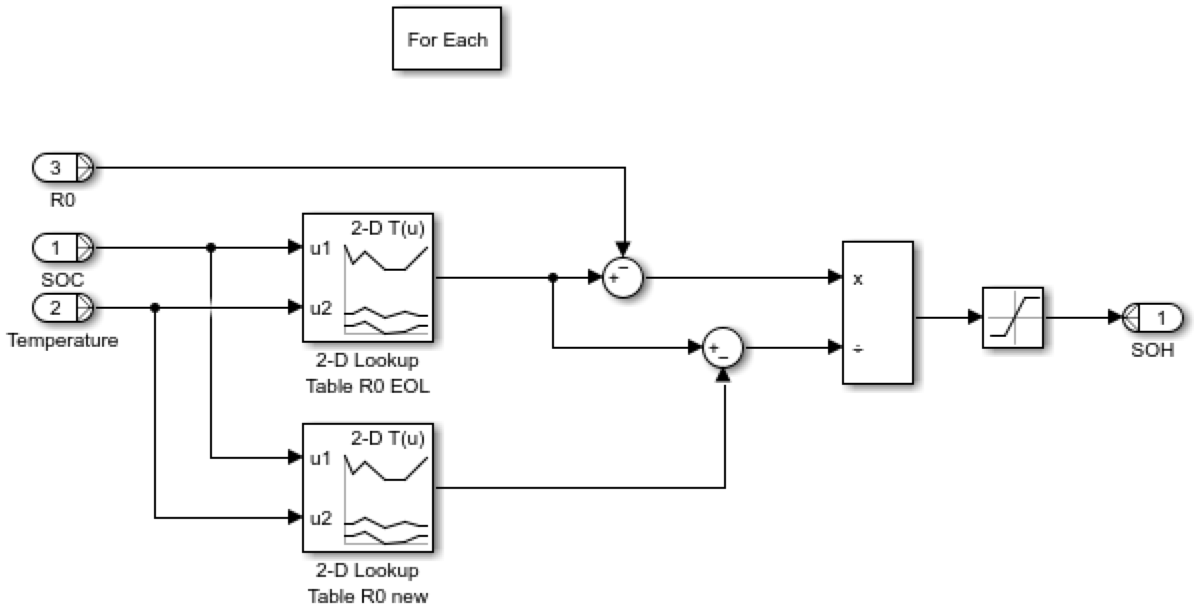
Description

This block implements an estimator that calculates the state-of-health (SOH) of a battery. The SOH is an indicator that reflects the condition of a battery relative to its ideal conditions. Health variations imply that the capacity and power of the battery fade over time.

This block supports both single-precision and double-precision floating-point simulation.

Note To enable single-precision floating-point simulation, the data type of all inputs and parameters must be `single`.

This diagram shows the structure of the block:



Equations

The terminal resistance R_0 is an important indicator of the aging of a battery. You can estimate the terminal resistance by using the SOC Estimator (Adaptive Kalman Filter) block. The SOH Estimator block computes the SOH as a function of the terminal resistance R_0:

$$SOH = \frac{R_{EOL} - R_0}{R_{EOL} - R_{0,\,new}}$$

where R_{EOL} is the end-of-life resistance and $R_{0,new}$ is the terminal resistance when the battery is new.

Ports

Input

SOC — Battery state-of-charge
scalar | vector

State-of-charge of the battery, specified as a scalar or vector of entries in the range [0, 1]. The size of this input port must be equal to the size of the **Temperature** and **R0** input ports.

Temperature — Cell temperature
scalar | vector

Temperature of a battery cell, specified as a scalar for a single cell or a vector for multiple cells. The size of this input port must be equal to the size of the **SOC** and **R0** input ports.

R0 — Terminal resistance
scalar | vector

Terminal resistance, specified as a scalar for a single cell or a vector for multiple cells. The size of this input port must be equal to the size of the **SOC** and **Temperature** input ports.

Output

SOH — Battery state-of-health
scalar | vector

State-of-health of the battery, returned as a scalar or vector of entries in the range [0, 1]. The size of this output port is equal to the size of the vectors at the input ports.

Parameters

Vector of state-of-charge values, SOC (-) — SOC breakpoints
`[0, .1, .25, .5, .75, .9, 1]` (default) | vector of nonnegative scalars between `0` and `1`

Vector of SOC breakpoints defining the points at which you specify lookup data. This vector must be strictly ascending.

Vector of temperatures, T — Temperature breakpoints
`[278, 293, 313]` (default) | vector of positive scalars

Vector of temperature breakpoints defining the points at which you specify lookup data. This vector must be strictly ascending and greater than `0 K`. The physical unit of this parameter must be the same as the physical unit of the **Temperature** input port.

Terminal resistance when battery cell is new, R0new(SOC,T), (ohm) — Terminal resistance when battery cell is new
`[.0117, .0085, .009; .011, .0085, .009; .0114, .0087, .0092; .0107, .0082, .0088; .0107, .0083, .0091; .0113, .0085, .0089; .0116, .0085, .0089]` (default) | matrix of nonnegative scalars

Terminal resistance when the battery cell is new, in ohm. The number of rows of this matrix is equal to the size of the **Vector of state-of-charge values, SOC (-)** parameter. The number of columns of this matrix is equal to the size of the **Vector of temperatures, T** parameter.

Terminal resistance at end of life, R0eol(SOC,T), (ohm) — Terminal resistance at end of life
`[.0234,.017,.018;.022,.017,.018;.0228,.0174,.0184;.0214,.0164,.0176;.0214,.0166,.0182;.0226,.017,.0178;.0232,.017,.0178]` (default) | matrix of nonnegative scalar

Terminal resistance when the battery is at the end of its life, in ohm. The number of rows of this matrix is equal to the size of the **Vector of state-of-charge values, SOC (-)** parameter. The number of columns of this matrix is equal to the size of the **Vector of temperatures, T** parameter.

Version History

Introduced in R2022b

References

[1] Noura, Nassim, Loïc Boulon, and Samir Jemeï. “A Review of Battery State of Health Estimation Methods: Hybrid Electric Vehicle Challenges.” *World Electric Vehicle Journal* 11, no. 4 (October 16, 2020): 66. https://doi.org/10.3390/wevj11040066.

Extended Capabilities

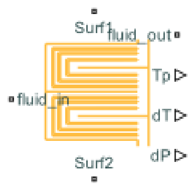
C/C++ Code Generation
Generate C and C++ code using Simulink® Coder™.

See Also

SOC Estimator (Adaptive Kalman Filter) | SOC Estimator (Coulomb Counting) | SOC Estimator (Coulomb Counting, Variable Capacity) | SOC Estimator (Kalman Filter)

U-shaped Channels

Cooling plate with flat channels



Libraries:
Simscape / Battery / Thermal

Description

This block models a battery cooling plate with flat channels. Use the `buildBattery` function to create a Simscape model of a battery and connect it to one or both sides of the cooling plate. You can cool any of the four edges of the cooling plate. The **fluid_in** and **fluid_out** ports are thermal fluid ports. The output ports are the plate temperature, **Tp**, the fluid pressure drop, **dP**, and the fluid temperature change, **dT**.

Ports

Output

Tp — Plate temperature
physical signal

Temperature of the cooling plate.

dT — Fluid temperature change
physical signal

Temperature change of the fluid.

dP — Fluid pressure drop
physical signal

Pressure drop of the fluid.

Conserving

fluid_in — Fluid in cooling plate
thermal fluid

Thermal fluid conserving port associated with the fluid that comes in the cooling plate.

fluid_out — Fluid out of cooling plate
thermal fluid

Thermal fluid conserving port associated with the fluid that comes out of the cooling plate.

Surf1 — Surface 1 of cooling plate
thermal

Thermal conserving port associated with the surface 1 of the cooling plate. This port connects the array of thermal nodes between the cooling plate and battery pack or module.

Surf2 — Surface 2 of cooling plate
thermal

Thermal conserving port associated with the surface 2 of the cooling plate. This port connects the array of thermal nodes between the cooling plate and battery pack or module.

Parameters

Interface

Battery Connectivity — Connectivity of battery
`Single sided` (default) | `Double sided`

Option to choose the connectivity of the battery.

Number of battery thermal nodes (surface 1) — Number of battery thermal nodes on surface 1
`1` (default)

Number of battery thermal nodes on surface 1 of the cooling plate.

Dimension of battery thermal nodes (surface 1) — Dimension of battery thermal nodes on surface 1
`ones(1, 2)` (default)

Dimension of battery thermal nodes on surface 1 of the cooling plate.

Coordinates of battery thermal nodes (surface 1) — Coordinates of battery thermal nodes on surface 1
`ones(1, 2)` (default)

Coordinates of battery thermal nodes on surface 1 of the cooling plate.

Number of battery thermal nodes (surface 2) — Number of battery thermal nodes on surface 2
`1` (default)

Number of battery thermal nodes on surface 2 of the cooling plate.

Dependencies

To enable this parameter, set **Battery Connectivity** to `Double sided`.

Dimension of battery thermal nodes (surface 2) — Dimension of battery thermal nodes on surface 2
`ones(1, 2)` (default)

Dimension of battery thermal nodes on surface 2 of the cooling plate.

Dependencies

To enable this parameter, set **Battery Connectivity** to `Double sided`.

Coordinates of battery thermal nodes (surface 2) — Coordinates of battery thermal nodes on surface 2
`ones(1, 2)` (default)

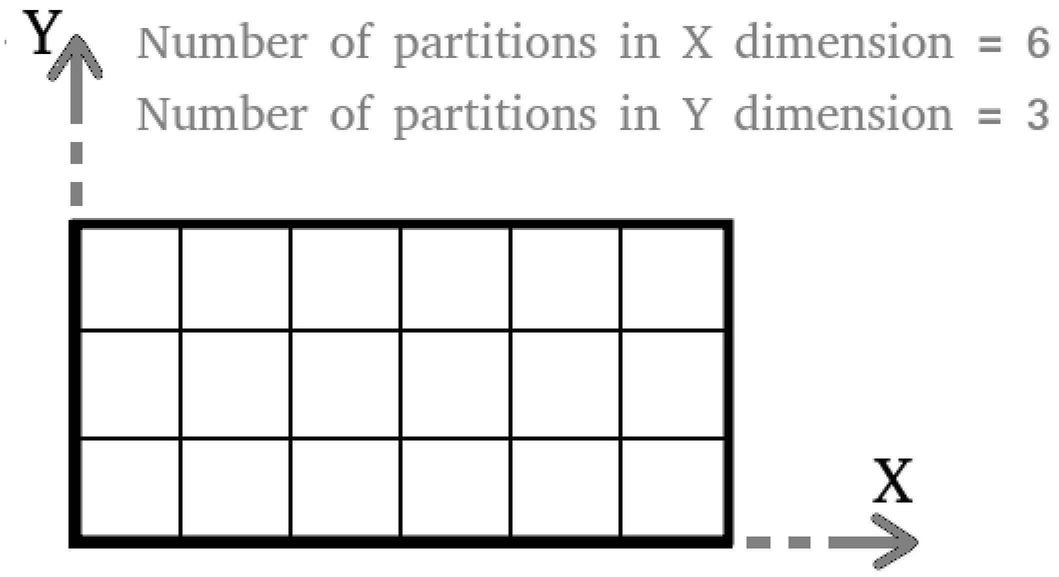
Coordinates of battery thermal nodes on surface 2 of the cooling plate.

Dependencies

To enable this parameter, set **Battery Connectivity** to `Double sided`.

Number of partitions in X dimension for the cooling plate — Number of partitions in X dimension for the cooling plate
2 (default)

Number of partitions in X dimension for the cooling plate.


Y
Number of partitions in X dimension = 6
Number of partitions in Y dimension = 3
X

Number of partitions in Y dimension for the cooling plate — Number of partitions in Y dimension for the cooling plate
5 (default)

Number of partitions in Y dimension for the cooling plate.

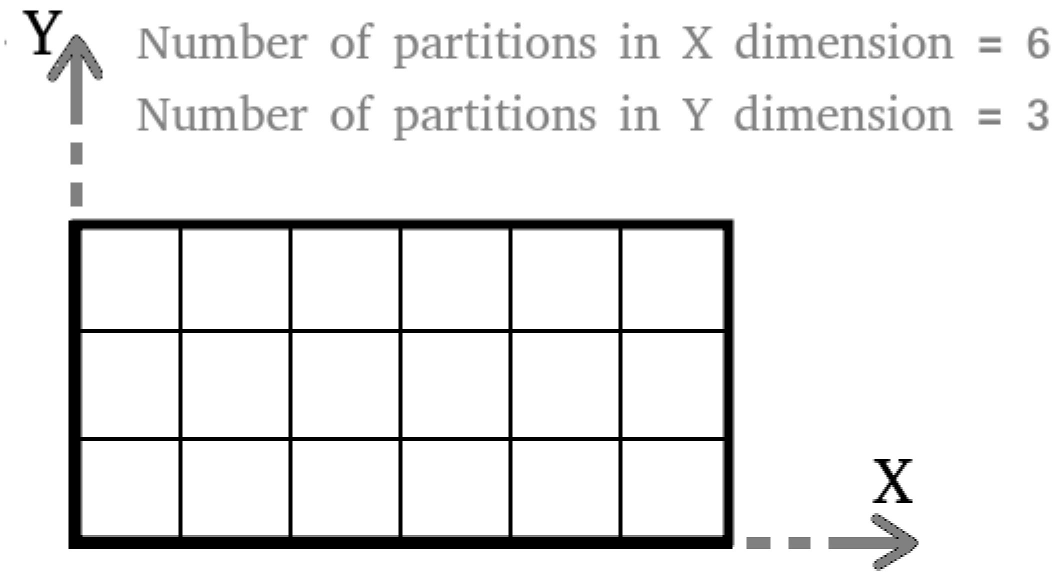

Y
Number of partitions in X dimension = 6
Number of partitions in Y dimension = 3
X

Plate Material

Thickness of cooling plate material — Thickness of material of cooling plate
`2e-3 m` (default)

Thickness of the material of the cooling plate.

Thermal conductivity of cooling plate material — Thermal conductivity of material of cooling plate
`20 W/(K*m)` (default)

Thermal conductivity of the material of the cooling plate.

Density of cooling plate material — Density of material of cooling plate
`2500 kg/m^3` (default)

Density of the material of the cooling plate.

Specific heat of cooling plate material — Specific heat of material of cooling plate
`447 J/(K*kg` (default)

Specific heat of the material of the cooling plate.

Initial temperature of the cooling plate and the coolant fluid — Initial temperature of cooling plate and coolant fluid
`300 K` (default)

Initial temperature of the cooling plate and of the coolant fluid.

Fluid Properties

Reynolds number upper limit for laminar flow — Upper limit of Reynolds number for laminar flow
`2000` (default)

Upper limit of Reynolds number for laminar flow.

Reynolds number lower limit for turbulent flow — Lower limit of Reynolds number for turbulent flow
`4000` (default)

Lower limit of Reynolds number for turbulent flow.

Nusselt number for laminar flow heat transfer — Nusselt number for laminar flow heat transfer
`3.66` (default)

Nusselt number for laminar flow heat transfer.

Aggregate equivalent length of local resistances — Aggregate equivalent length of local resistances
`1e-3 m` (default)

Aggregate equivalent length of local resistances.

Laminar friction constant for Darcy friction factor — Laminar friction constant for Darcy friction factor
`64` (default)

Laminar friction constant for the Darcy friction factor.

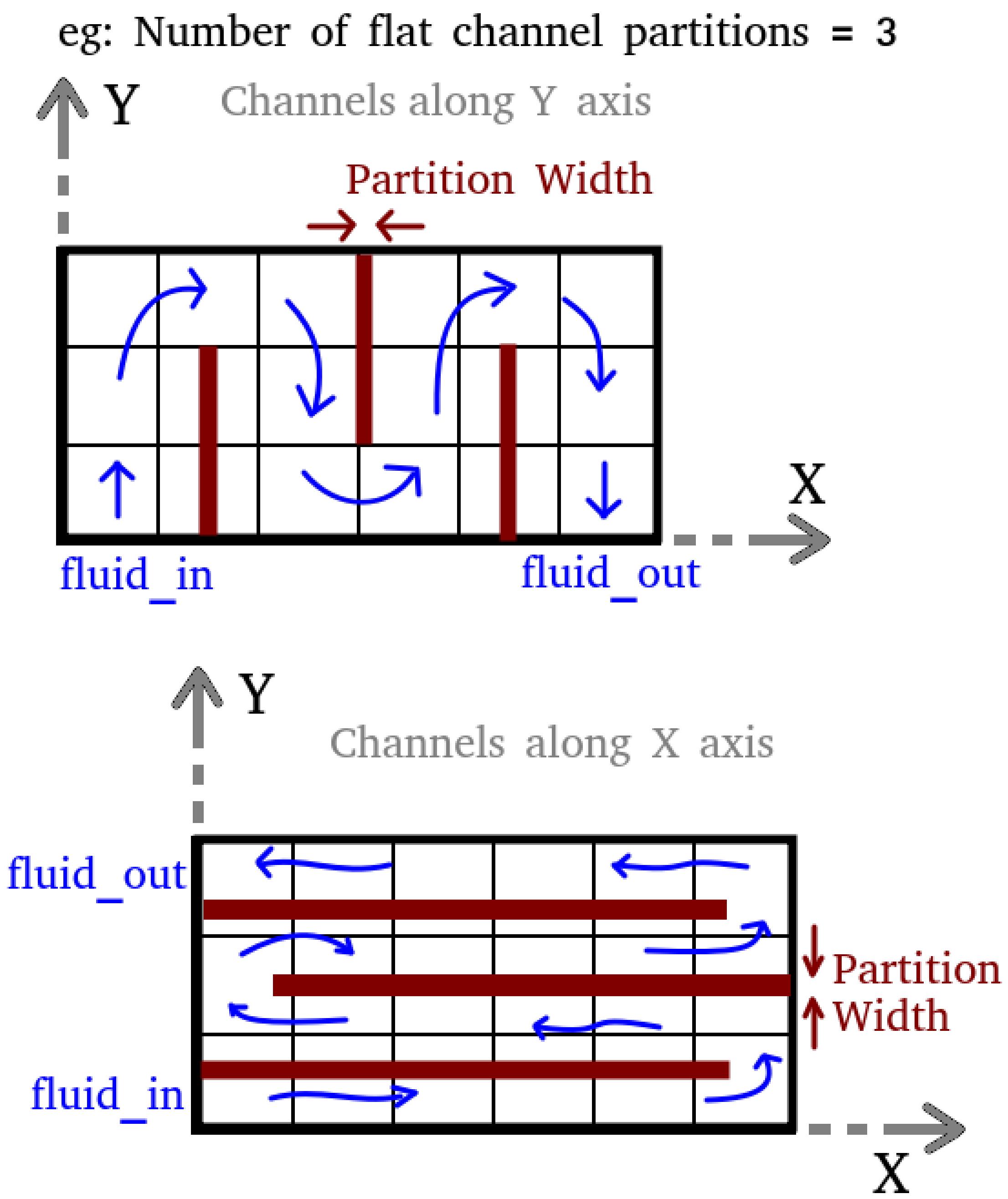
Design

Number of flat channel partitions — Number of partitions of flat channels
`1` (default)

Number of the partitions of the flat channels.

Select channel orientation direction — Option for direction of channel orientation
`Channel along X axis` (default) | `Channel along Y axis`

Option to choose the axis direction of the channel orientation.



Channel thickness — Thickness of channel
5e-03 m (default)

Thickness of the coolant channel.

Partition width — Width of partition
5e-03 m (default)

Width of the partition.

Coolant channel roughness — Roughness of coolant channel
`1.5e-05` m (default)

Roughness of the coolant channel.

Version History

Introduced in R2023a

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using Simulink® Coder™.

See Also

Simscape Blocks
arrayOfThermalNodesConnector | Edge Cooling | Parallel Channels

Objects
`Cell` | `ParallelAssembly` | `Module` | `ModuleAssembly` | `Pack`

Functions
`buildBattery`

Topics
"Connect Cooling Plate to Battery Blocks"

2

Functions

applyCellDataFromPart

Apply data from selected parameterization to Cell object

Syntax

```
applyCellDataFromPart(batteryCell)
```

Description

`applyCellDataFromPart(batteryCell)` updates the `Geometry`, `Mass`, `Capacity`, and `Energy` properties of the `Cell` object `batteryCell`. The applied data depends on the parameterization you specified in the `ParameterizationManufacturer` and `ParameterizationPartNumber` properties of the `Cell` object.

Examples

Update Cell Properties From Selected Parameterization

Create a `Cell` object.

```
import simscape.battery.builder.*
battCell = Cell(Geometry = CylindricalGeometry);
```

Choose the parameterization. Set the `ParameterizationManufacturer` and `ParameterizationPartNumber` properties.

```
battCell.ParameterizationManufacturer = "A123";
battCell.ParameterizationPartNumber = "ALM12V7";
```

Apply the parameterization data by using the `applyCellDataFromPart` function.

```
battCell.applyCellDataFromPart;
```

Input Arguments

`batteryCell` — Battery cell
`Cell` object (default)

Cell to update with the data from the selected parameterization, specified as a `Cell` object.

Version History

Introduced in R2023a

See Also

Apps
Battery Builder

Objects
`Cell`

Topics
"Build Simple Model of Battery Pack in MATLAB and Simscape"
"Build Detailed Model of Battery Pack From Pouch Cells"

BatteryChart

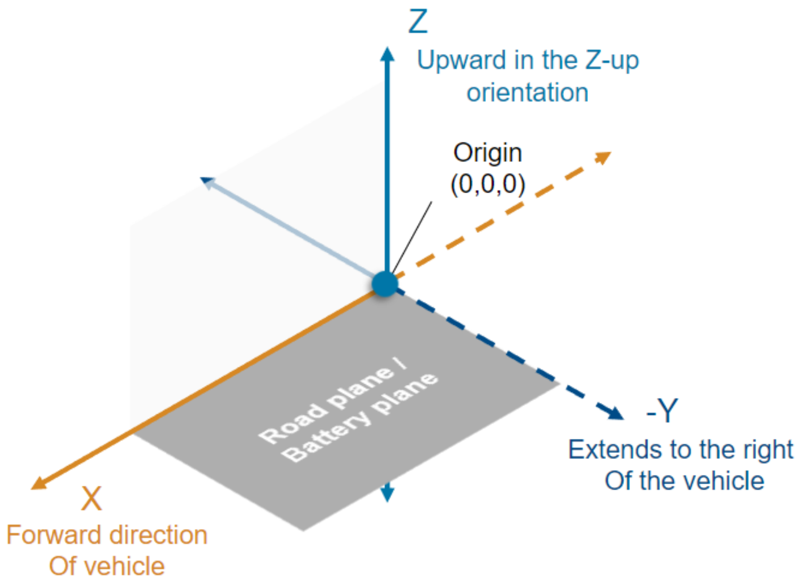
Visualize battery objects

Description

Use `BatteryChart` to construct a battery chart for visualizing a Simscape Battery™ object.

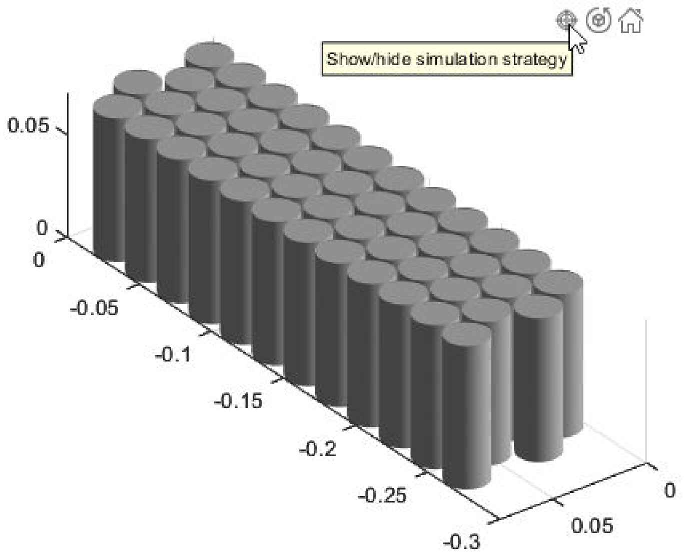
Note The `BatteryChart` object does not store or modify any of the properties of the objects it displays.

The `BatteryChart` object displays the battery object according to the global coordinate system for batteries:

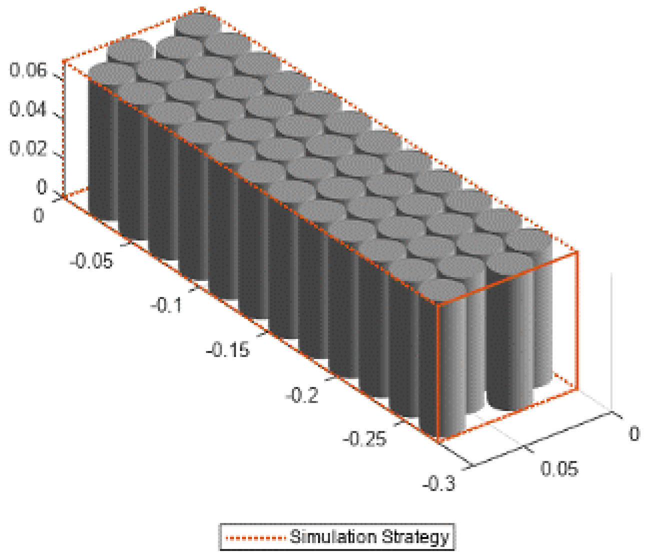

Z
Upward in the Z-up orientation
Origin (0,0,0)
Road plane / Battery plane
-Y
Extends to the right Of the vehicle
X
Forward direction Of vehicle

To enable these labels in your chart, use the `setDefaultLabels` function. For more information about setting labels, see “Set Labels for BatteryChart Object” on page 2-11.

The `BatteryChart` object also allows you to check the current simulation strategy and model resolution of the battery component. To visualize the simulation strategy in the chart, set the `SimulationStrategyVisible` property to `"on"` or click on the "Show/hide simulation strategy" button on the top-right corner of the chart.

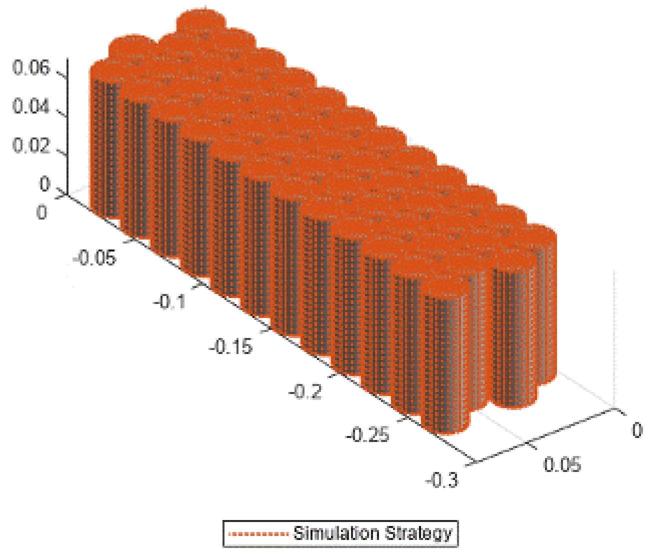


For example, if you set the `ModelResolution` property of a `ParallelAssembly` object to `"Lumped"`, the object automatically scales the electrical parameters of the cell model blocks by using the `NumParallelCells` property. This figure shows the simulation strategy for a lumped parallel assembly:



Only one cell model block represents all the cell components inside the orange box.

If you set the `ModelResolution` property of a `ParallelAssembly` object to `"Grouped"`, a number of cell model blocks equal to the value of the `NumParallelCells` property represents each cell component instead.



Creation

Syntax

`import simscape.battery.builder.*; chart = BatteryChart(Name=Value)`

Description

`import simscape.battery.builder.*; chart = BatteryChart(Name=Value)` creates a `BatteryChart` object that visualizes the battery object specified in the `Battery` property, in the container specified in the `Parent` property. This syntax also sets further “Properties” on page 2-6 using one or more name-value arguments

Properties

`Parent` — Container for battery visualization
`Figure` object (default)

Container for the visualization of the battery, specified as a `Figure` object.

`Battery` — Battery object to visualize
`Cell` object | `ParallelAssembly` object | `Module` object | `ModuleAssembly` object | `Pack` object

Battery object to visualize, specified as a `Cell`, `ParallelAssembly`, `Module`, `ModuleAssembly`, or `Pack` object.

`AxesVisible` — Option to visualize axes of chart
`"on"` (default) | `"off"`

Option to visualize the axes of the chart, specified as `"on"` or `"off"`.

`AxesXDir` — Direction of increasing values along *x*-axis
`"reverse"` (default) | `"normal"`

Direction of increasing values along the *x*-axis, specified as:

- `"normal"` — Values increase outward from the center of the chart.
- `"reverse"` — Values decrease outward from the center of the chart.

`AxesYDir` — Direction of increasing values along *y*-axis
`"normal"` (default) | `"reverse"`

Direction of increasing values along the *y*-axis, specified as:

- `"normal"` — Values increase outward from the center of the chart.
- `"reverse"` — Values decrease outward from the center of the chart.

`AxesZDir` — Direction of increasing values along *z*-axis
`"normal"` (default) | `"reverse"`

Direction of increasing values along the *z*-axis, specified as:

- `"normal"` — Values increase outward from the center of the chart.
- `"reverse"` — Values decrease outward from the center of the chart.

`SimulationStrategyVisible` — Option to visualize axes of chart
`"on"` (default) | `"off"`

Visibility of the simulation strategy highlighted on the chart, specified as `"on"` or `"off"`.

`SimulationStrategyLineColor` — Color of line for simulation strategy
`[0.8500 0.3250 0.0980]` (default) | RGB triplet | hexadecimal color code | `'r'` | `'g'` | `'b'` | ...

Color of the line for the simulation strategy highlighted on the chart, specified as an RGB triplet, a hexadecimal color code, a color name, or a short name.

For a custom color, specify an RGB triplet or a hexadecimal color code.

- An RGB triplet is a three-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range `[0,1]`; for example, `[0.4 0.6 0.7]`.
- A hexadecimal color code is a character vector or a string scalar that starts with a hash symbol (`#`) followed by three or six hexadecimal digits, which can range from `0` to `F`. The values are not case sensitive. Thus, the color codes `'#FF8800'`, `'#ff8800'`, `'#F80'`, and `'#f80'` are equivalent.

Alternatively, you can specify some common colors by name. This table lists the named color options, the equivalent RGB triplets, and hexadecimal color codes.

Color Name	Short Name	RGB Triplet	Hexadecimal Color Code	Appearance
`'red'`	`'r'`	`[1 0 0]`	`'#FF0000'`	🟥
`'green'`	`'g'`	`[0 1 0]`	`'#00FF00'`	🟩
`'blue'`	`'b'`	`[0 0 1]`	`'#0000FF'`	🟦
`'cyan'`	`'c'`	`[0 1 1]`	`'#00FFFF'`	🟦
`'magenta'`	`'m'`	`[1 0 1]`	`'#FF00FF'`	🟪

Color Name	Short Name	RGB Triplet	Hexadecimal Color Code	Appearance
`'yellow'`	`'y'`	`[1 1 0]`	`'#FFFF00'`	🟨
`'black'`	`'k'`	`[0 0 0]`	`'#000000'`	⬛
`'white'`	`'w'`	`[1 1 1]`	`'#FFFFFF'`	⬜
`'none'`	Not applicable	Not applicable	Not applicable	No color

Here are the RGB triplets and hexadecimal color codes for the default colors.

RGB Triplet	Hexadecimal Color Code	Appearance
`[0 0.4470 0.7410]`	`'#0072BD'`	🟦
`[0.8500 0.3250 0.0980]`	`'#D95319'`	🟧
`[0.9290 0.6940 0.1250]`	`'#EDB120'`	🟨
`[0.4940 0.1840 0.5560]`	`'#7E2F8E'`	🟪
`[0.4660 0.6740 0.1880]`	`'#77AC30'`	🟩
`[0.3010 0.7450 0.9330]`	`'#4DBEEE'`	🟦
`[0.6350 0.0780 0.1840]`	`'#A2142F'`	🟥

`SimulationStrategyLineStyle` — Style of line for simulation strategy
`":"` (default) | character vector | string

Style of the line for the simulation strategy highlighted on the chart, specified as a character vector or string containing symbols.

Example: `'--'` is a dashed line

Line Style	Description	Resulting Line
`'-'`	Solid line	————————
`'--'`	Dashed line	- — — — —
`':'`	Dotted line	....
`'-.'`	Dash-dotted line	—·—·—·—·-

`SimulationStrategyLineWidth` — Width of line for simulation strategy
`1.5` (default) | positive value

Width of the line for the simulation strategy highlighted on the chart, specified as a positive value in points, where 1 point = 1/72 of an inch.

The line width cannot be thinner than the width of a pixel. If you set the line width to a value that is less than the width of a pixel on your system, the line displays as one pixel wide.

`LightColor` — Color of light
`[1 1 1]` (default) | RGB triplet | hexadecimal color code | `'r'` | `'g'` | `'b'` | ...

Color of light, specified as an RGB triplet, a hexadecimal color code, a color name, or a short name. The default RGB triplet of `[1 1 1]` corresponds to white.

RGB triplets and hexadecimal color codes are useful for specifying custom colors.

- An RGB triplet is a three-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range `[0,1]`; for example, `[0.4 0.6 0.7]`.
- A hexadecimal color code is a character vector or a string scalar that starts with a hash symbol (`#`) followed by three or six hexadecimal digits, which can range from `0` to `F`. The values are not case sensitive. Thus, the color codes `'#FF8800'`, `'#ff8800'`, `'#F80'`, and `'#f80'` are equivalent.

Alternatively, you can specify some common colors by name. This table lists the named color options, the equivalent RGB triplets, and hexadecimal color codes.

Color Name	Short Name	RGB Triplet	Hexadecimal Color Code	Appearance
`'red'`	`'r'`	`[1 0 0]`	`'#FF0000'`	🟥
`'green'`	`'g'`	`[0 1 0]`	`'#00FF00'`	🟩
`'blue'`	`'b'`	`[0 0 1]`	`'#0000FF'`	🟦
`'cyan'`	`'c'`	`[0 1 1]`	`'#00FFFF'`	🟦
`'magenta'`	`'m'`	`[1 0 1]`	`'#FF00FF'`	🟪
`'yellow'`	`'y'`	`[1 1 0]`	`'#FFFF00'`	🟨
`'black'`	`'k'`	`[0 0 0]`	`'#000000'`	⬛
`'white'`	`'w'`	`[1 1 1]`	`'#FFFFFF'`	⬜

Here are the RGB triplets and hexadecimal color codes for the default colors.

RGB Triplet	Hexadecimal Color Code	Appearance
`[0 0.4470 0.7410]`	`'#0072BD'`	🟦
`[0.8500 0.3250 0.0980]`	`'#D95319'`	🟧
`[0.9290 0.6940 0.1250]`	`'#EDB120'`	🟨
`[0.4940 0.1840 0.5560]`	`'#7E2F8E'`	🟪
`[0.4660 0.6740 0.1880]`	`'#77AC30'`	🟩
`[0.3010 0.7450 0.9330]`	`'#4DBEEE'`	🟦
`[0.6350 0.0780 0.1840]`	`'#A2142F'`	🟥

Example: `'green'`

`LightStyle` — Type of light source
`'infinite'` (default) | `'local'`

Type of light source, specified as:

- `'infinite'` — Place the light at infinity. Use the `LightPosition` property to specify the direction from which the light shines in parallel rays.
- `'local'` — Place the light at the location specified by the `LightPosition` property. The light is a point source that radiates from the location in all directions.

`LightPosition` — Location of light source
`[-1 -1 1]` (default) | three-element vector of the form `[x y z]`

Location of light source, specified as a three-element vector of the form `[x y z]`. Define the vector elements in data units from the axes origin to the (*x*, *y*, *z*) coordinate. The actual location of the light depends on the value of the `LightStyle` property.

Example: `[-40 -4 140]`

`LightVisible` — Visibility of light from light source
`'on'` (default) | on/off logical value

Visibility of light from light source, specified as `'on'` or `'off'`, or as numeric or logical `1` (`true`) or `0` (`false`). A value of `'on'` is equivalent to `true`, and `'off'` is equivalent to `false`. Thus, you can use the value of this property as a logical value.

Examples

Visualize Cell Object with Cylindrical Geometry

Create a cylindrical `Cell` object.

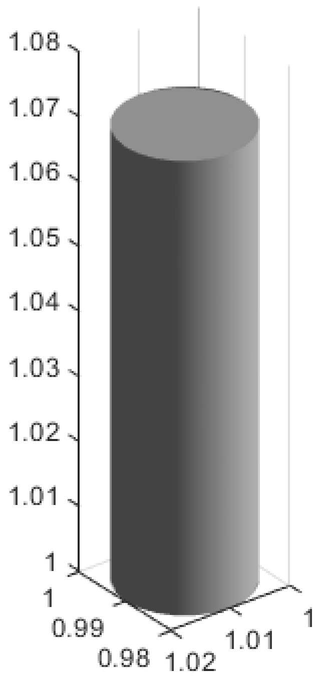
```
batteryCell = Cell(Geometry=CylindricalGeometry)
```

Modify the position of the cell.

```
batteryCell.Position = [1 1 1]
```

Use the `BatteryChart` object to visualize the `Cell` object.

```
cellChart = BatteryChart(Battery=batteryCell)
```



Set Labels for `BatteryChart` Object

To set the labels for a `BatteryChart` object, follow the steps in "Visualize Cell Object with Cylindrical Geometry" on page 2-10 to create a `BatteryChart` object and then use the `setDefaultLabels` method.

```
cellChart.setDefaultLabels
```

Version History

Introduced in R2022b

See Also

Apps
Battery Builder

Objects
`Cell` | `ParallelAssembly` | `Module` | `ModuleAssembly` | `Pack`

Topics
"Build Simple Model of Battery Pack in MATLAB and Simscape"

"Build Detailed Model of Battery Pack From Pouch Cells"

buildBattery

Build custom library blocks from battery objects

Syntax

`buildBattery(battery,Name=Value)`

Description

`buildBattery(battery,Name=Value)` generates one or more custom Simscape library files from the battery object `battery`. You can build a custom Simscape library file from the `ParallelAssembly`, `Module`, `ModuleAssembly`, and `Pack` objects.

Note When you generate a library file for the `ParallelAssembly` and `Module` objects, the `buildBattery` function first generates the corresponding Simscape SSC file and then calls the `ssc_build` function to generate the library file.

The blocks that are generated from `ModuleAssembly` and `Pack` objects are Simulink subsystems that comprise parallel assemblies and module blocks.

Examples

Build Custom Block Libraries from Pack Object

Create a `Pack` object by creating a `Cell`, `ParallelAssembly`, `Module`, and `ModuleAssembly` objects, in this order.

```
import simscape.battery.builder.*;
batteryCell = Cell(Geometry=CylindricalGeometry);
pSet = ParallelAssembly(Cell=batteryCell,NumParallelCells=48,Topology="Hexagonal",Rows=4);
module = Module(ParallelAssembly=pSet,NumSeriesAssemblies=4);
moduleAssembly = ModuleAssembly(Module=repmat(module,1,2));
pack = Pack(ModuleAssembly=repmat(moduleAssembly,1,4),BalancingStrategy="Passive");
```

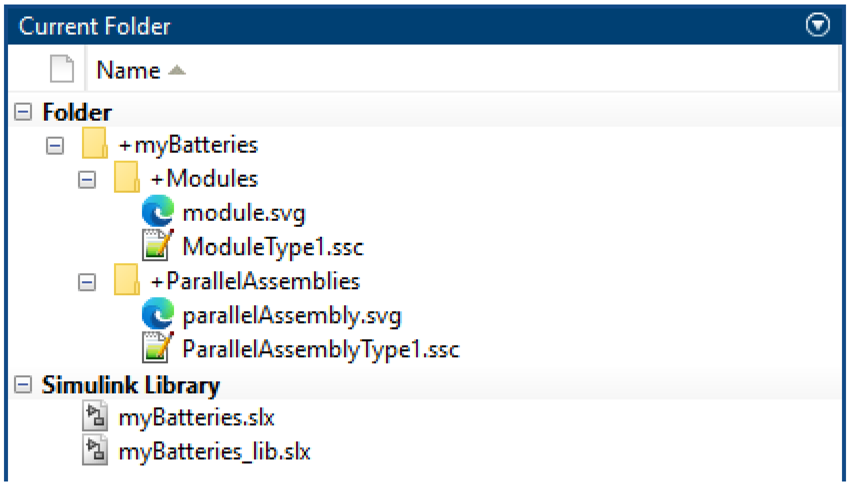
```
pack = 

  Pack with properties:

    ModuleAssembly: [1×4 simscape.battery.builder.ModuleAssembly]
```

Use the `buildBattery` function to build the library file from the `Pack` object. Provide the name-value arguments to the function to give the library a meaningful name and to specify the output folder.

```
buildBattery(pack,LibraryName="myBatteries",Directory="C:\Work\BatteryFolder")
```

```
Generating Simulink library 'myBatteries_lib' in the current directory 'C:\Work\BatteryFolder' .
```

This figure shows the content of the folder after the function finishes generating the library files:

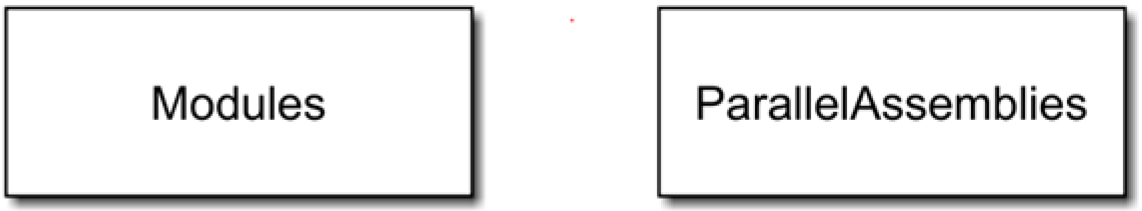

Current Folder

Name

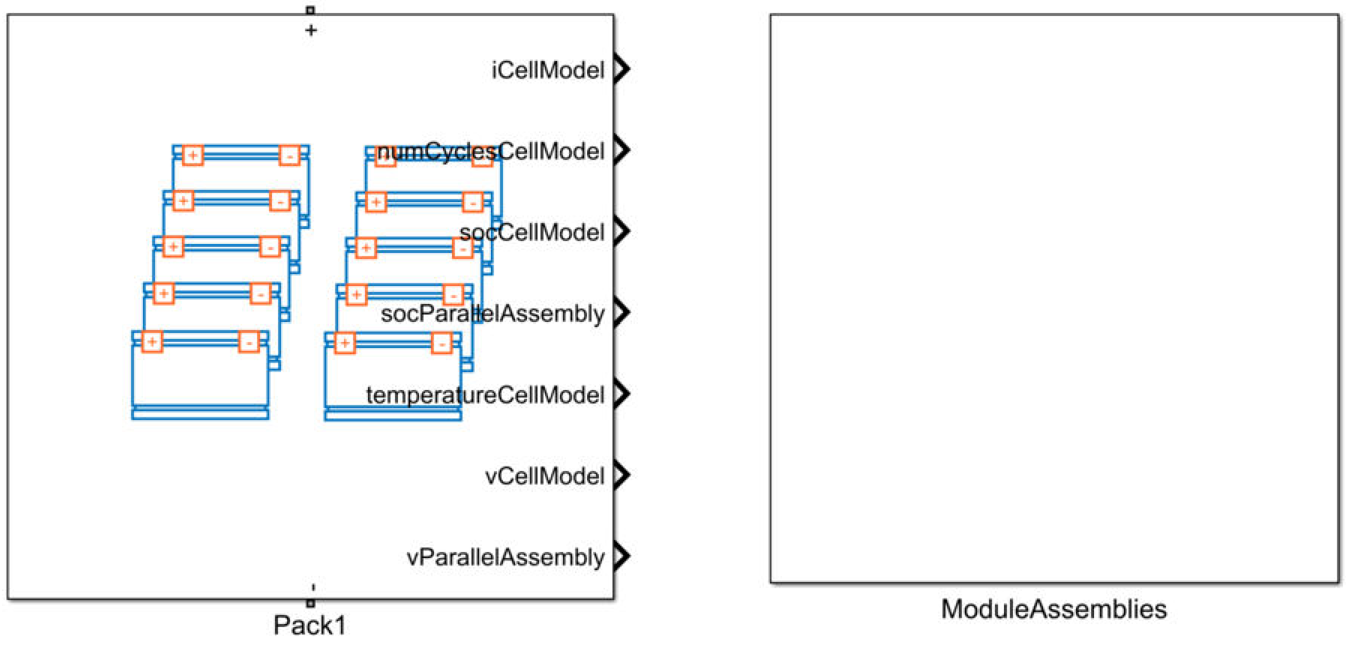
Folder
- +myBatteries
  - +Modules
    - module.svg
    - ModuleType1.ssc
  - +ParallelAssemblies
    - parallelAssembly.svg
    - ParallelAssemblyType1.ssc

Simulink Library
- myBatteries.slx
- myBatteries_lib.slx

Open the generated library `myBatteries_lib` SLX file to access the `ParallelAssembly` and `Module` objects as Simscape blocks.



Open the generated library `myBatteries` SLX file to access the `ModuleAssembly` and `Pack` objects as Simscape subsystems.



Input Arguments

`battery` — Battery object

`ParallelAssembly` object | `Module` object | `ModuleAssembly` object | `Pack` object

Battery object from which to generate the custom Simscape library blocks, specified as a `ParallelAssembly`, `Module`, `ModuleAssembly`, or `Pack` object.

Note When you build a battery object, the `buildBattery` function also builds all of its subcomponents. For example, if you build a `Pack` object, this function generates custom Simscape library blocks for the `ModuleAssembly`, `Module`, and `ParallelAssembly` objects that comprise the pack.

Example: `buildBattery(batteryPack)` builds a block library from a `Pack` battery object named `batteryPack`.

Name-Value Pair Arguments

Example: `buildBattery(batteryPack,LibraryName="myBatteries",Directory="C:\Work\batteryLibFolder")`

`LibraryName` — Name of block library file

character vector (default) | string scalar

Name of the block library file generated from the battery object, specified as a character vector or a string scalar. The name of the library file generated from a `ParallelAssembly` or `Module` object has the suffix `_lib`.

Example: `buildBattery(batteryPack,LibraryName="myBatteries")` builds two libraries named `myBatteries` and `myBatteries_lib` from a `Pack` battery object named `batteryPack`.

`Directory` — Name and location of library folder
character vector (default) | string scalar

Name and location of the folder where you want to store the generated block library file, specified as a character vector or a string scalar.

Example: `buildBattery(batteryPack,Directory="C:\Work\batteryLibFolder")` builds a library from a `Pack` battery object named `batteryPack` in the folder `C:\Work\batteryLibFolder`.

`MaskParameters` — Option to use numeric values or variable names for parameters
`"NumericValues"` (default) | `"VariableNames"`

Option to use default numeric values or variable names for the parameters in each type of module and parallel assembly defined in the `Battery` property. If you set this argument to `"VariableNames"`, the function generates a script with all the run-time parameters required for simulation.

`MaskInitialTargets` — Option to use numeric values or variable names for initial targets
`"NumericValues"` (default) | `"VariableNames"`

Option to use default numeric values or variable names for the initial targets in each type of module and parallel assembly defined in the `Battery` property. If you set this argument to `"VariableNames"`, the function generates a script with all the initial conditions required for simulation.

Version History

Introduced in R2022b

See Also

Apps
Battery Builder

Objects
`Cell` | `ParallelAssembly` | `Module` | `ModuleAssembly` | `Pack`

Simscape Blocks
ParallelAssembly (Generated Block) | Module (Generated Block) | ModuleAssembly (Generated Block) | Pack (Generated Block)

Topics
“Build Simple Model of Battery Pack in MATLAB and Simscape”
“Build Detailed Model of Battery Pack From Pouch Cells”
“Build Model of Hybrid-Cell Battery Pack”
“Manage Battery Run-Time Parameters with Centralized Script”

Cell

Create single electrochemical battery cell

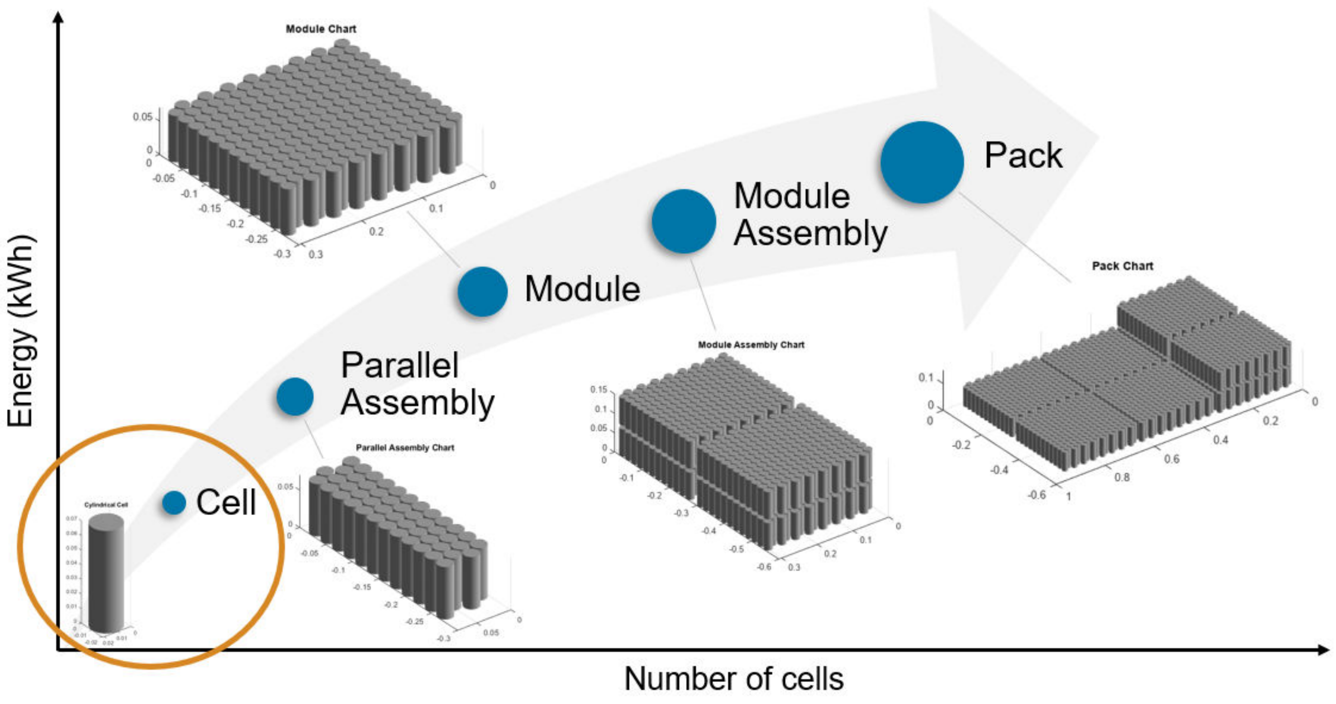
Description

Use `Cell` to construct a battery object that represents a single electrochemical cell. You can use this object as an input to the `ParallelAssembly` object.

Use the properties of this object to describe the basic characteristics of a physical battery cell and link this object to a Simscape cell model block for Simscape simulation. This object only supports the specification of conditional parameters. You can modify the run-time parameters for this model block, such as the battery cell resistance or the battery open-circuit voltage, after you create the model. The Simscape cell model block can be the Battery (Table-Based) block or a custom block with specific requirements. For more information, see the CellModelBlockPath property of the `CellModelBlock` object.

You can also choose to directly parameterize this object with one of the available built-in parameterizations. This pre-parameterization data allows you to set up the object to represent components by specific suppliers. The parameterizations of these batteries match the discharge curves in the manufacturer data sheets.

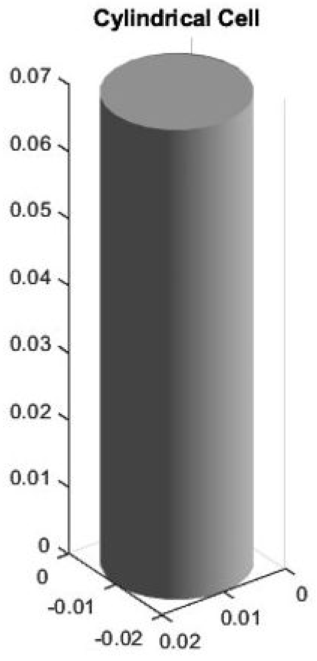
An electrochemical battery cell is the fundamental building block in the manufacturing of larger battery systems. To obtain the required energy and voltage levels, multiple battery cells are typically connected electrically in parallel and/or in series. To meet the battery packaging and space requirements, you can arrange the battery cells in many different topologies or geometrical configurations. To mirror real-world behavior, the Simscape Battery `Cell` object is the foundational element for the creation of a battery pack system model. By using the `ParallelAssembly`, `Module`, `ModuleAssembly`, and `Pack` objects, you can connect the cell model block in parallel and/or in series and scale it up to generate larger battery system models.

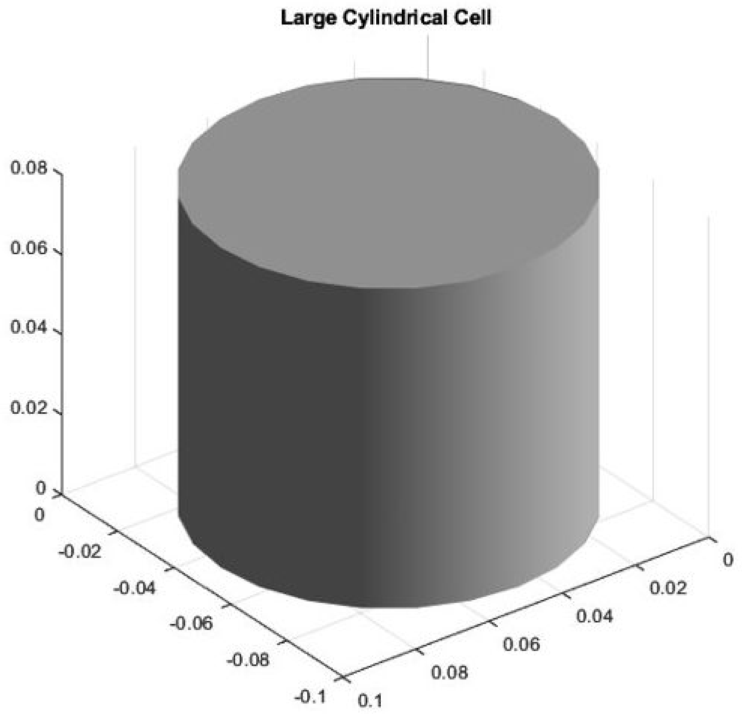


Pack models are required for architecture evaluation in early development stages, software and hardware development, system integration and requirement evaluation, cooling system design, control strategy development hardware-in-the-loop, and many more applications.

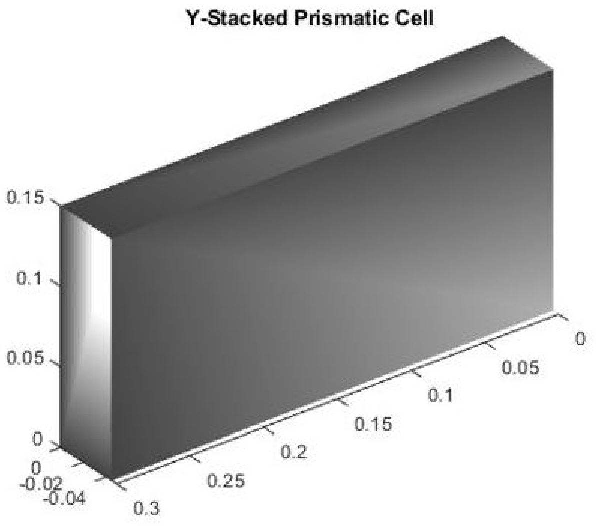
A battery cell is an electrochemical energy storage device that provides electrical energy from stored chemical energy. Cells have three main formats:

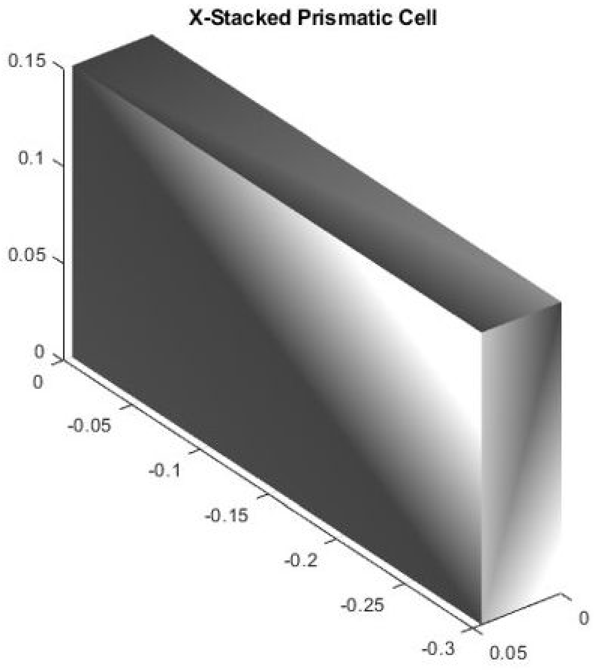
- Cylindrical — The electrode layers are typically rolled into a jelly roll through a winding process. The jelly roll is then secured inside a rigid cylindrical can of stainless steel or another suitable material. Cylindrical cells offer good gravimetric energy density and retention against volume expansion. To model a cylindrical battery cell, specify a `CylindricalGeometry` object as the `Geometry` property.



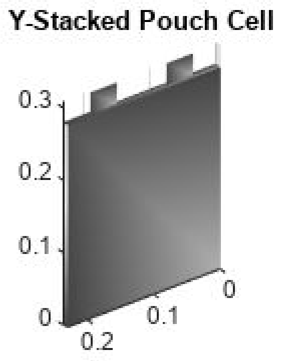


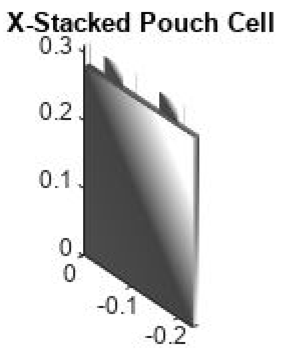
- Prismatic — Rolled or stacked electrode layers pressed in metallic cubic containers. Prismatic cells are usually cooled along their height or *z*-axis in the direction of increasing thermal conductivity. To model a prismatic battery cell, specify a `PrismaticGeometry` object as the `Geometry` property.

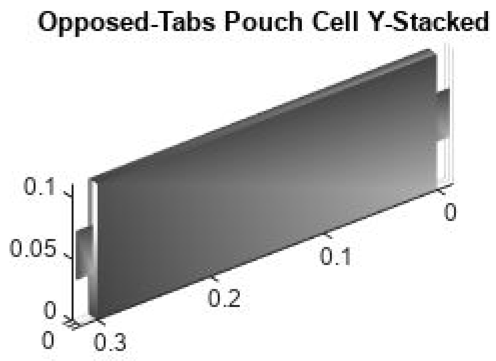


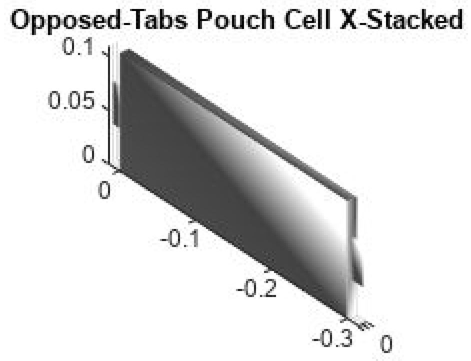


- Pouch — Stacked electrodes and separators pressed in a metallized foil pouch. Packs of pouch cells can fit the available space of a device better than the other packs but they tend to expand and delaminate. To model a pouch battery cell, specify a `PouchGeometry` object as the `Geometry` property.









To visualize a battery `Cell` object and the associated geometry, you must first specify a geometry object by using the `Geometry` property. You can then plot the `Cell` object by using the `BatteryChart` object, which displays objects in a 3-D Cartesian coordinate system based on the world coordinate system. The height of a `Cell` object is aligned with the *z*-axis of the reference frame.

The cell `Geometry` property is not essential for system simulation except in use cases where you must consider detailed thermal and electrical design.

Creation

Syntax

```
import simscape.battery.builder.*; batteryCell = Cell
import simscape.battery.builder.*; batteryCell = Cell(Name=Value)
```

Description

`import simscape.battery.builder.*; batteryCell = Cell` creates a single battery cell with default property values. The object is linked to the Battery (Table-Based) block.

`import simscape.battery.builder.*; batteryCell = Cell(Name=Value)` creates a single battery cell and sets “Properties” on page 2-21 using one or more name-value arguments. For example, `batteryCell = Cell(Geometry=CylindricalGeometry,Mass=simscape.Value(1,"kg"))` creates a cylindrical battery cell with a mass of 1 kg.

Properties

`Geometry` — Geometry of battery cell
`double.empty` (default) | `CylindricalGeometry` object | `PouchGeometry` object | `PrismaticGeometry` object

Set of cell geometrical parameters associated with a specific cell format, specified as a `CylindricalGeometry` object, a `PouchGeometry` object, or a `PrismaticGeometry` object.

Example: `batteryCell.Geometry = CylindricalGeometry()`

`CellModelOptions` — Conditional parameters of cell component
`CellModelBlock` object (default)

Conditional parameters of the cell component model block used for simulation, specified as a `CellModelBlock` object. By default, the cell component model block is the Battery (Table-Based) block. You can also use your own cell component model block by specifying the `CellModelBlockPath` property of the `CellModelBlock` object accordingly.

`Mass` — Mass of battery cell
`simscape.Value(0.1,"kg")` (default) | `simscape.Value(positive scalar,"Length unit")` | positive scalar

Mass of the battery cell, specified as a `simscape.Value` object that represents a scalar with a unit of length. The value of this property must be strictly positive and lower than 100 kg.

If you set this property directly with a positive scalar value instead of using a `simscape.Value` object, the object converts the value to a `simscape.Value` object with `kg` as its physical unit.

Example: `batteryCell.Mass = simscape.Value(0.5,"kg")`

`Capacity` — Capacity of battery cell
`simscape.Value(5,"A*hr")` (default) | `simscape.Value(positive scalar,"Length unit")` | positive scalar

Capacity of the battery cell, specified as a `simscape.Value` object that represents a scalar with a unit of length. The value of this property must be strictly positive. This value is not carried over into the generated battery model.

If you set this property directly with a positive scalar value instead of using a `simscape.Value` object, the object converts the value to a `simscape.Value` object with `A*hr` as its physical unit.

Example: `batteryCell.Capacity = simscape.Value(0.5,"A*hr")`

`Energy` — Energy of battery cell
`simscape.Value(50,"W*hr")` (default) | `simscape.Value(positive scalar,"Length unit")` | positive scalar

Energy of the battery cell, specified as a `simscape.Value` object that represents a scalar with a unit of length. The value of this property must be strictly positive. This value is not carried over into the generated battery model.

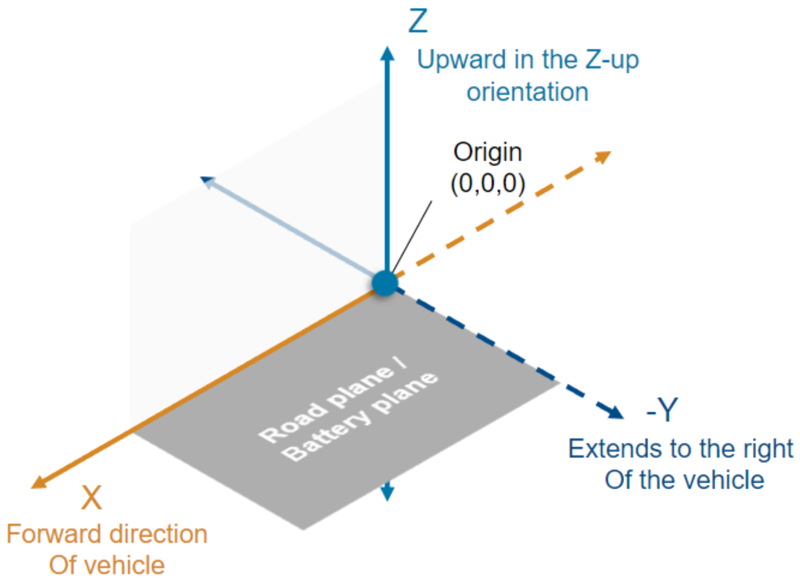
If you set this property directly with a positive scalar value instead of using a `simscape.Value` object, the object converts the value to a `simscape.Value` object with `W*hr` as its physical unit.

Example: `batteryCell.Energy = simscape.Value(50,"W*hr")`

`StackingAxis` — Preferential stacking direction
`"Y"` (default) | `"X"`

Preferential stacking direction for the arrangement of battery cells in a 2-D Cartesian coordinate system, specified as `"X"` or `"Y"`.

This figure shows the global coordinate system for batteries.


Z
Upward in the Z-up orientation
Origin (0,0,0)
Road plane / Battery plane
-Y
Extends to the right Of the vehicle
X
Forward direction Of vehicle

Example: `batteryCell.StackingAxis = "Y"`

`Position` — Location of battery cell
`[0 0 0]` (default) | vector of real and finite entries

Location of the battery cell in a 3-D Cartesian coordinate system, specified as a vector of real and finite entries. When you link the `Cell` object to a parent battery object, such as a `ParallelAssembly` object, the parent object overwrites this position.

Example: `batteryCell.Position = [0 0 0]`

`Name` — Name of battery cell
`"Cell1"` (default) | string

Name of the battery cell, specified as a string.

Example: `batteryCell.Name = "Cell2"`

`ParameterizationManufacturer` — Name of parameterization supplier
`"None"` (default) | `"A123"` | `"Korea Powercell"` | `"Panasonic"` | `"Sanyo"` | `"Tenergy"` | `"Valence"` | string

Name of the supplier you want to parameterize the battery cell from, specified as a string. To apply the parameterization data to your cell, such as geometry, mass, capacity, and energy, use the `applyCellDataFromPart` function.

For a list of available parameterizations, see “List of Pre-Parameterized Components” (Simscape Electrical).

Example: `batteryCell.ParameterizationManufacturer = "Panasonic"`

Dependencies

To enable this property, the `CellModelBlockPath` property of the `CellModelBlock` object you specified inside the `CellModelOptions` property of this cell must be equal to`"batt_lib/Cells/Battery(Table-Based)"`. In other words, the cell component model block of this object must be the Battery (Table-Based) block.

`ParameterizationPartNumber` — Part number of pre-parameterized battery cell
`"None"` (default) | string

Part number of the pre-parameterized battery cell, specified as a string. You can only choose a part available to the supplier you specified in the `ParameterizationManufacturer` property. To apply the parameterization data to your cell, such as geometry, mass, capacity, and energy, use the `applyCellDataFromPart` function.

For a list of available parameterizations, see “List of Pre-Parameterized Components” (Simscape Electrical).

Example: `batteryCell.ParameterizationPartNumber = "NCA103450"`

Dependencies

To enable this property, set `ParameterizationManufacturer` to any value other than `"None"`.

`PackagingVolume` — Volume of battery
`simscape.Value([],"m^3")` (default) | `simscape.Value` object

This property is read-only.

Volume of the battery, returned as a `simscape.Value` object with a unit of volume.

`CumulativeMass` — Cumulative mass of battery
`simscape.Value`

This property is read-only.

Cumulative mass of the battery, returned as a `simscape.Value` object with a unit of mass.

`NumModels` — Number of cell model blocks
positive scalar

This property is read-only.

Number of cell model blocks used for simulation, returned as a positive scalar.

`Format` — Geometrical format of battery cell
`"Pouch"` | `"Prismatic"` | `"Cylindrical"`

This property is read-only.

Geometrical format of the battery cell, returned as `"Pouch"`, `"Prismatic"`, or `"Cylindrical"`. This object derives this property value from the `Geometry` property.

`ThermalEffects` — Option to use lumped thermal mass
`"omit"` | `"model"`

This property is read-only.

Option to use or omit a lumped thermal mass model during the simulation, returned as `"omit"` or `"model"`.

`AvailableParameterizationPartNumbers` — List of all available parameterization part numbers
vector of strings

This property is read-only.

List of all parameterization part numbers available for the supplier you specified in the `ParameterizationManufacturer` property, returned as a vector of strings.

`AvailableParameterizationManufacturers` — List of all available parameterization manufacturers
vector of strings

This property is read-only.

List of all parameterization manufacturers available for the battery cell parameterization, returned as a vector of strings.

Type — Type of battery
`"Cell"`

This property is read-only.

Type of battery object, returned as `"Cell"`.

Examples

Create Cylindrical Cell Object

Create a `Cell` object with a cylindrical geometry.

```
batteryCell = Cell(Geometry = CylindricalGeometry)
```

Double the radius of the cell.

```
batteryCell.Geometry.Radius = 2*batteryCell.Geometry.Radius
```

Modify the position of the cell.

```
batteryCell.Position = [1 1 1]
```

Visualize the cell by using a `BatteryChart` object.

```
cellChart = BatteryChart(Battery = batteryCell)
```

Update Cell Properties From Selected Parameterization

Create a `Cell` object with a cylindrical geometry.

```
batteryCell = Cell(Geometry = CylindricalGeometry);
```

Choose the parameterization for the cell. Set the `ParameterizationManufacturer` and `ParameterizationPartNumber` properties.

```
batteryCell.ParameterizationManufacturer = "A123";
batteryCell.ParameterizationPartNumber = "ALM12V7";
```

Apply the parameterization data by using the `applyCellDataFromPart` function.

```
batteryCell.applyCellDataFromPart;
```

Version History

Introduced in R2022b

See Also

Apps
Battery Builder

Objects
`ParallelAssembly` | `BatteryChart` | `CylindricalGeometry` | `PouchGeometry` | `PrismaticGeometry` | `CellModelBlock`

Functions
`applyCellDataFromPart`

Topics
"Battery Modeling Workflow"
"Build Simple Model of Battery Pack in MATLAB and Simscape"
"Build Detailed Model of Battery Pack From Pouch Cells"

CellModelBlock

Set of conditional parameters for `Cell` object

Description

Use `CellModelBlock` to construct the `CellModelOptions` property of a `Cell` object. This property contains the conditional parameters of the block that you specify in the `CellModelBlockPath` property.

By default, the `CellModelBlockPath` property uses the Battery (Table-Based) block, but you can also specify your own custom block. For more information, see the CellModelBlockPath property.

Creation

Syntax

`import simscape.battery.builder.*; cellModelOpt = CellModelBlock`

Description

`import simscape.battery.builder.*; cellModelOpt = CellModelBlock` sets default model options for the `Cell` object. To modify these options for your application, use dot notation.

Properties

`CellModelBlockPath` — Library path to Simscape battery cell block
`"batt_lib/Cells/Battery(Table-Based)"` (default) | string

Library path to a Simscape battery cell block, specified as a string. The default component model block is the Battery (Table-Based) block.

You can also specify your own custom cell block, as long as all your custom block meets all of these requirements:

- The custom block must provide exactly two electrical conserving ports named **n** and **p**.
- All variables must be scalar-valued.
- The custom block cannot contain a variable or intermediate variable called *power_dissipated*.

You can connect the custom block to any number of thermal nodes. The software connects all nodes in parallel.

If you specify your own custom block, the fields of the `BlockParameters` structure depend on the parameters that you specified inside your custom block. To modify these options for your application, use dot notation.

Example: `cellModelOpt.CellModelBlock(CellModelBlockPath="CustomBlocks_lib/CellBlock");`

BlockParameters — Conditional compile-time parameters of cell block
structure (default)

Conditional compile-time parameters of the cell block that you specify in the `CellModelBlockPath` property.

Note This property does not include the run-time parameters of the specified cell block.

These are the conditional parameters for the default Battery (Table-Based) block:

- `T_dependence` — Temperature-dependent tables
- `thermal_port` — Thermal port
- `prm_age_OCV` — Storage condition
- `prm_age_capacity` — Capacity calendar aging
- `prm_age_resistance` — Internal resistance calendar aging
- `prm_age_modeling` — Modeling option
- `prm_dyn` — Charge dynamics
- `prm_dir` — Current directionality
- `prm_fade` — Fade characteristic
- `prm_leak` — Self-discharge

If you specify a custom block in the `CellModelBlockPath` property, these fields might differ.

Example: `cellModelOpt.BlockParameters.thermal_port = "model"`

Examples

Set Default CellModelOptions Property

Create a default `CellModelBlock` option set.

```
cellModelOpt = CellModelBlock
```

```
  CellModelBlock with properties:

    CellModelBlockPath: "batt_lib/Cells/Battery↵(Table-Based)"
       BlockParameters: [1×1 struct]
```

Show the `BlockParameters` property.

```
cellModelOpt.BlockParameters
```

```
    struct with fields:

          T_dependence: no
          thermal_port: omit
           prm_age_OCV: OCV
      prm_age_capacity: disabled
    prm_age_resistance: disabled
      prm_age_modeling: equation
```

```
                prm_dyn: off
                prm_dir: noCurrentDirectionality
               prm_fade: disabled
               prm_leak: disabled
```

Specify options using dot notation.

```
cellModelOpt.BlockParameters.thermal_port = "model";
```

Any property values you do not specify retain their default values.

Modify `BlockParameters` Property of a `Cell` Object

Create a default `Cell` object.

```
batteryCell = Cell
```

```
  batteryCell = 

  Cell with properties:

            Geometry: []
    CellModelOptions: [1×1 simscape.battery.builder.CellModelBlock]
                Mass: [1×1 simscape.Value]
```

Show the `BlockParameters` property of the `CellModelOptions` object inside the `Cell` object.

```
batteryCell.CellModelOptions.BlockParameters
```

```
      struct with fields:

          T_dependence: no
          thermal_port: model
           prm_age_OCV: OCV
      prm_age_capacity: disabled
    prm_age_resistance: disabled
      prm_age_modeling: equation
               prm_dyn: off
               prm_dir: noCurrentDirectionality
              prm_fade: disabled
              prm_leak: disabled
```

Specify options using dot notation.

```
batteryCell.CellModelOptions.BlockParameters.thermal_port = "model";
```

Version History

Introduced in R2022b

See Also

Apps
Battery Builder

Objects
`Cell`

Simscape Blocks
Battery (Table-Based)

Topics
“Build Simple Model of Battery Pack in MATLAB and Simscape”
“Build Detailed Model of Battery Pack From Pouch Cells”

CylindricalGeometry

Cylindrical geometry for battery cell

Description

Use `CylindricalGeometry` to represent the geometry of a cylindrical battery cell. To specify the dimensions of a cylindrical geometry, use the `Radius` and `Height` properties.

Creation

Syntax

```
import simscape.battery.builder.*; cylinder = CylindricalGeometry
import simscape.battery.builder.*; cylinder = CylindricalGeometry(Name=Value)
```

Description

`import simscape.battery.builder.*; cylinder = CylindricalGeometry` creates a cylindrical geometry with default property values.

`import simscape.battery.builder.*; cylinder = CylindricalGeometry(Name=Value)` creates a cylindrical geometry with a specified radius and height using name-value arguments.

Properties

`Radius` — Radius of cylinder
`simscape.Value(0.01,"m")` (default) | `simscape.Value(positive scalar,"Length unit")` | positive scalar

Radius of the cylinder, specified as a positive scalar or a `simscape.Value` object that represents a positive scalar with a unit of length. This value must be less than `0.5` meters.

If you set this property directly with a positive scalar value instead of using a `simscape.Value` object, the object converts the value to a `simscape.Value` object with meter as its physical unit.

Example: `cylinder.Radius = simscape.Value(0.0105,"m")`

Example: `cylinder.Radius = 0.0105`

`Height` — Height of cylinder
`simscape.Value(0.07,"m")` (default) | `simscape.Value(positive scalar,"Length unit")` | positive scalar

Height of the cylinder, specified as a positive scalar or a `simscape.Value` object that represents a positive scalar with a unit of length. The length of the cylinder is aligned with the *z*-axis of the reference frame. This value must be strictly positive.

If you set this property directly with a positive scalar value instead of using a `simscape.Value` object, the object converts the value to a `simscape.Value` object with meter as its physical unit.

Example: `cylinder.Height = simscape.Value(0.07,"m")`

Example: `cylinder.Height = 0.07`

Version History

Introduced in R2022b

See Also

Apps
Battery Builder

Objects
`PouchGeometry` | `PrismaticGeometry` | `Cell` | `ParallelAssembly` | `Module` | `ModuleAssembly` | `Pack`

Topics
"Build Simple Model of Battery Pack in MATLAB and Simscape"
"Build Detailed Model of Battery Pack From Pouch Cells"

Module

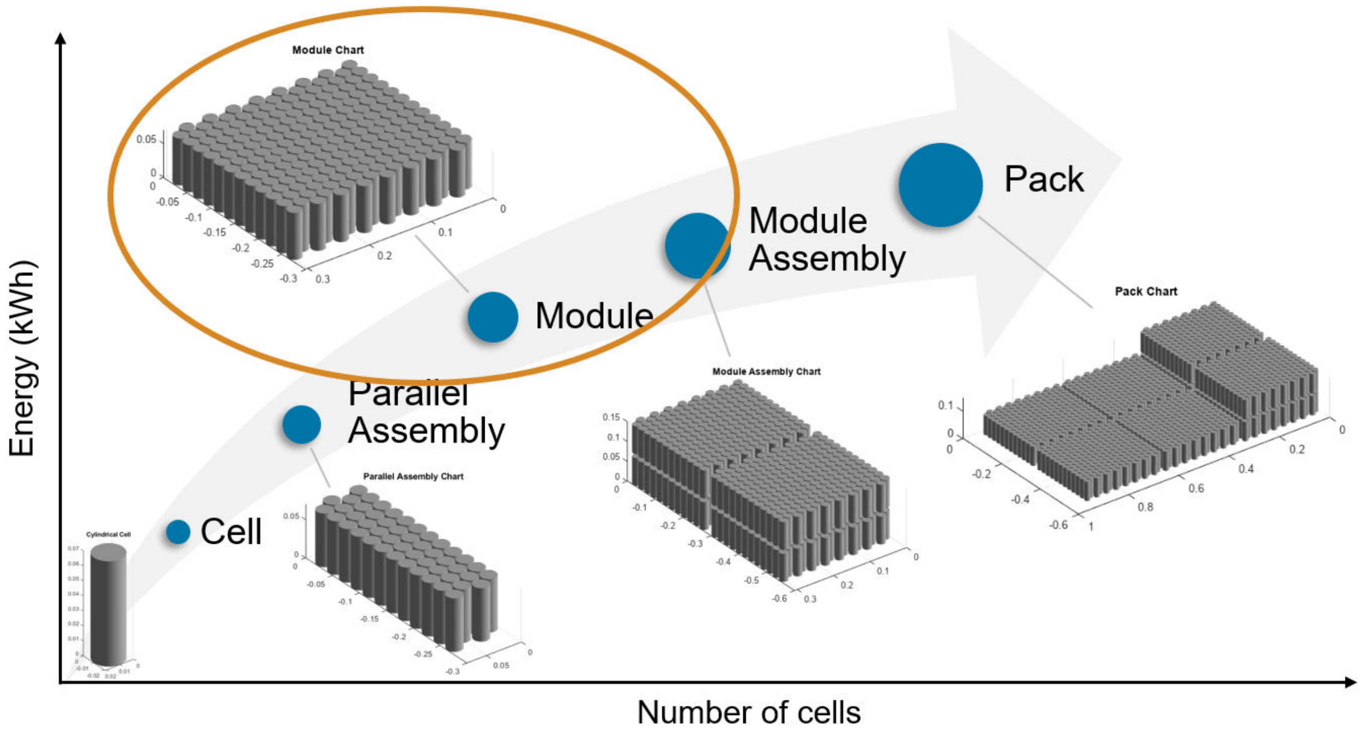
Create module of battery parallel assemblies

Description

Use `Module` to create a battery module object that represents a number of battery parallel assemblies connected electrically in series. You can use this object as an input to the `ModuleAssembly` object to create larger battery models.

To define the number of parallel assemblies or cells connected in series, use the `NumSeriesAssemblies` property. To generate a Simscape model of your `Module` object, use the `buildBattery` function. This object only supports the definition of structural or design parameters. You can modify the run-time parameters for this model block and its constituent cells after you create the model.

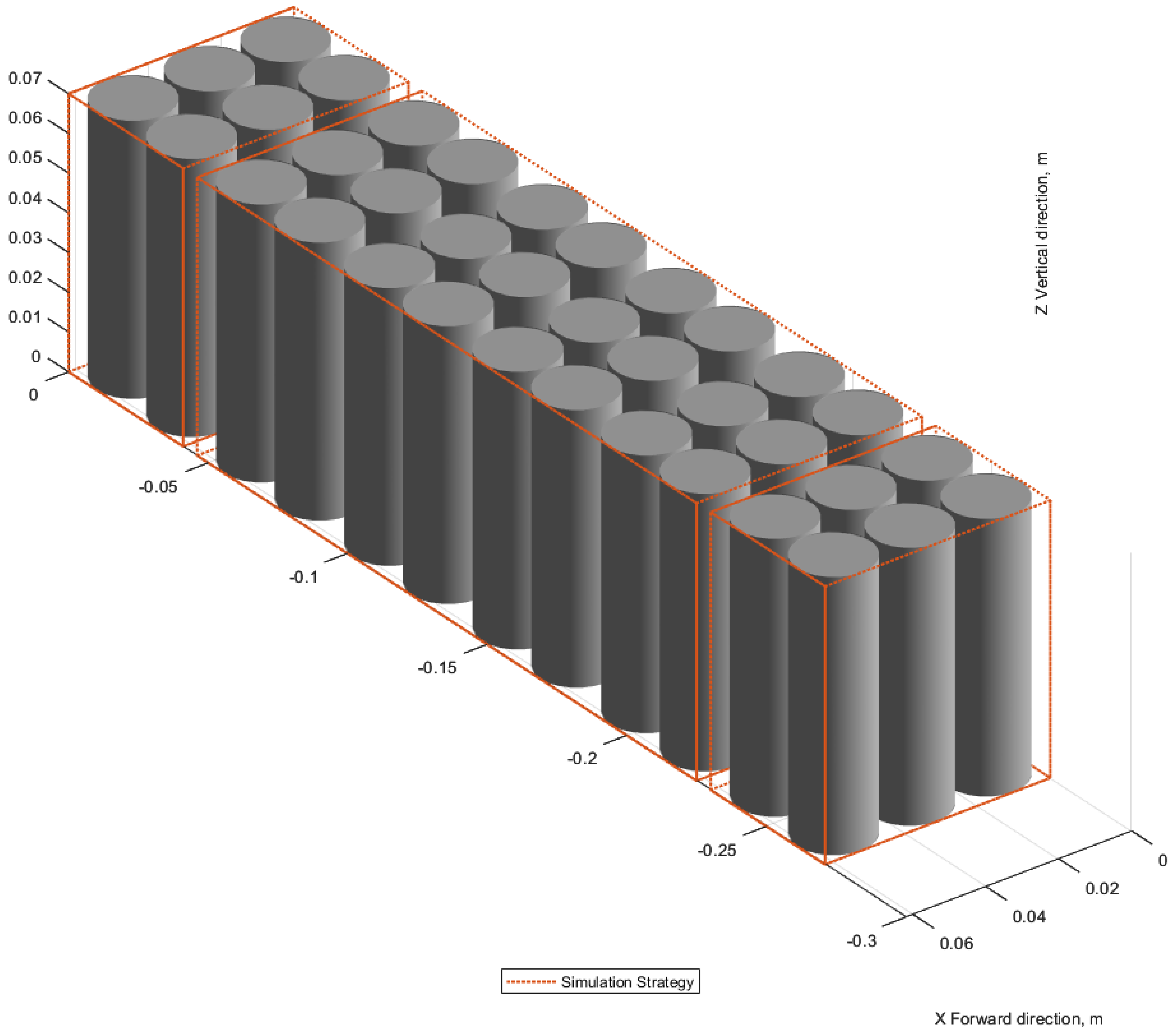
The `Module` object is the third stage of a battery pack system model in a bottom-up approach. Pack models are required for architecture evaluation in early development stages, software and hardware development, system integration and requirement evaluation, cooling system design, control strategy development hardware-in-the-loop, and many more applications.



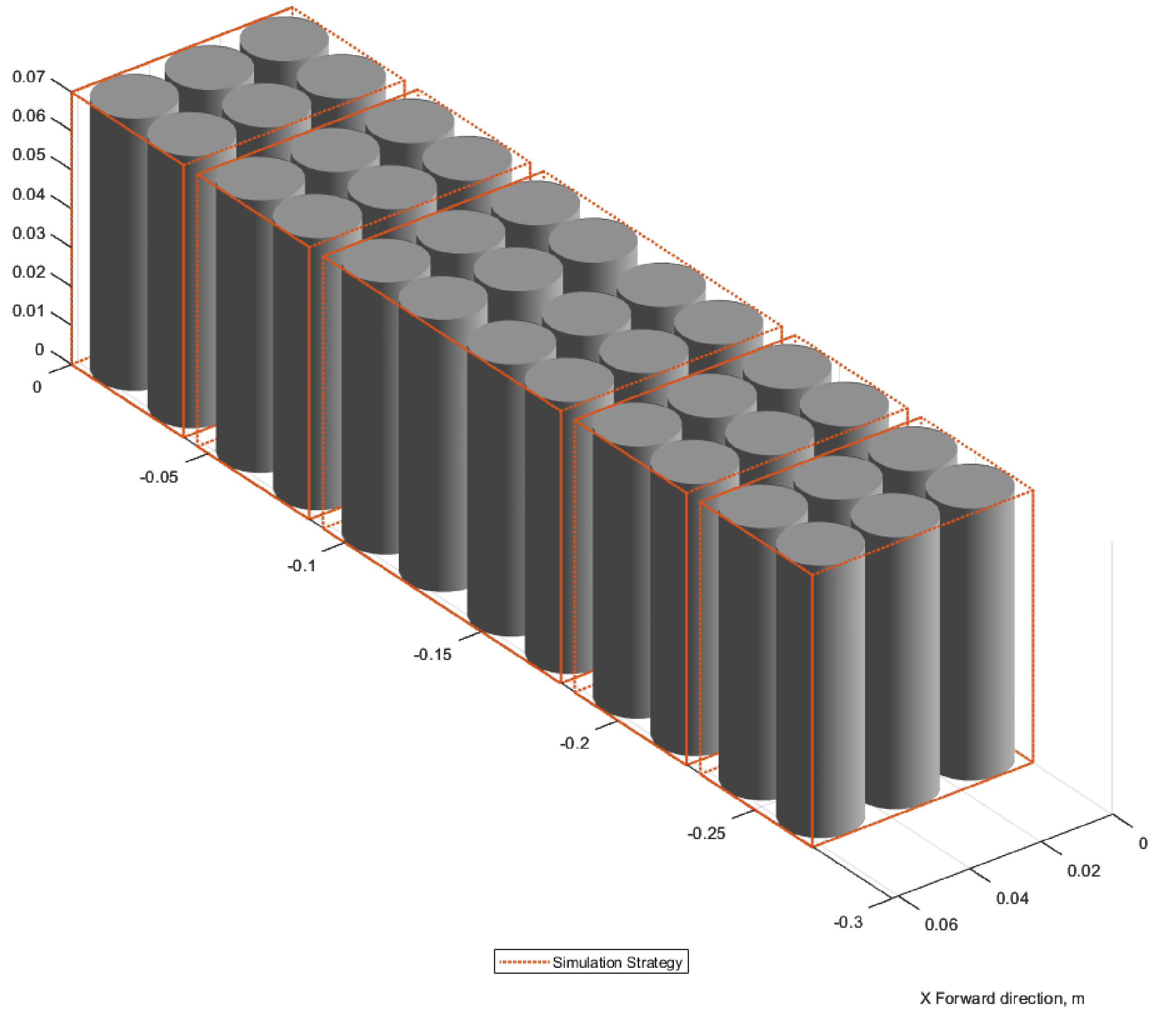
By default, the parallel assemblies or cells are stacked along the *y*-axis of a Cartesian coordinate system and replicated a number of times equal to the value of the `NumSeriesAssemblies` property. To stack the parallel assemblies along the *x*-axis, set the `StackingAxis` property to `"X"`.

Specify Model Resolution

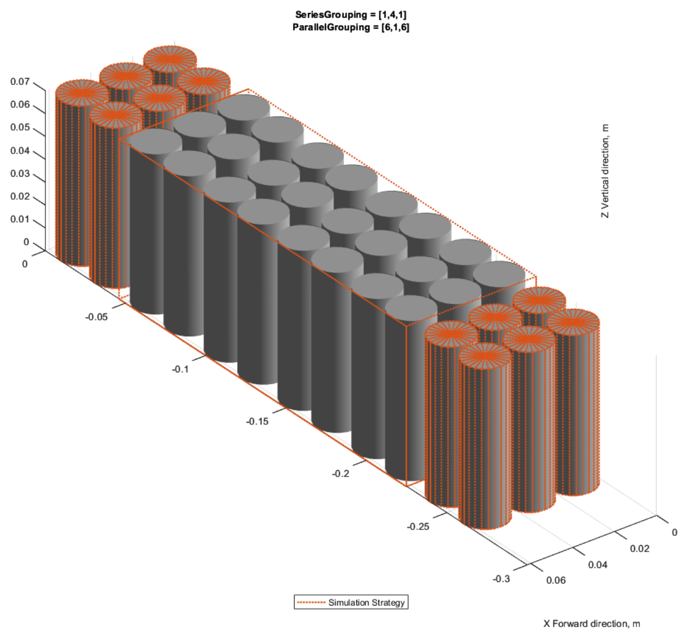
The recommended model resolution for a `Module` object is `"Lumped"`. You can increase or decrease the local resolution for a `Module` object by using the `ModelResolution` property. If you set the `ModelResolution` property to `"Grouped"`, you can define a custom model resolution by using the `SeriesGrouping` and `ParallelGrouping` properties. For example, consider a battery architecture that comprises six parallel assemblies connected in series (`NumSeriesAssemblies` = 6). Each parallel assembly comprises six cells. This architecture requires at least 36 electrical battery models for simulation. To improve model speed and optimize the compilation times, you can choose to use only three electrical models by setting the `SeriesGrouping` property to `[1 4 1]`. The first and third submodules comprise one parallel assembly. The second submodule comprises the remaining four parallel assemblies.

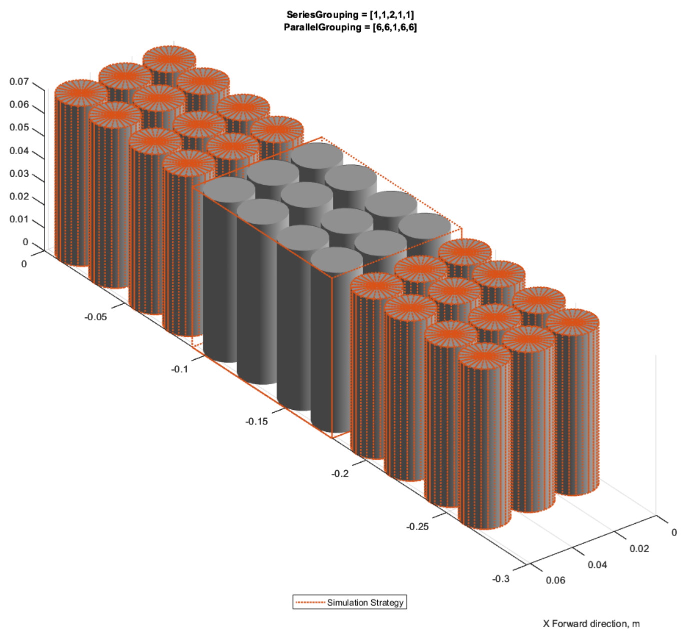


Alternatively, you can choose to have further resolution for every parallel assembly by changing `SeriesGrouping` to `[1 1 2 1 1]`.



For the submodules with a single parallel assembly, you can further discretize the electrical and thermal models to obtain the resolution for each cell by setting the `ParallelGrouping` property to [6 1 6] and [6 6 1 6 6], respectively.





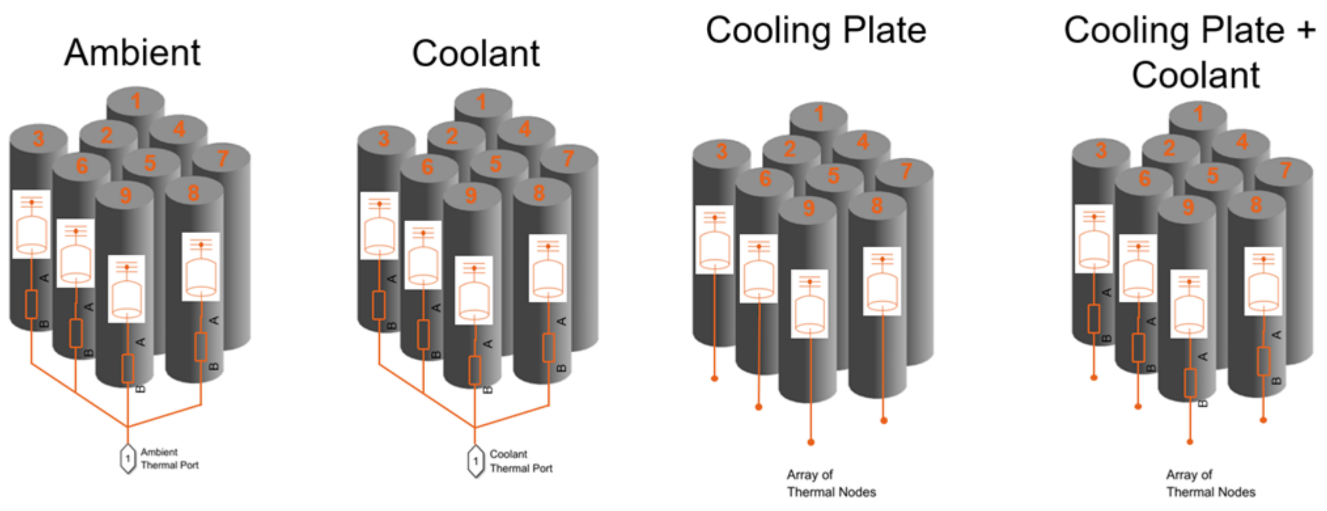
For more information about model resolution and simulation strategy, see the “SeriesGrouping” on page 2-0 and “ParallelGrouping” on page 2-0 properties.

Thermal Boundary Conditions

Thermal boundary conditions define the specific heat transfer mechanisms that occur at each interface of a cell thermal model and its surroundings. In battery systems, cells are typically thermally coupled to different heat sources and sinks, all of which have an effect on the battery cell temperature. The number and type of thermal boundary conditions for a cell model depends on the thermal and mechanical design of the battery system.

For example, you can place cells on an aluminium cooling plate to enhance heat removal and, at the same time, join them together mechanically with a potting compound that effectively eliminates or decreases the inter-cell heat exchange path. The cell temperature has a direct impact on battery performance and lifetime. Therefore, it is crucial to predict this state in dynamic simulation.

Inside a battery object, you can set up a thermal network of lumped-thermal-mass cell models to simultaneously capture the thermal paths to the ambient, the coolant, and/or the cooling plate:

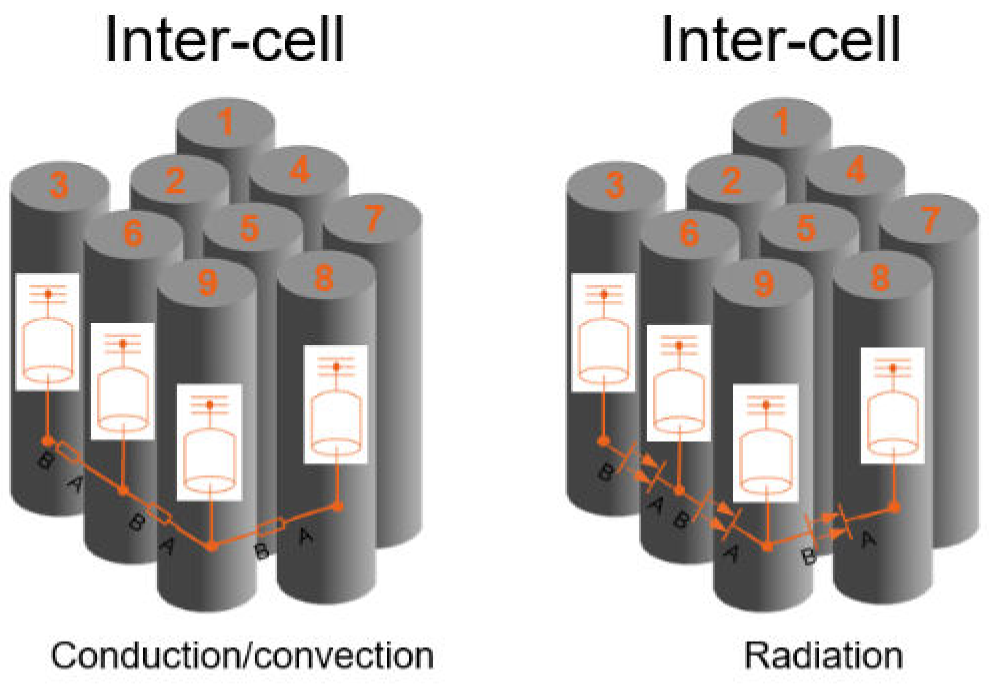


These options are not mutually exclusive. For example, your battery model can combine both the coolant thermal path and the cooling thermal plates to model individual thermal resistances between the individual cells and the sections of the cooling plate.

For more information about thermal paths, see the “AmbientThermalPath” on page 2-0 , “CoolantThermalPath” on page 2-0 , and “CoolingPlate” on page 2-0 properties.

You can also model direct cell-to-cell heat exchange. This is important when you want to simulate more detailed thermal management strategies or even thermal propagation scenarios where inter-cell heat transfer happens at faster rates than ambient or coolant rates. In the battery industry, you can link battery cells to each other through many different means. For example, you can link cylindrical cells by using potting compounds for mechanical rigidity, stability, and thermal isolation, or other types of thermal interface materials. You can also use dielectric fluids or other compounds to heat or cool down cylindrical cells, as well as forced air convection.

You can define the thermal parameters for the inter-cell heat exchange after you create the battery model. You can find these parameters from first principles calculations and more detailed 3D simulations.



These options are not mutually exclusive.

For more information about inter-cell thermal paths, see the “InterCellThermalPath” on page 2-0 and “InterCellRadiativeThermalPath” on page 2-0 properties.

Creation

Syntax

```
import simscape.battery.builder.*; batteryModule = Module
import simscape.battery.builder.*; batteryModule = Module(Name=Value)
```

Description

`import simscape.battery.builder.*; batteryModule = Module` creates a battery module that comprises parallel assemblies with default property values.

`import simscape.battery.builder.*; batteryModule = Module(Name=Value)` sets “Properties” on page 2-38 using one or more name-value arguments. For example, create a module with four default parallel assemblies stacked along the *y*-axis with a gap between the parallel assemblies equal to 0.05 m.

```
module = Module(...
    ParallelAssembly=ParallelAssembly(), ...
    NumSeriesAssemblies=4, ...
    StackingAxis="Y",...
    InterParallelAssemblyGap=simscape.Value(0.05,"m"));
```

Properties

`ParallelAssembly` — Parallel assembly component in module

`ParallelAssembly` object (default)

Parallel assembly component in the module, specified as a `ParallelAssembly` object. The `Module` object creates this component and then electrically connects it in series a number of times equal to the value of the `NumSeriesAssemblies` property.

Example: `batteryModule.ParallelAssembly = ParallelAssembly`

`NumSeriesAssemblies` — Number of parallel assemblies connected in series

1 (default) | positive integer

Number of parallel assemblies connected in series inside the module, specified as a strictly positive and finite integer. The value of this property must be less than `150`.

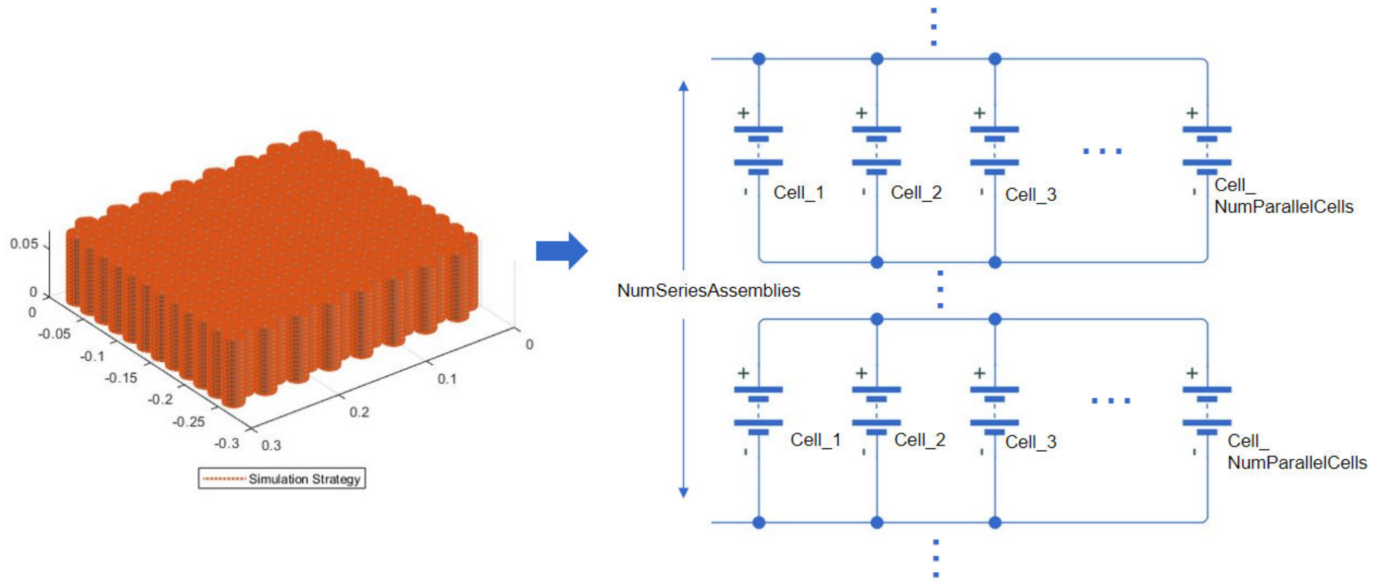
Example: `batteryModule.NumSeriesAssemblies = 48`

`ModelResolution` — Model resolution in simulation

`"Lumped"` (default) | `"Detailed"` | `"Grouped"`

Model resolution or fidelity in the simulation, specified as:

- `"Lumped"` — Choose this value for the lowest fidelity. The module uses only one electrical mode. To obtain the fastest compilation time and running time, use this value.
- `"Detailed"` — Choose this value for the highest fidelity. The module uses one electrical model and one thermal model for each battery cell.



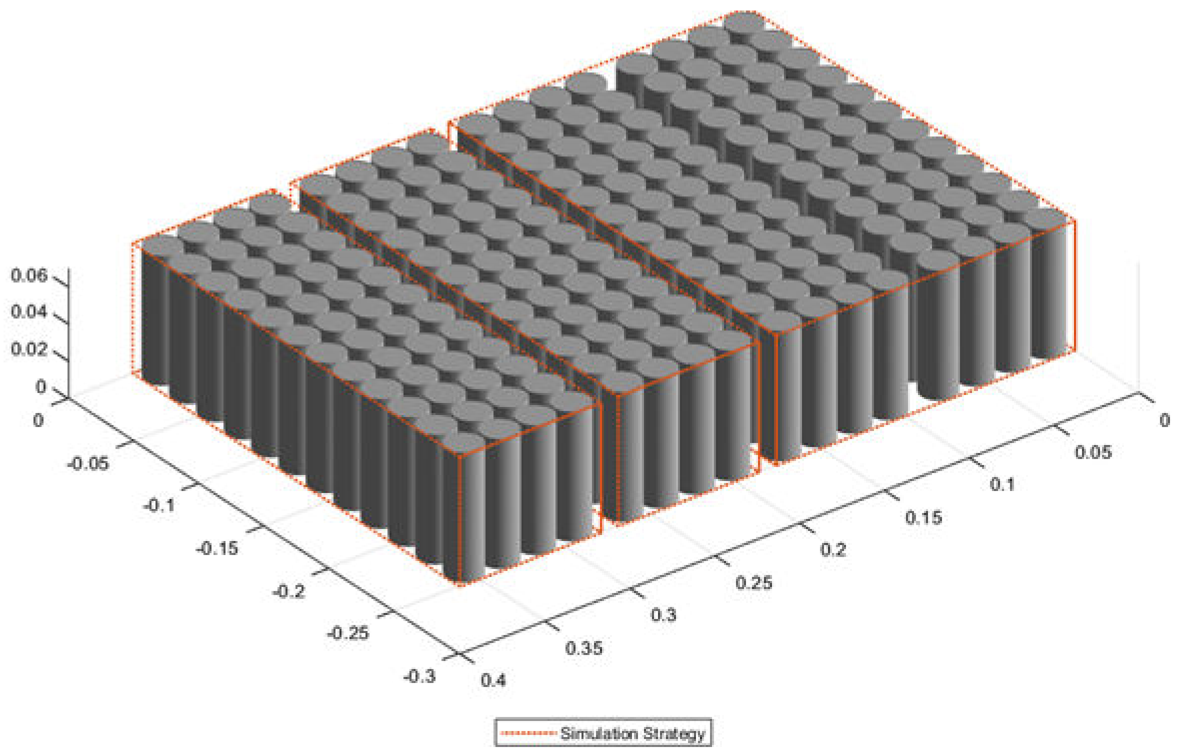
- `"Grouped"` — Choose this value to use a custom simulation strategy based on the `SeriesGrouping` and `ParallelGrouping` properties.

Example: `batteryModule.ModelResolution = "Detailed"`

SeriesGrouping — Modeling strategy along series connections
1 (default) | row vector of positive integers

Modeling strategy for the module along the series connections, specified as a row vector of positive integers.

The length of this vector specifies the number of individual electrical models required. Each entry specifies how many parallel assemblies are lumped within the specified electrical model. For example, if your module comprises four parallel assemblies (`NumSeriesAssemblies` = 4) and you set this property to `[2 1 1]`, the object discretizes the module in three individual electrical models. The first model comprises two of the original parallel assemblies.



The sum of the entries in this vector must be equal to the value of the `NumSeriesAssemblies` property.

Example: `batteryModule.SeriesGrouping = [2 1 1]`

Dependencies

To enable this property, set the `ModelResolution` property to `"Grouped"`.

ParallelGrouping — Modeling strategy for every parallel assembly in module
1 (default) | row vector of positive integers

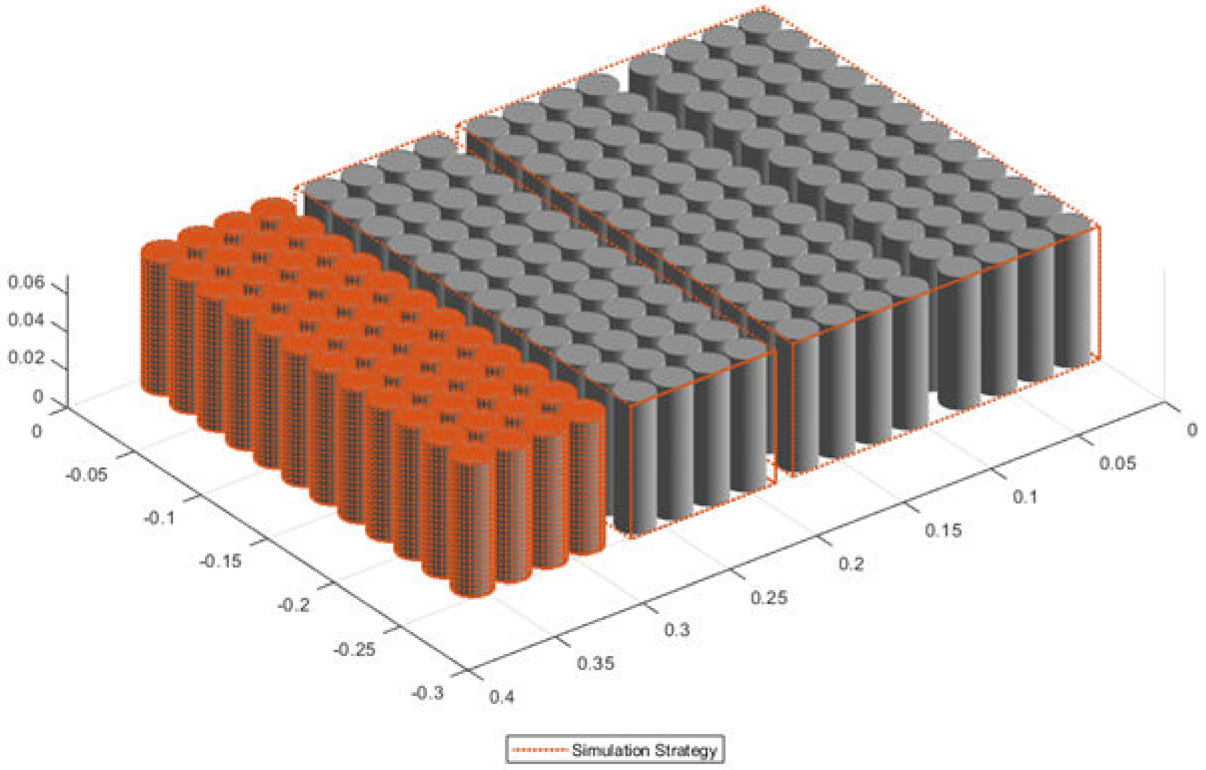
Modeling strategy for every parallel assembly defined in the `SeriesGrouping` property, specified as a vector of positive integers. The length of this vector must be equal to the length of the `SeriesGrouping` property value.

Each entry of this vector specifies the number of individual electrical models for entry of the `SeriesGrouping` property. The values of the entries must be `1` or the value of the `NumParallelCells` property.

For example, assume that your module comprises these elements:

- Four parallel assemblies (`NumSeriesAssemblies` = `4`)
- 48 cylindrical cells for each parallel assembly (`NumParallelCells` = `48`)
- Three individual electrical models, in which the first model comprises two of the original parallel assemblies (`SeriesGrouping` = `[2 1 1]`)

If you set this property to `[1 1 48]`, the object discretizes the module into 50 individual electrical models, where each cell of the fourth parallel assembly has an electrical model.



The value of an entry in this property must be `1` if the value of the corresponding entry of the `SeriesGrouping` property is larger than 1.

Example: `batteryModule.ParallelGrouping = [1 1 48]`

Dependencies

To enable this property, set the `ModelResolution` property to `"Grouped"`.

`InterParallelAssemblyGap` — Shortest distance between parallel assemblies
`simscape.Value(0.001,"m")` (default) | `simscape.Value(positive scalar,"Length unit")` | positive scalar

Shortest distance between parallel assemblies inside the module, specified as a positive scalar or a `simscape.Value` object that represents a positive scalar with a unit of length. The value of this property must be less than 0.1 m.

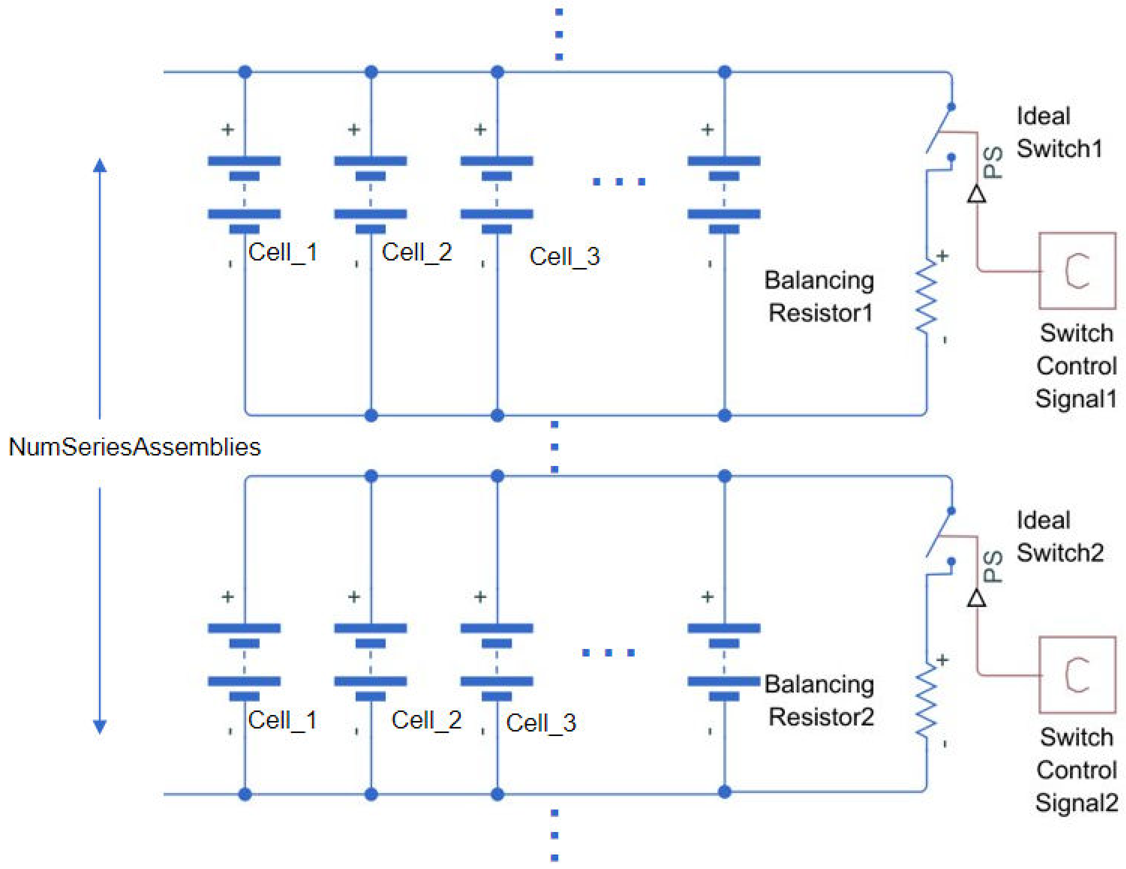
If you set this property directly with a positive scalar value instead of using a `simscape.Value` object, the object converts the value to a `simscape.Value` object with meter as its physical unit.

Example: `batteryModule.InterParallelAssemblyGap = simscape.Value(0.01,"m")`

Example: `batteryModule.InterParallelAssemblyGap = 0.01`

`BalancingStrategy` — State-of-charge balancing strategy
`"None"` (default) | `"Passive"`

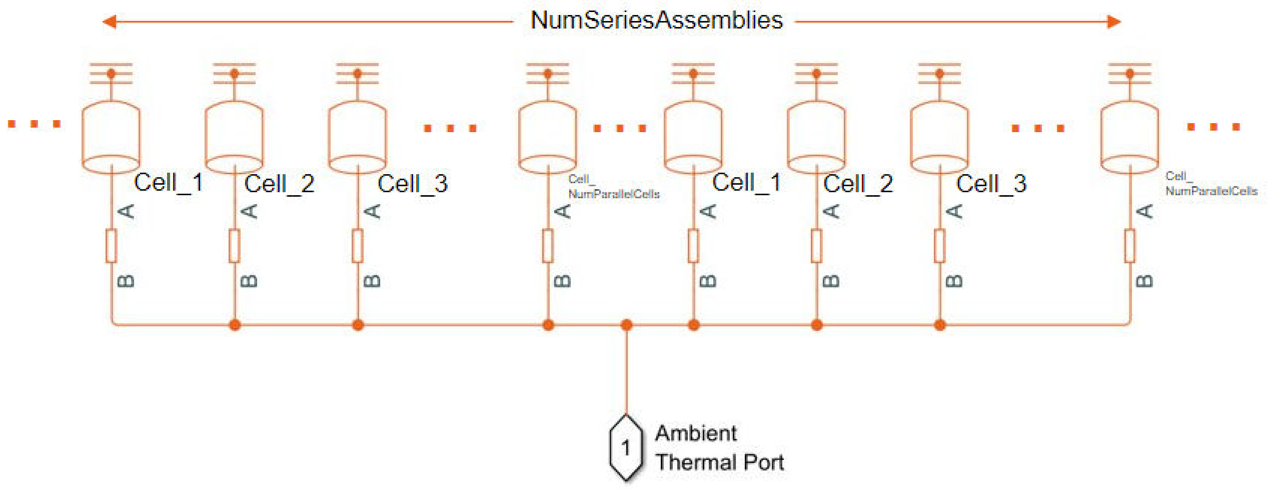
State-of-charge balancing strategy for the module, specified as `"None"` or `"Passive"`. Set this property to `"Passive"` to add an array of ideal balancing circuits electrically connected in parallel to every parallel assembly and a physical port of size equal to the value of the `NumSeriesAssemblies` property used for switch control. The switch is open when the control signal is equal to `0`. The switch is closed when the control signal is equal to `1`.



Example: `batteryModule.BalancingStrategy = "Passive"`

`AmbientThermalPath` — Option to use simple thermal resistance block
`"None"` (default) | `"CellBasedThermalResistance"`

Option to use a simple thermal resistance block connected between the cells and a Simscape thermal network, specified as `"CellBasedThermalResistance"` or `"None"`.



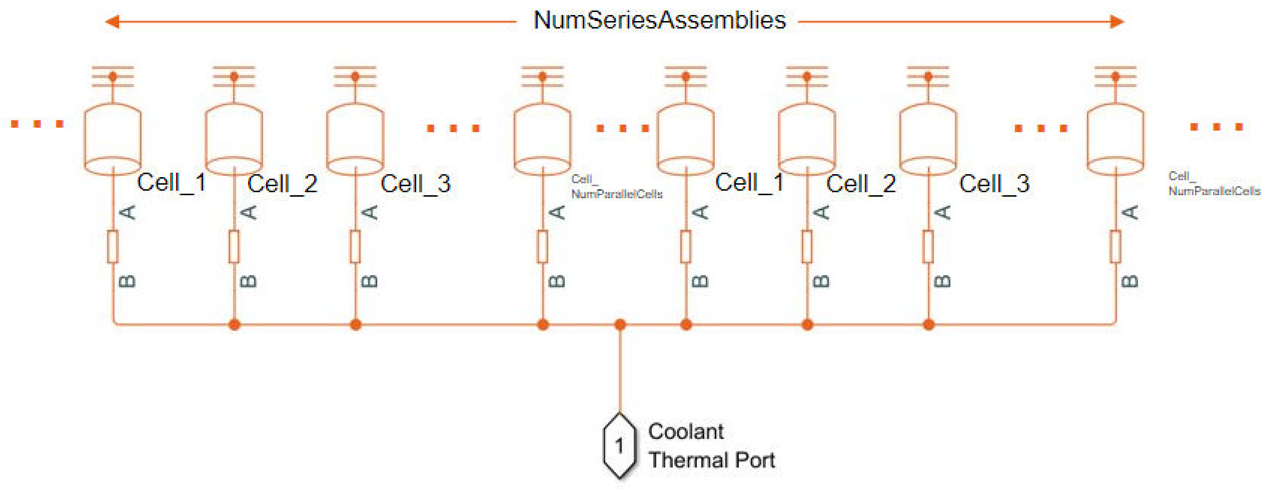
Note Setting this property automatically propagates its value to all the subcomponent battery objects inside this Module object. However, this change does not propagate to the other battery objects in your MATLAB workspace.

Example: `batteryModule.AmbientThermalPath = "CellBasedThermalResistance"`

`CoolantThermalPath` — Option to use simple thermal resistance block
`"None"` (default) | `"CellBasedThermalResistance"`

Option to use a simple thermal resistance block connected between the cells and a Simscape thermal network, specified as `"CellBasedThermalResistance"` or `"None"`.

If you also define a cooling plate surface, the object connects the thermal resistance block between each battery thermal model and its corresponding element in the array of thermal nodes.

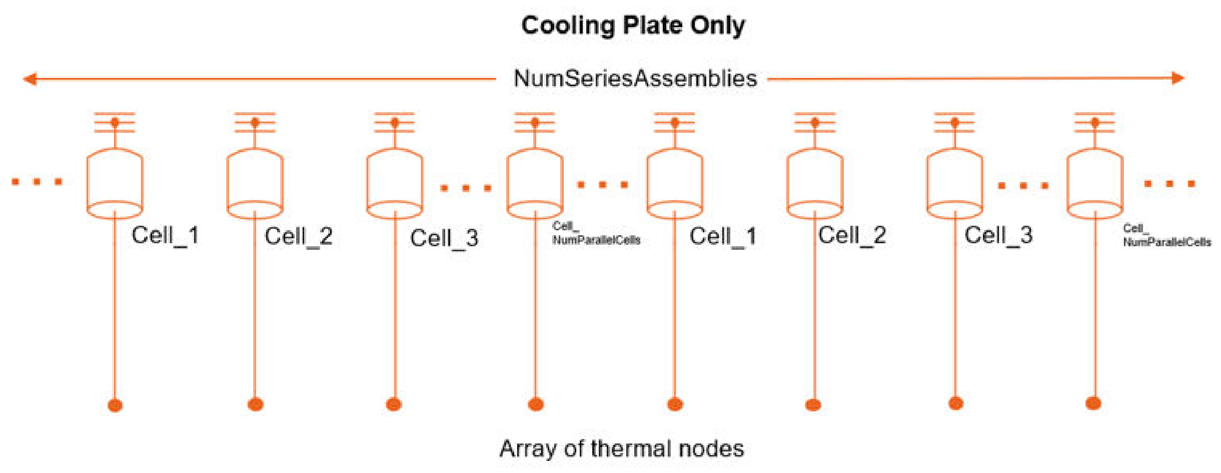


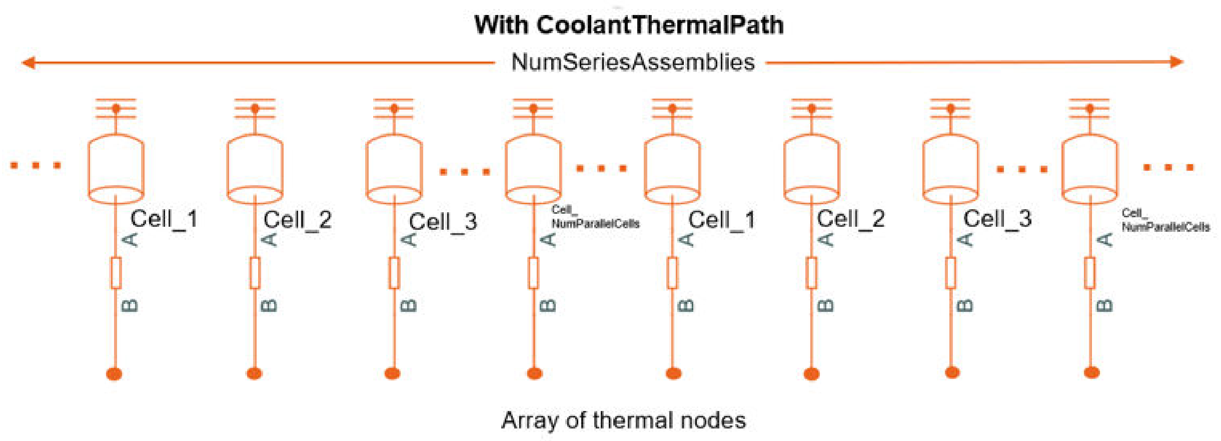
Note Setting this property automatically propagates its value to all the subcomponent battery objects inside this Module object. However, this change does not propagate to the other battery objects in your MATLAB workspace.

Example: `batteryModule.CoolantThermalPath = "CellBasedThermalResistance"`

`CoolingPlate` — Option to use cooling plate component
`"None"` (default) | `"Top"` | `"Bottom"`

Option to use a cooling plate component at a specific surface boundary, specified as `"Top"`, `"Bottom"`, or `"None"`.





Note Setting this property automatically propagates its value to all the subcomponent battery objects inside this Module object. However, this change does not propagate to the other battery objects in your MATLAB workspace.

Example: `batteryModule.CoolingPlate = "Top"`

`CoolingPlateBlockPath` — Option to specify which cooling plate block to assign
`"None"` (default) | `"batt_lib/Thermal/Edge Cooling"` | `"batt_lib/Thermal/Parallel Channels"` | `"batt_lib/Thermal/U-Shaped Channels"`

Option to specify which cooling plate block you want to assign to the Module object at the boundary defined by the `CoolingPlate` property, specified as `"batt_lib/Thermal/Edge Cooling"`, `"batt_lib/Thermal/Parallel Channels"`, `""batt_lib/Thermal/U-Shaped Channels""`, or `"None"`.

Note Setting this property automatically propagates its value to all the subcomponent battery objects inside this Module object. However, this change does not propagate to the other battery objects in your MATLAB workspace.

Example: `batteryModule.CoolingPlateBlockPath = "batt_lib/Thermal/Edge Cooling"`

`Position` — Location of battery object
`[0 0 0]` (default) | vector of real and finite entries

Location of the battery object in a 3-D Cartesian coordinate system, specified as a vector of real and finite entries.

Example: `batteryModule.Position = [0 0 0]`

`Name` — Name of module
`"Module1"` (default) | string

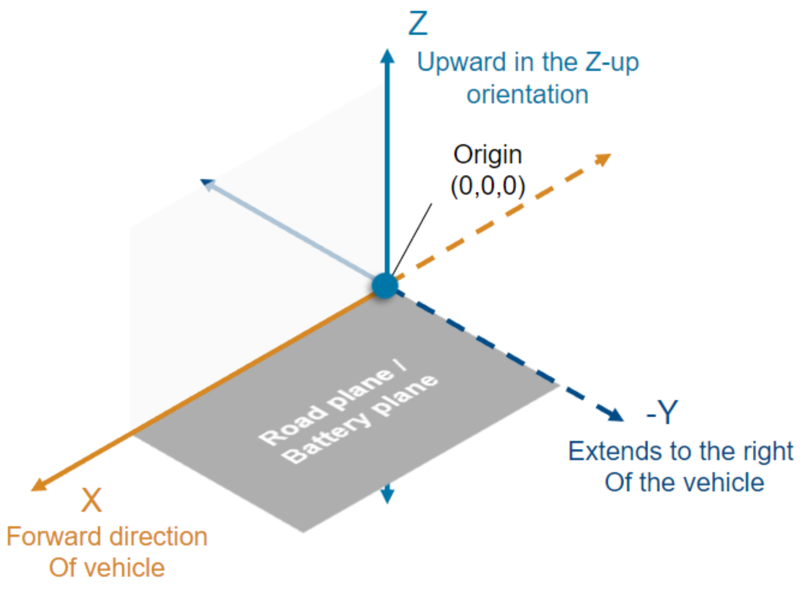
Name of the battery module, specified as a string.

Example: `batteryModule.Name = "Module2"`

`StackingAxis` — Preferential stacking direction
`"Y"` (default) | `"X"`

Preferential stacking direction for the arrangement of battery cells in a 2-D Cartesian coordinate system, specified as `"X"` or `"Y"`.

This figure shows the global coordinate system for batteries.


Z
Upward in the Z-up orientation
Origin (0,0,0)
Road plane / Battery plane
-Y
Extends to the right Of the vehicle
X
Forward direction Of vehicle

To plot the module object in the direction of the *x*-axis, set this property to `"X"` before creating the `BatteryChart` object.

Example: `batteryModule.StackingAxis = "Y"`

`MassFactor` — Additional non-cell-related mass
1 (default) | positive double

Additional non-cell-related mass added to the module by components such as busbars, tabs, and collector plates, specified as a strictly positive double greater than or equal to 1.

Example: `batteryModule.MassFactor = 1.2`

`NonCellResistance` — Option to use electrical resistance blocks
`'off'` (default) | on/off logical value

Option to use electrical resistance blocks to represent additional electrical resistances from non-cell components, specified as `'off'`, `'on'`, or as numeric or logical `1` (`true`) or `0` (`false`). A value of `'on'` is equivalent to `true`, and `'off'` is equivalent to `false`. Thus, you can use the value of this property as a logical value. The value is stored as an on/off logical value of type `matlab.lang.OnOffSwitchState`.

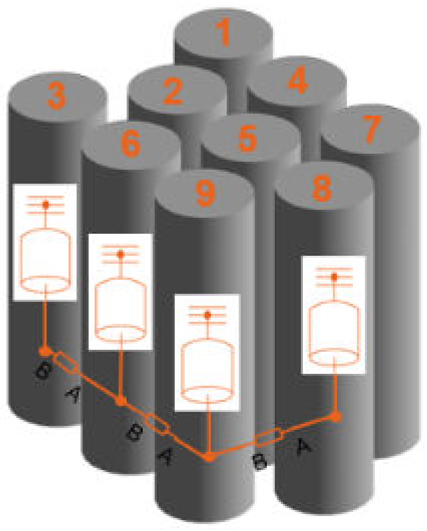
Example: `batteryModule.NonCellResistance = 'on'`

`InterCellThermalPath` — Option to use simple thermal resistance block for inter cell heat
`'off'` (default) | on/off logical value

Option to use thermal resistance blocks to represent a linear cell-to-cell heat transfer path between adjacent battery cells, specified as `'off'`, `'on'`, or as numeric or logical `1` (`true`) or `0` (`false`). A value of `'on'` is equivalent to `true`, and `'off'` is equivalent to `false`. Thus, you can use the value of this property as a logical value. The value is stored as an on/off logical value of type `matlab.lang.OnOffSwitchState`.

Enabling the inter-cell thermal path is useful only when you have selected a cell thermal model and more than one cell or cell model exists inside the battery.

You can define the value for the thermal resistance parameter after model creation.



If you set this property to `'on'`, you cannot set the **InterCellRadiativeThermalPath** property to `'on'`.

Note Setting this property automatically propagates its value to all the subcomponent battery objects inside this Module object. However, this change does not propagate to the other battery objects in your MATLAB workspace.

Example: `batteryModule.InterCellThermalPath = "on"`

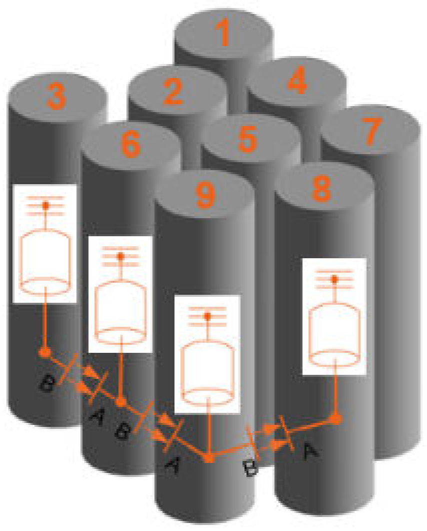
`InterCellRadiativeThermalPath` — Option to use simple thermal resistance block for inter cell heat

`'off'` (default) | on/off logical value

Option to use thermal resistance blocks to represent a radiative cell-to-cell heat transfer path between adjacent battery cells, specified as `'off'`, `'on'`, or as numeric or logical `1` (`true`) or `0` (`false`). A value of `'on'` is equivalent to `true`, and `'off'` is equivalent to `false`. Thus, you can use the value of this property as a logical value. The value is stored as an on/off logical value of type `matlab.lang.OnOffSwitchState`.

Enabling the inter-cell radiative thermal path is useful only when you have selected a cell thermal model and more than one cell or cell model exists inside the battery.

You can define the value for the radiation heat transfer parameters after model creation.



If you set this property to `'on'`, you cannot set the **InterCellThermalPath** property to `'on'`.

Note Setting this property automatically propagates its value to all the subcomponent battery objects inside this Module object. However, this change does not propagate to the other battery objects in your MATLAB workspace.

Example: `batteryModule.InterCellRadiativeThermalPath = "on"`

`PackagingVolume` — Volume of battery

`simscape.Value([],"m^3")` (default) | `simscape.Value` object

This property is read-only.

Volume of the battery, returned as a `simscape.Value` object with unit of volume.

`CumulativeMass` — Cumulative mass of battery
`simscape.Value` object

This property is read-only.

Cumulative mass of the battery, returned as a `simscape.Value` object with a unit of mass.

`CumulativeCellCapacity` — Cumulative cell capacity of battery
`simscape.Value` object

This property is read-only.

Cumulative cell capacity of the battery, returned as a `simscape.Value` object with unit of current multiplied by time.

The value of this property is equal to the individual cell capacity multiplied by the number of cells electrically connected in parallel inside this battery object.

`CumulativeCellEnergy` — Cumulative cell energy of battery
`simscape.Value` object

This property is read-only.

Cumulative cell energy of the battery, returned as a `simscape.Value` object with unit of energy.

The value of this property is equal to the individual cell energy multiplied by the total number of cells inside this battery object.

`NumModels` — Number of cell model blocks
double

This property is read-only.

Number of cell model blocks in the simulation, returned as a double.

`NumInterCellThermalConnections` — Number of internal thermal connections between cells
integer

This property is read-only.

Number of the internal thermal connections between the cells, returned as an integer.

`NumInterParallelAssemblyThermalConnections` — Number of internal thermal connections between parallel assemblies
integer

This property is read-only.

Number of the internal thermal connections between the parallel assemblies, returned as an integer.

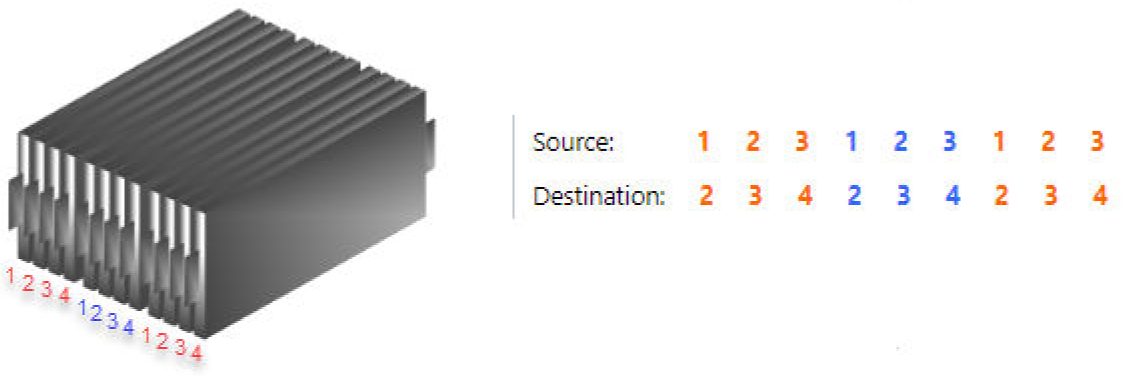
`InterCellConnectionsMapping` — 2-D map of inter-cell connections
matrix of integer

This property is read-only.

2-D map of the internal connections between the cells, returned as a matrix of integer.

The cell numbering convention refers to the `Layout` property. To understand the number assigned to each cell and how they relate to the their neighbors,see the `Layout` property.

For example, for a battery module with three parallel assemblies of four pouch cells each, this property is equal to:



Source:	1	2	3	1	2	3	1	2	3
Destination:	2	3	4	2	3	4	2	3	4

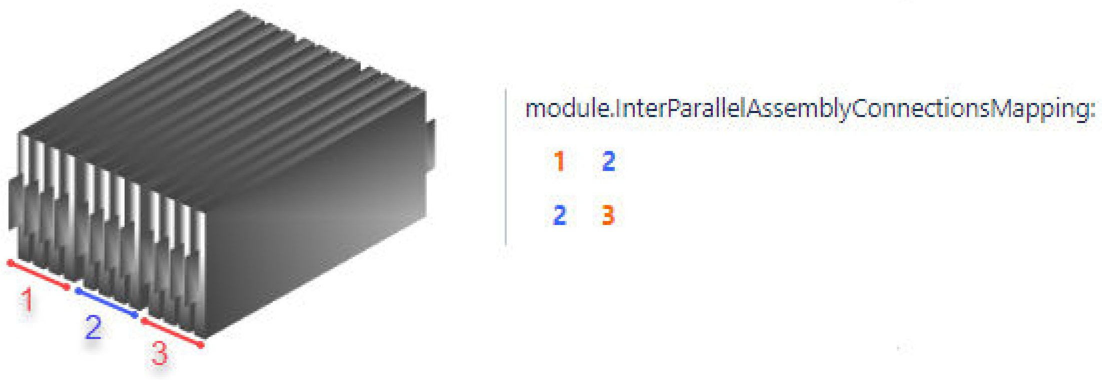
`InterParallelAssemblyConnectionsMapping` — 2-D map of inter-parallel assemblies connections
matrix of integer

This property is read-only.

2-D map of the internal connections between the parallel assemblies, returned as a matrix of integer.

The parallel assembly numbering increases in the direction of the stacking axis.

For example, for a battery module with three parallel assemblies of four pouch cells each, this property is equal to:



module.InterParallelAssemblyConnectionsMapping:

```
1  2
2  3
```

`InterParallelAssemblyCellConnectionsMapping` — 2-D map of internal connections between cells and parallel assemblies
matrix of integer

This property is read-only.

2-D map of the internal connections between the individual battery cell models inside each parallel assembly to their corresponding neighbor in the neighboring parallel assembly, returned as a matrix of integer.

The cell numbering convention for the mapping is related to the `Layout` property of each individual parallel assembly.

`CellNumbering` — Numbering for all cells
structure

This property is read-only.

Numbering for all of the cells inside of the battery, returned as a structure.

`ThermalNodes` — Vectorized thermal node information
structure

This property is read-only.

Vectorized thermal node information for external boundary conditions, returned as a structure. This information comprise XY location, XY dimensions, and number of thermal nodes. If you do not define a battery cell geometry, this property is an empty structure.

`Layout` — 2-D distribution of sub-components in parent
matrix of integer

This property is read-only.

2-D distribution of sub-components or cells within the parent, returned as a matrix of integer.

This property depends on the `StackingAxis` property and on the number of cells.

`Type` — Type of battery
`"Module"`

This property is read-only.

Type of battery object, returned as `"Module"`.

Examples

Create Pouch Cell Module with 10 Parallel Assemblies

Create a `Cell` object with a pouch geometry.

```
batteryCell = Cell(Geometry=PouchGeometry)
```

Create a `ParallelAssembly` object of three cells with the default topology.

```
pSet = ParallelAssembly(Cell=batteryCell,NumParallelCells=3)
```

Use this `ParallelAssembly` object to create a `Module` object of 10 parallel assemblies connected in series.

```
batteryModule = Module(ParallelAssembly=pSet,NumSeriesAssemblies=10)
```

Visualize the module by using a `BatteryChart` object.

```
moduleChart = BatteryChart(Battery=batteryModule);
```

Create Module with 10 Parallel Assemblies and Discretize Using Three Electrical Models

Create a `Cell` object and use it to create a `ParallelAssembly` object.

```
batteryCell = Cell(Geometry=CylindricalGeometry)
pSet = ParallelAssembly(Cell=batteryCell,NumParallelCells=4,Rows=4)
```

Use this `ParallelAssembly` object to create a `Module` object of 10 parallel assemblies connected in series.

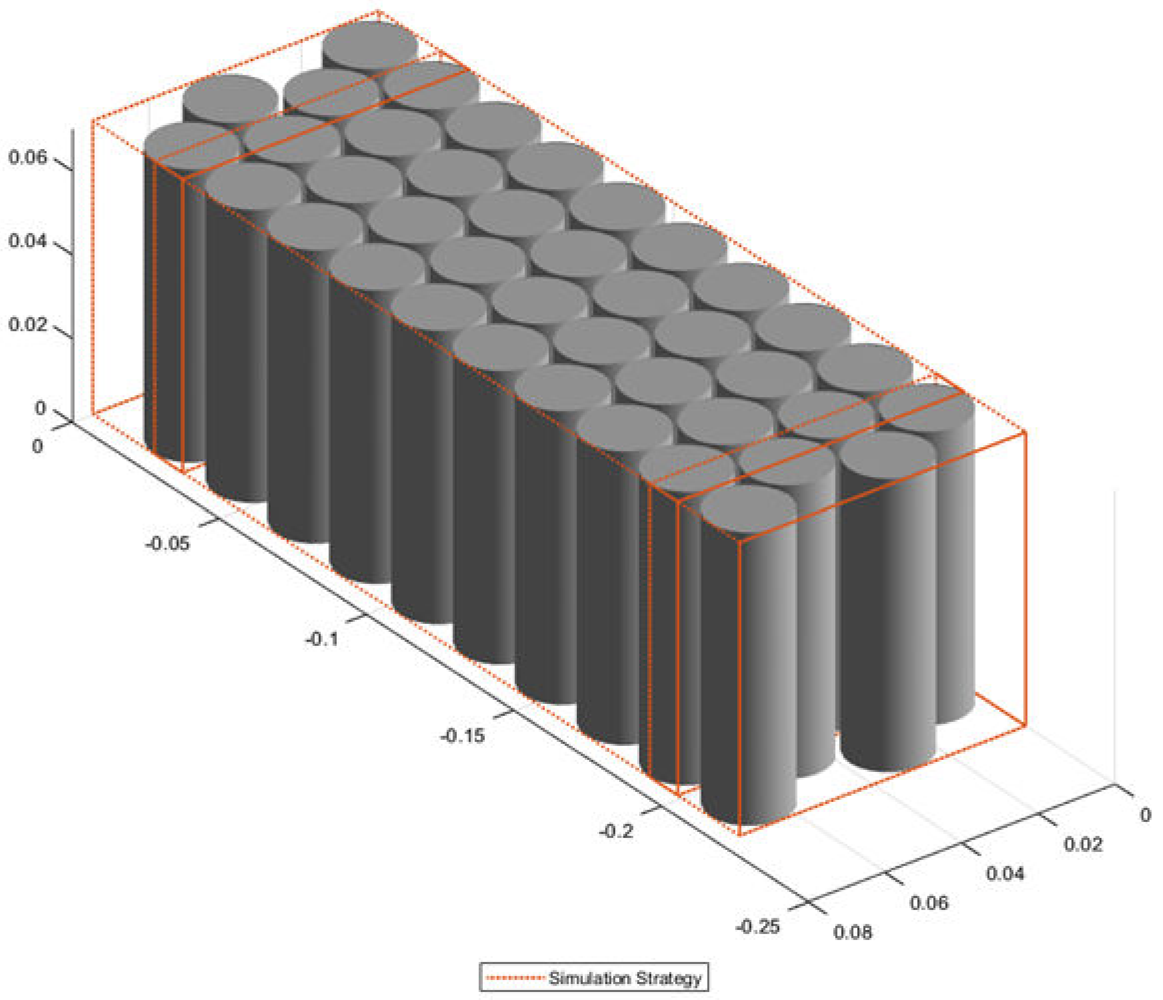
```
batteryModule = Module(ParallelAssembly=pSet,NumSeriesAssemblies=10)
```

Discretize this module in three individual electrical models. The second model comprises eight of the original parallel assemblies of the module.

```
batteryModule.SeriesGrouping = [1 8 1]
```

To verify your modeling resolution, visualize the module by using the `BatteryChart` object and click on the "Show/hide simulation strategy" button on the top-right corner of the plot.

```
moduleChart = BatteryChart(Battery=module);
```



Version History

Introduced in R2022b

See Also

Apps
Battery Builder

Objects
`ParallelAssembly` | `ModuleAssembly` | `BatteryChart`

Functions
`buildBattery`

Simscape Blocks
Module (Generated Block)

Topics
"Battery Modeling Workflow"
"Build Simple Model of Battery Pack in MATLAB and Simscape"
"Build Detailed Model of Battery Pack From Pouch Cells"

ModuleAssembly

Create assembly of battery modules

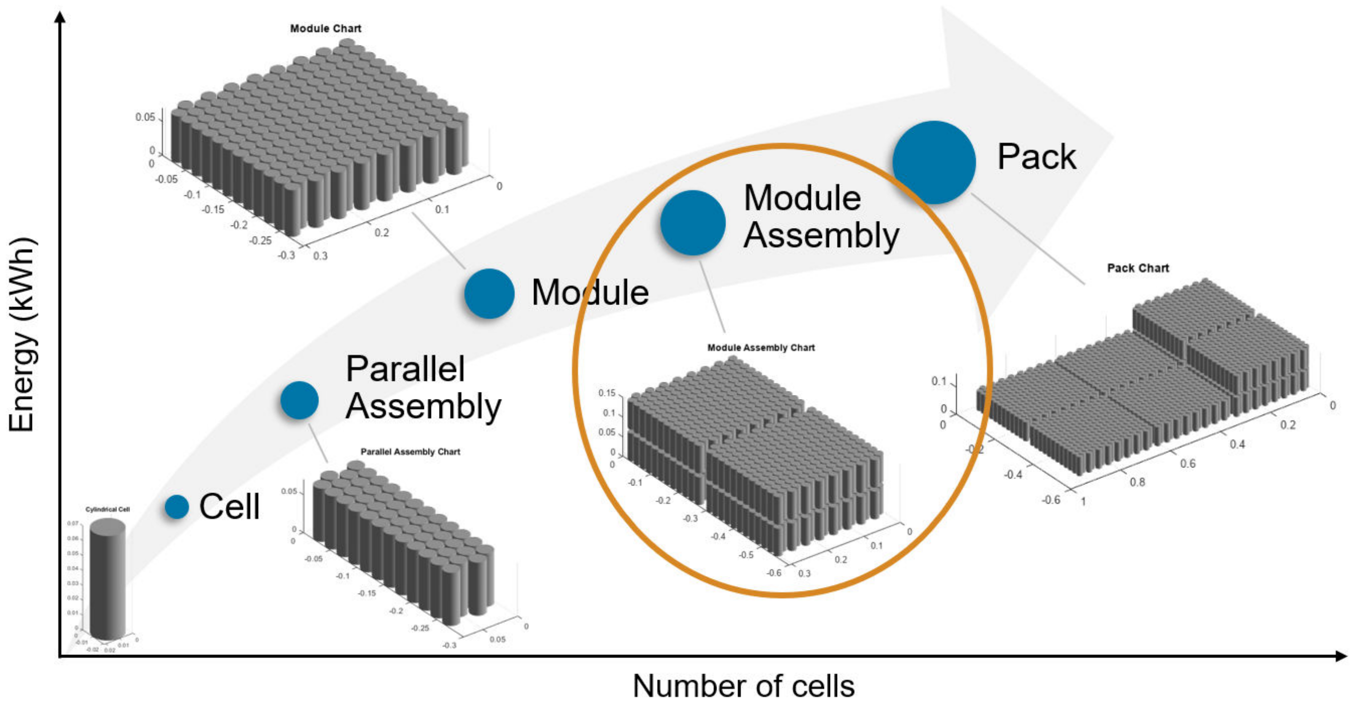
Description

Use `ModuleAssembly` to create a module assembly object that represents a number of `Module` objects connected electrically in series or in parallel. You can use this object as an input to the `Pack` object to create larger battery models.

To generate a Simscape model of your `ModuleAssembly` object, use the `buildBattery` function.

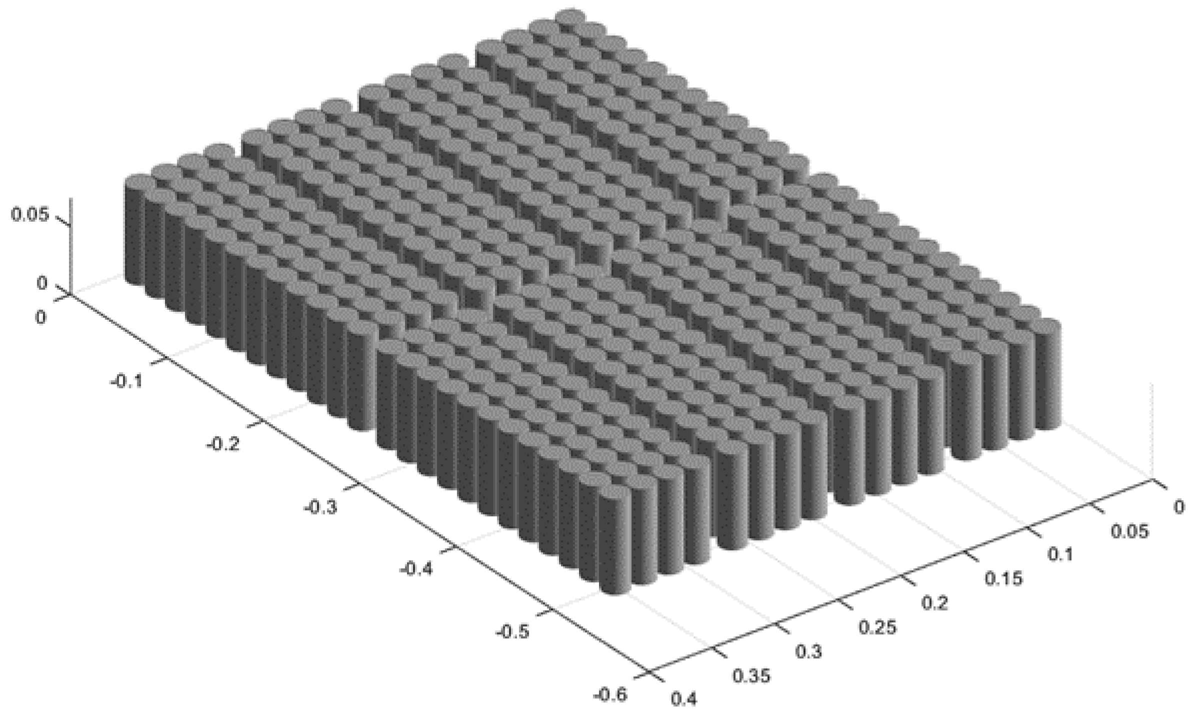
The `ModuleAssembly` object only supports the definition of structural or design parameters. You can modify the run-time parameters for this object and its constituent module models after you create the model. The `ModuleAssembly` object keeps track of its unique constituent module models and generates parameters for these models instead of doing it for every model instance. You can generate a script that contains all the parameters of the `ModuleAssembly` object by specifying the `MaskParameters` argument in the `buildBattery` function. A unique module model is characterized by having the same properties (such as `NumSeriesAssemblies` and `ModelResolution`) and the same constituent cell properties (such as name, format, and weight).

The `ModuleAssembly` object is the fourth element stage of a battery pack system model in a bottom-up approach. Pack models are required for architecture evaluation in early development stages, software and hardware development, system integration and requirement evaluation, cooling system design, control strategy development hardware-in-the-loop, and many more applications.



To specify the number and attributes of the modules, use the `Module` property.

This figure shows a module assembly that comprises two modules made of four parallel assemblies of 48 parallel cylindrical cells each.

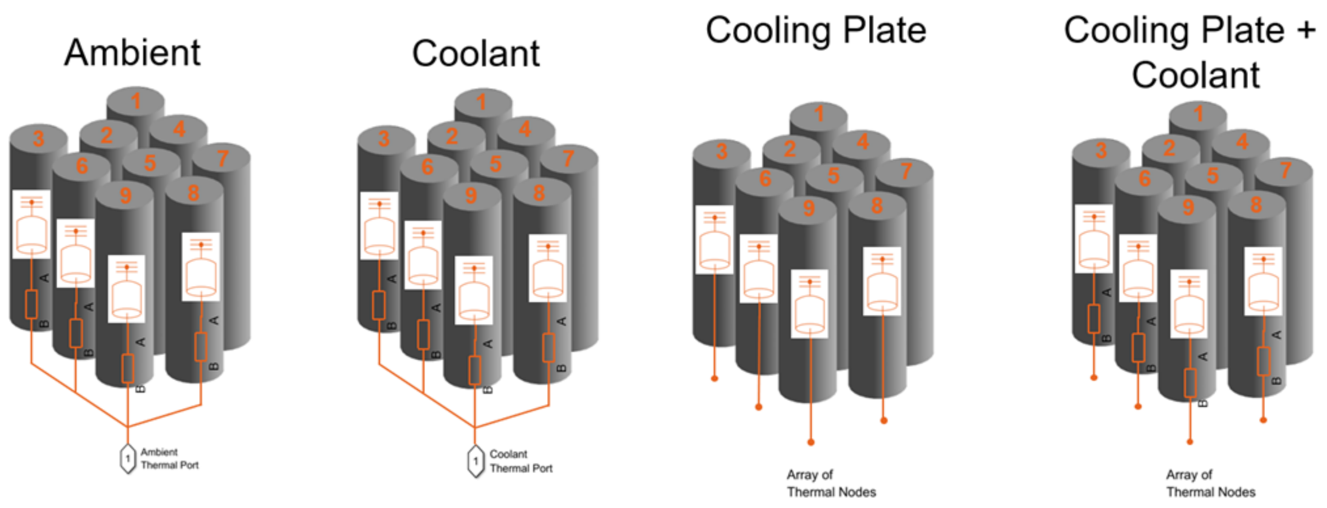


Thermal Boundary Conditions

Thermal boundary conditions define the specific heat transfer mechanisms that occur at each interface of a cell thermal model and its surroundings. In battery systems, cells are typically thermally coupled to different heat sources and sinks, all of which have an effect on the battery cell temperature. The number and type of thermal boundary conditions for a cell model depends on the thermal and mechanical design of the battery system.

For example, you can place cells on an aluminium cooling plate to enhance heat removal and, at the same time, join them together mechanically with a potting compound that effectively eliminates or decreases the inter-cell heat exchange path. The cell temperature has a direct impact on battery performance and lifetime. Therefore, it is crucial to predict this state in dynamic simulation.

Inside a battery object, you can set up a thermal network of lumped-thermal-mass cell models to simultaneously capture the thermal paths to the ambient, the coolant, and/or the cooling plate:

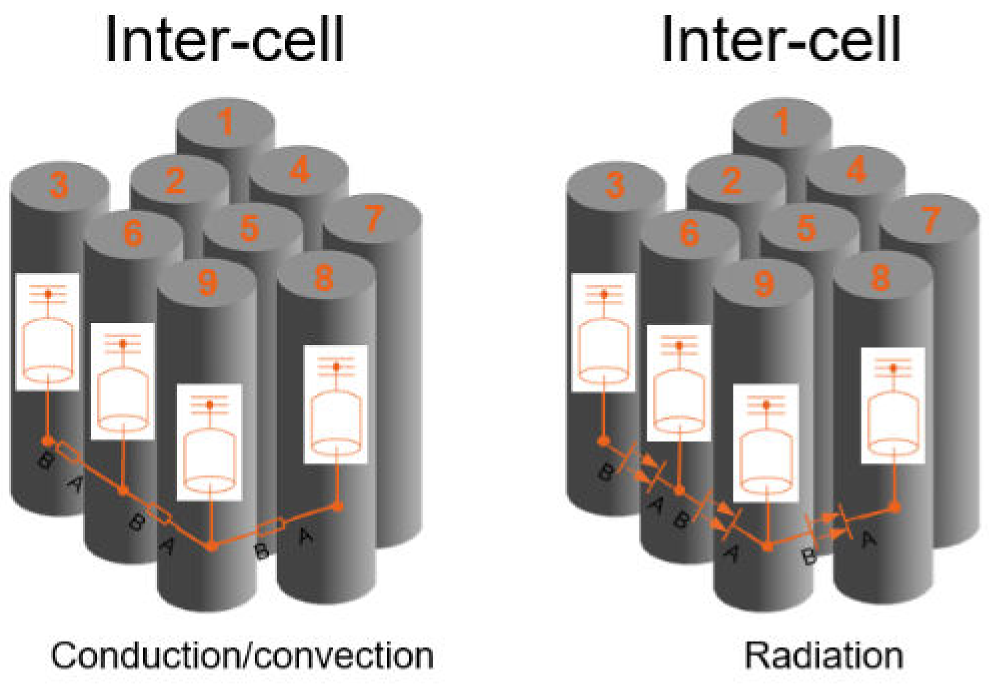


These options are not mutually exclusive. For example, your battery model can combine both the coolant thermal path and the cooling thermal plates to model individual thermal resistances between the individual cells and the sections of the cooling plate.

For more information about thermal paths, see the “AmbientThermalPath” on page 2-0 , “CoolantThermalPath” on page 2-0 , and “CoolingPlate” on page 2-0 properties.

You can also model direct cell-to-cell heat exchange. This is important when you want to simulate more detailed thermal management strategies or even thermal propagation scenarios where inter-cell heat transfer happens at faster rates than ambient or coolant rates. In the battery industry, you can link battery cells to each other through many different means. For example, you can link cylindrical cells by using potting compounds for mechanical rigidity, stability, and thermal isolation, or other types of thermal interface materials. You can also use dielectric fluids or other compounds to heat or cool down cylindrical cells, as well as forced air convection.

You can define the thermal parameters for the inter-cell heat exchange after you create the battery model. You can find these parameters from first principles calculations and more detailed 3D simulations.



These options are not mutually exclusive.

For more information about inter-cell thermal paths, see the “InterCellThermalPath” on page 2-0 and “InterCellRadiativeThermalPath” on page 2-0 properties.

Creation

Syntax

`import simscape.battery.builder.*; batteryModuleAssembly = ModuleAssembly`
`import simscape.battery.builder.*; batteryModuleAssembly = ModuleAssembly(Name=Value)`

Description

`import simscape.battery.builder.*; batteryModuleAssembly = ModuleAssembly` creates a battery module assembly that comprises modules with default property values.

`import simscape.battery.builder.*; batteryModuleAssembly = ModuleAssembly(Name=Value)` sets “Properties” on page 2-58 using one or more name-value arguments. For example, create a battery module assembly with two default modules that are connected in series and stacked along the *y*-axis, with a gap of 0.05 m between each module.

```
batteryModuleAssembly = ModuleAssembly(...
    Module=repmat(Module,1,2), ...
    StackingAxis="Y",...
    InterModuleGap=simscape.Value(0.005,"m"));
```

You can define the number and types of modules in the `Module` property. If your module assembly comprises many modules with the same property values, you can use the repmat function to specify the `Module` property. Otherwise, specify an array of distinct modules.

Properties

`Module` — Set of modules in module assembly
`Module` object (default) | array of `Module` objects

Set of battery modules in the module assembly, specified as a `Module` object or an array of `Module` objects. The `ModuleAssembly` object electrically connects the modules in series or in parallel according to the `CircuitConnection` property. If your module assembly comprises many modules with exactly the same property values, you can use the repmat function to specify this property. Otherwise, specify an array of distinct modules.

Note The array dimensions of the Module input (rows:columns) do not affect how the `ModuleAssembly` object stacks the modules. Only the `StackingAxis` and `NumLevels` properties influence the stacking strategy.

Example: `batteryModule.Module = repmat(Module,1,2)`

Example: `batteryModule.Module = [repmat(module1,1,2),module2]`

`InterModuleGap` — Shortest distance between modules
`simscape.Value(0.001,"m")` (default) | `simscape.Value(positive scalar,"Length unit")` | positive scalar

Shortest distance between modules inside the module assembly, specified as a positive scalar or a `simscape.Value` object that represents a positive scalar with a unit of length. The value of this property must be less than 0.1 m.

If you set this property directly with a positive scalar value instead of using a `simscape.Value` object, this object converts the value to a `simscape.Value` object with meter as its physical unit.

Example: `batteryModuleAssembly.InterModuleGap = simscape.Value(0.01,"m")`

Example: `batteryModuleAssembly.InterModuleGap = 0.01`

`CircuitConnection` — Type of electrical connection
`"Series` (default) | `"Parallel"`

Type of electrical connection between modules, specified as either `"Series` or `"Parallel"`.

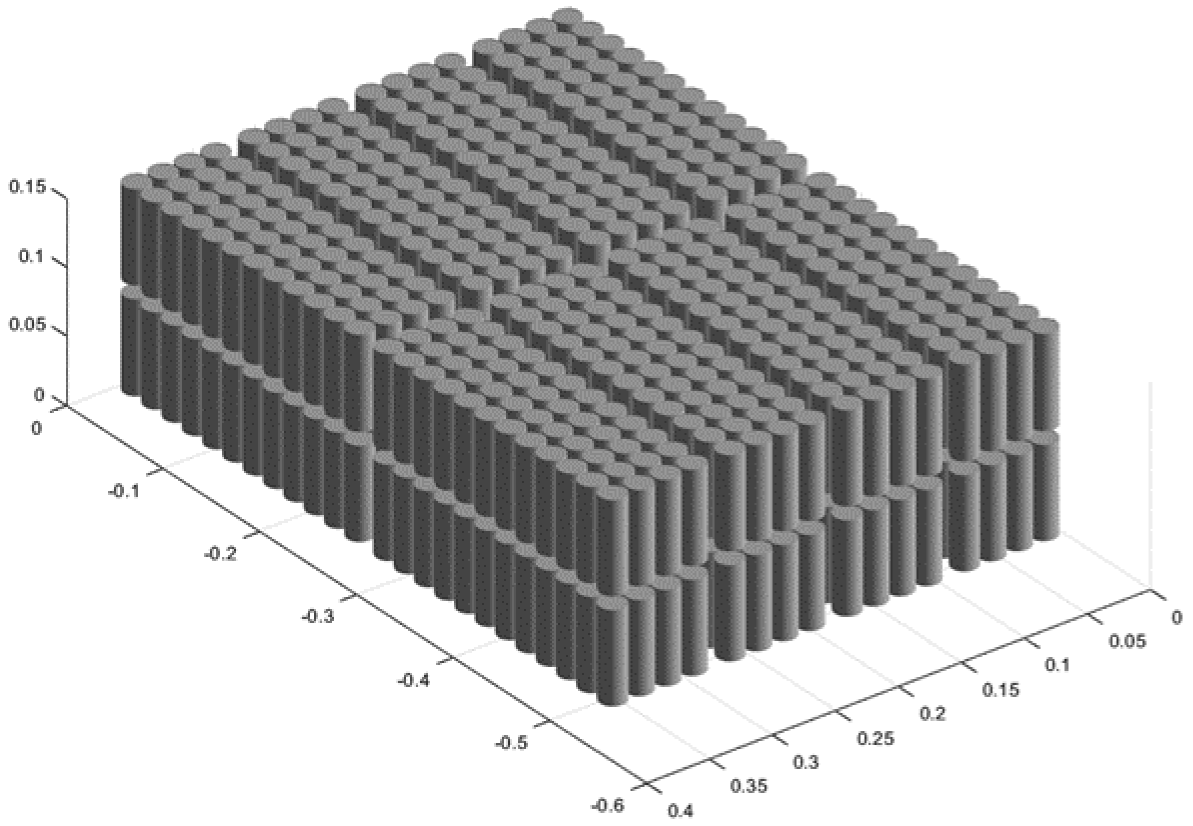
Example: `batteryModuleAssembly.CircuitConnection = "Series"`

`NumLevels` — Number of levels in module assembly
1 (default) | positive integer

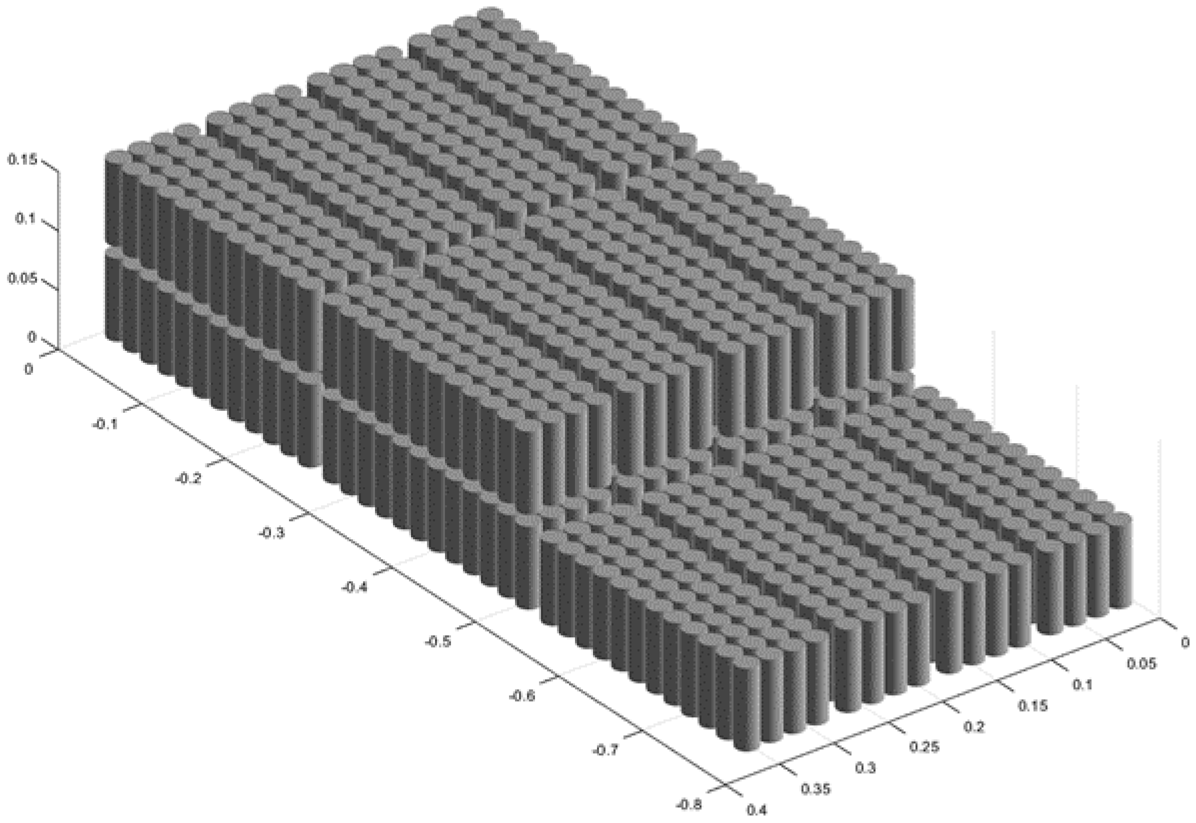
Number of levels, tiers, or floors of the module assembly, specified as a positive integer. The value of this property must be equal to or less than the number of modules in the module assembly.

The `ModuleAssembly` object stacks the modules symmetrically according to the number of levels and modules in the assembly.

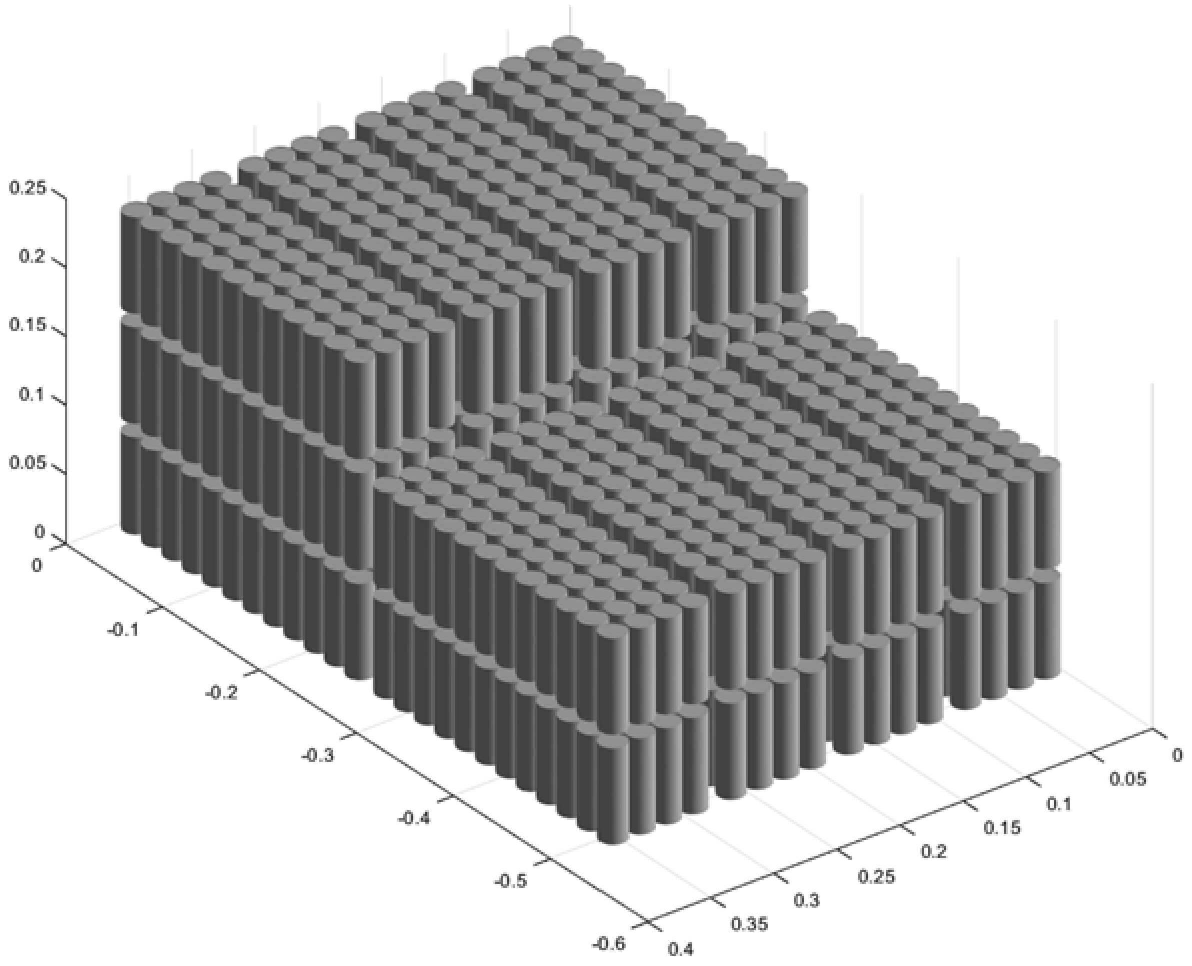
For example, this figure shows how the `ModuleAssembly` object stacks four identical modules when you set this property to `2`.



This figure shows how the `ModuleAssembly` object stacks five identical modules when you still set this property to 2.



This figure shows how the `ModuleAssembly` object stacks five identical modules when you set this property to `3` instead.



Example: `batteryModuleAssembly.NumLevels = 1`

`BalancingStrategy` — State-of-charge balancing strategy
`"None"` (default) | `"Passive"`

State-of-charge balancing strategy for the module assembly, specified as `"None"` or `"Passive"`. Set this property to `"Passive"` to add an ideal balancing circuit connected in parallel to every parallel assembly inside the `ModuleAssembly` Simscape model and a physical port of size equal to `NumSeriesAssemblies*numel(Module)` for switch control.

Note Modifying this property in this object also modifies this property in the Cell, ParallelAssembly, and Module subcomponents of this object.

Example: `batteryModuleAssembly.BalancingStrategy = "Passive"`

`AmbientThermalPath` — Option to use simple thermal resistance block
`"None"` (default) | `"CellBasedThermalResistance"`

Option to use a simple thermal resistance block connected between the cells and a Simscape thermal network, specified as `"CellBasedThermalResistance"` or `"None"`.

Note Setting this property automatically propagates its value to all the subcomponent battery objects inside this ModuleAssembly object. However, this change does not propagate to the other battery objects in your MATLAB workspace.

Example: `batteryModuleAssembly.AmbientThermalPath = "CellBasedThermalResistance"`

`CoolantThermalPath` — Option to use simple thermal resistance block
`"None"` (default) | `"CellBasedThermalResistance"`

Option to use a simple thermal resistance block connected between the cells and a Simscape thermal network, specified as `"CellBasedThermalResistance"` or `"None"`. If you use a cooling plate, the object connects the thermal resistance block between the cells and the cooling plate block by using an array of thermal nodes.

Note Setting this property automatically propagates its value to all the subcomponent battery objects inside this ModuleAssembly object. However, this change does not propagate to the other battery objects in your MATLAB workspace.

Example: `batteryModuleAssembly.CoolantThermalPath = "CellBasedThermalResistance"`

`CoolingPlate` — Option to use cooling plate component
`"None"` (default) | `"Top"` | `"Bottom"`

Option to use a cooling plate component at a specific surface boundary, specified as `"Top"`, `"Bottom"`, or `"None"`.

If you want the Battery Pack Builder to automatically add a cooling plate in your generated model, specify the path of the Cooling Plate block by using the `CoolingPlateBlockPath` property.

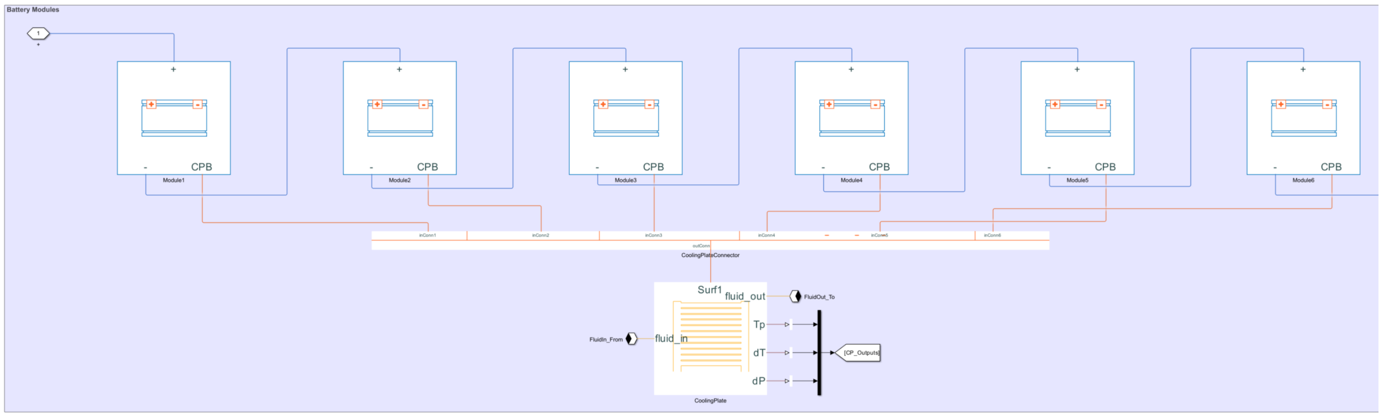
Note Setting this property automatically propagates its value to all the subcomponent battery objects inside this ModuleAssembly object. However, this change does not propagate to the other battery objects in your MATLAB workspace.

Example: `batteryModule.CoolingPlate = "Top"`

`CoolingPlateBlockPath` — Option to specify which cooling plate block to assign
`"None"` (default) | `"batt_lib/Thermal/Edge Cooling"` | `"batt_lib/Thermal/Parallel Channels"` | `"batt_lib/Thermal/U-Shaped Channels"`

Option to specify which cooling plate block you want to assign to the ModuleAssembly object at the boundary defined by the `CoolingPlate` property, specified as `"batt_lib/Thermal/Edge Cooling"`, `"batt_lib/Thermal/Parallel Channels"`, `""batt_lib/Thermal/U-Shaped Channels""`, or `"None"`.

This figure shows the internal structure of a module assembly when you set the `CoolingPlate` property to `"Bottom"`:



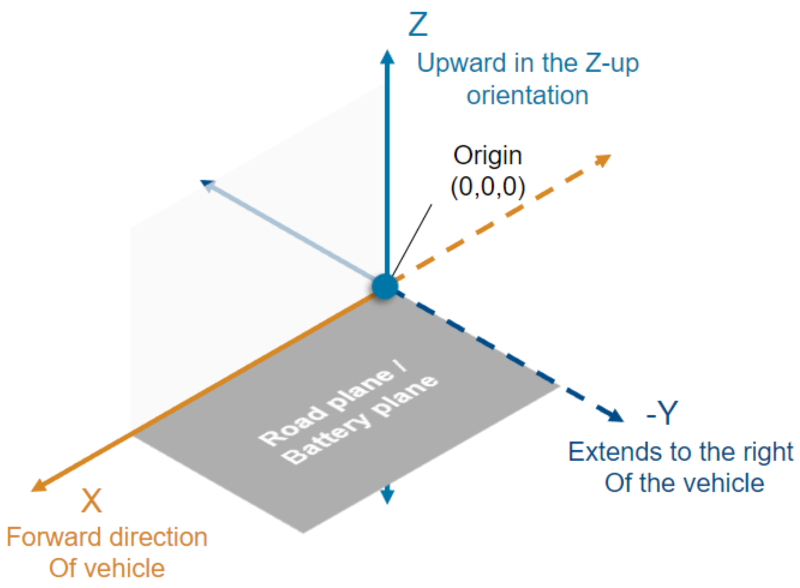
Note Setting this property automatically propagates its value to all the subcomponent battery objects inside this ModuleAssembly object. However, this change does not propagate to the other battery objects in your MATLAB workspace.

Example: `batteryModuleAssembly.CoolingPlateBlockPath = "batt_lib/Thermal/Edge Cooling"`

`StackingAxis` — Preferential stacking direction
`"Y"` (default) | `"X"`

Preferential stacking direction for the arrangement of battery cells in a 2-D Cartesian coordinate system, specified as `"X"` or `"Y"`.

This figure shows the global coordinate system for batteries.


Z
Upward in the Z-up orientation
Origin (0,0,0)
Road plane / Battery plane
-Y
Extends to the right Of the vehicle
X
Forward direction Of vehicle

To plot the module assembly object in the direction of the *x*-axis, set this property to `"X"` before creating the `BatteryChart` object.

Example: `batteryModuleAssembly.StackingAxis = "Y"`

`MassFactor` — Additional non-cell-related mass
`1` (default) | positive double

Additional non-cell-related mass added to the module assembly by components such as busbars, tabs, and collector plates, specified as a strictly positive double greater than or equal to 1.

Example: `batteryModuleAssembly.MassFactor = 1.2`

`Position` — Location of battery object
`[0 0 0]` (default) | vector of real and finite entries

Location of the battery object in a 3-D Cartesian coordinate system, specified as a vector of real and finite entries.

Example: `batteryModuleAssembly.Position = [0 0 0]`

`Name` — Name of module assembly
`"ModuleAssembly1"` (default) | string

Name of the battery module assembly, specified as a string.

Example: `batteryModuleAssembly.Name = "ModuleAssembly2"`

`NonCellResistance` — Option to use electrical resistance blocks
`'off'` (default) | on/off logical value

Option to use electrical resistance blocks to represent additional electrical resistances from non-cell components, specified as `'off'`, `'on'`, or as numeric or logical `1` (`true`) or `0` (`false`). A value of `'on'` is equivalent to `true`, and `'off'` is equivalent to `false`. Thus, you can use the value of this property as a logical value. The value is stored as an on/off logical value of type `matlab.lang.OnOffSwitchState`.

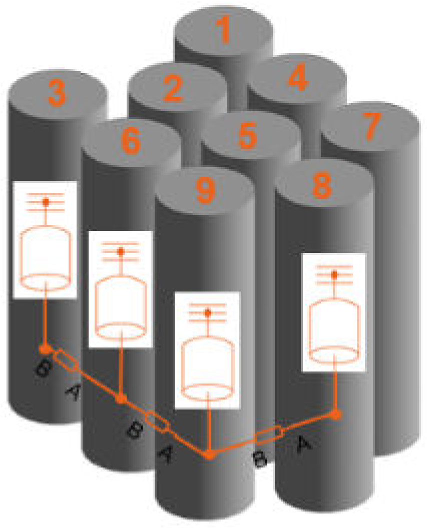
Example: `batteryModuleAssembly.NonCellResistance = 'on'`

`InterCellThermalPath` — Option to use simple thermal resistance block for inter cell heat
`'off'` (default) | on/off logical value

Option to use thermal resistance blocks to represent a linear cell-to-cell heat transfer path between adjacent battery cells, specified as `'off'`, `'on'`, or as numeric or logical `1` (`true`) or `0` (`false`). A value of `'on'` is equivalent to `true`, and `'off'` is equivalent to `false`. Thus, you can use the value of this property as a logical value. The value is stored as an on/off logical value of type `matlab.lang.OnOffSwitchState`.

Enabling the inter-cell thermal path is useful only when you have selected a cell thermal model and more than one cell or cell model exists inside the battery.

You can define the value for the thermal resistance parameter after model creation.



If you set this property to `'on'`, you cannot set the **InterCellRadiativeThermalPath** property to `'on'`.

Note Setting this property automatically propagates its value to all the subcomponent battery objects inside this ModuleAssembly object. However, this change does not propagate to the other battery objects in your MATLAB workspace.

Example: `batteryModuleAssembly.InterCellThermalPath = "on"`

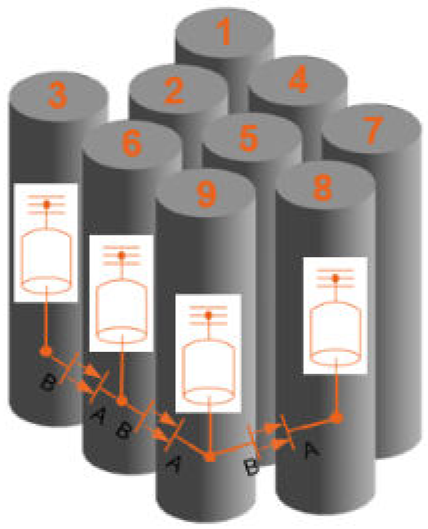
`InterCellRadiativeThermalPath` — Option to use simple thermal resistance block for inter cell heat

`'off'` (default) | on/off logical value

Option to use thermal resistance blocks to represent a radiative cell-to-cell heat transfer path between adjacent battery cells, specified as `'off'`, `'on'`, or as numeric or logical `1` (`true`) or `0` (`false`). A value of `'on'` is equivalent to `true`, and `'off'` is equivalent to `false`. Thus, you can use the value of this property as a logical value. The value is stored as an on/off logical value of type `matlab.lang.OnOffSwitchState`.

Enabling the inter-cell radiative thermal path is useful only when you have selected a cell thermal model and more than one cell or cell model exists inside the battery.

You can define the value for the radiation heat transfer parameters after model creation.



If you set this property to `'on'`, you cannot set the **InterCellThermalPath** property to `'on'`.

Note Setting this property automatically propagates its value to all the subcomponent battery objects inside this ModuleAssembly object. However, this change does not propagate to the other battery objects in your MATLAB workspace.

Example: `batteryModuleAssembly.InterCellRadiativeThermalPath = "on"`

`PackagingVolume` — Volume of battery

`simscape.Value([],"m^3")` (default) | `simscape.Value` object

This property is read-only.

Volume of the battery, returned as a `simscape.Value` object with a unit of volume.

`CumulativeMass` — Cumulative mass of battery

`simscape.Value` object

This property is read-only.

Cumulative mass of the battery, returned as a `simscape.Value` object with a unit of mass.

`CumulativeCellCapacity` — Cumulative cell capacity of battery

`simscape.Value` object

This property is read-only.

Cumulative cell capacity of the battery, returned as a `simscape.Value` object with unit of current multiplied by time.

The value of this property is equal to the individual cell capacity multiplied by the number of cells electrically connected in parallel inside this battery object.

`CumulativeCellEnergy` — Cumulative cell energy of battery

`simscape.Value` object

This property is read-only.

Cumulative cell energy of the battery, returned as a `simscape.Value` object with unit of energy.

The value of this property is equal to the individual cell energy multiplied by the total number of cells inside this battery object.

`NumModels` — Number of cell model blocks
double

This property is read-only.

Number of cell model blocks in simulation, returned as a double.

`CellNumbering` — Numbering for all cells
structure

This property is read-only.

Numbering for all of the cells inside of the battery, returned as a structure.

`ThermalNodes` — Vectorized thermal node information
structure

This property is read-only.

Vectorized thermal node information for external boundary conditions, returned as a structure. This information comprise XY location, XY dimensions, and number of thermal nodes.

Type — Type of battery
`"ModuleAssembly"`

This property is read-only.

Type of battery, returned as `"ModuleAssembly"`.

Examples

Create Pouch Cell Module Assembly with Four Series Modules

Create a `Cell` object with a pouch geometry.

```
batteryCell = Cell(Geometry=PouchGeometry)
```

Create a `ParallelAssembly` object of three parallel cells with the default topology .

```
pSet = ParallelAssembly(Cell=batteryCell,NumParallelCells=3)
```

Use this `ParallelAssembly` object to create a `Module` object of 10 parallel assemblies connected in series.

```
batteryModule = Module(ParallelAssembly=pSet,NumSeriesAssemblies=10)
```

Use this `batteryModule` object to create a `ModuleAssembly` object of four identical modules connected in series.

```
batteryModuleAssembly = ModuleAssembly(Module=repmat(batteryModule,1,4))
```

Visualize the module assembly by using a `BatteryChart` object.

```
moduleAssemblyChart = BatteryChart(Battery=batteryModuleAssembly);
```

Create Module Assembly with Single Bottom Cooling Plate

Create a `Cell` object with a Cylindrical geometry and enable the modeling of its thermal effects.

```
battCell = Cell(Geometry = CylindricalGeometry);
battCell.CellModelOptions.BlockParameters.thermal_port = "model";
```

Create a `ParallelAssembly` object of 30 parallel cells in five rows with the default topology.

```
pSet = ParallelAssembly(Cell=battCell, NumParallelCells = 30, Rows = 5);
```

Use this `ParallelAssembly` object to create a `Module` object of two parallel assemblies connected in series.

```
module = Module(ParallelAssembly=pSet, NumSeriesAssemblies = 2);
```

Use this `Module` object to create a `ModuleAssembly` object of six identical modules connected in series.

```
moduleAssembly = ModuleAssembly(Module = repmat(module,1,6));
```

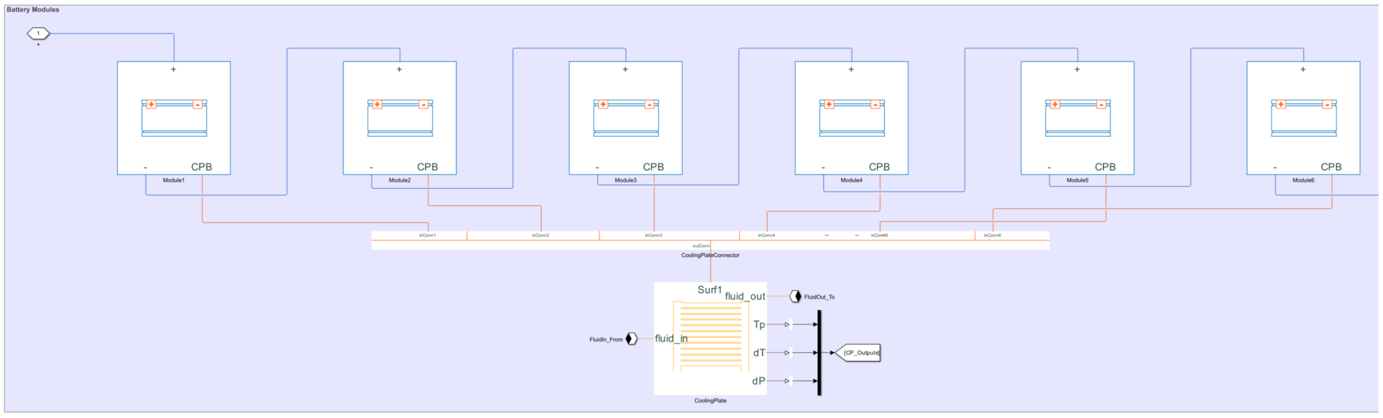
Assign a single cooling plate to the bottom of the module assembly and choose the Parallel Channels block as the cooling plate block.

```
moduleAssembly.CoolingPlate = "Bottom";
moduleAssembly.CoolingPlateBlockPath = "batt_lib/Thermal/Parallel Channels";
```

Build this `ModuleAssembly` object by calling the `buildBattery` function.

```
buildBattery(moduleAssembly, "LibraryName","Permutation1_SingleBottomCoolingPlate",...
"MaskParameters","VariableNames","MaskInitialTargets","VariableNames");
```

This figure shows the internal structure of the `ModuleAssembly` object:



Create Module Assembly with Module-Specific Bottom Cooling Plates

Create a `Cell` object with a Cylindrical geometry and enable the modeling of its thermal effects.

```
battCell = Cell(Geometry = CylindricalGeometry);
battCell.CellModelOptions.BlockParameters.thermal_port = "model";
```

Create a `ParallelAssembly` object of 30 parallel cells in five rows with the default topology.

```
pSet = ParallelAssembly(Cell=battCell, NumParallelCells = 30, Rows = 5);
```

Use this `ParallelAssembly` object to create a `Module` object of two parallel assemblies connected in series.

```
module = Module(ParallelAssembly=pSet, NumSeriesAssemblies = 2);
```

Use this `Module` object to create a `ModuleAssembly` object of six identical modules connected in series.

```
moduleAssembly = ModuleAssembly(Module = repmat(module,1,6));
```

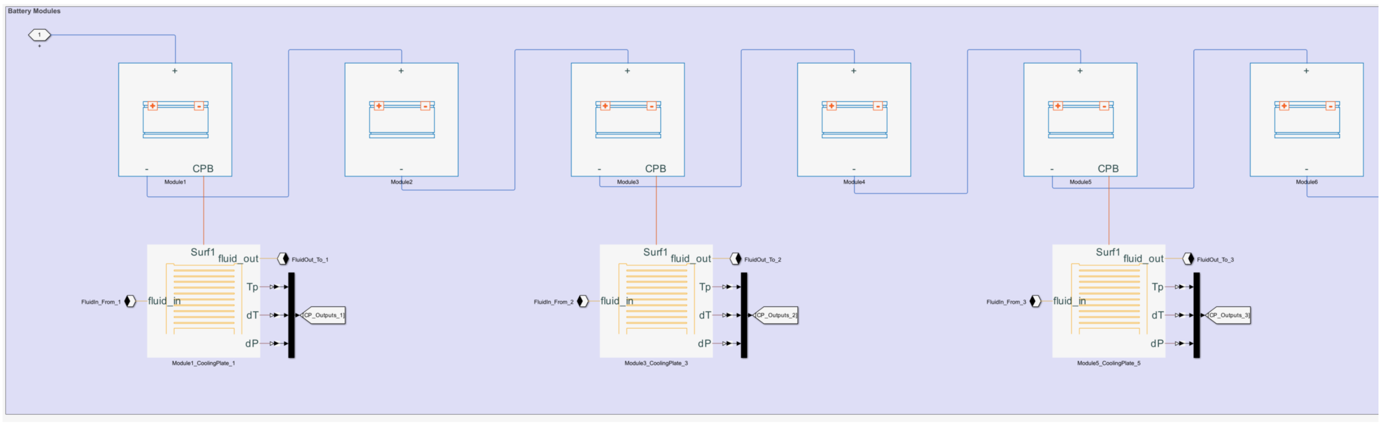
Assign a cooling plate to the bottom of the first, third, and fifth modules inside the module assembly and choose the Parallel Channels block as the cooling plate block.

```
moduleAssembly.Module(1).CoolingPlate = "Bottom";
moduleAssembly.Module(1).CoolingPlateBlockPath = "batt_lib/Thermal/Parallel Channels";
moduleAssembly.Module(3).CoolingPlate = "Bottom";
moduleAssembly.Module(3).CoolingPlateBlockPath = "batt_lib/Thermal/Parallel Channels";
moduleAssembly.Module(5).CoolingPlate = "Bottom";
moduleAssembly.Module(5).CoolingPlateBlockPath = "batt_lib/Thermal/Parallel Channels";
```

Build this `ModuleAssembly` object by calling the `buildBattery` function.

```
buildBattery(moduleAssembly, "LibraryName","Permutation2_ModuleSpecificBottomCoolingPlate",...
"MaskParameters","VariableNames","MaskInitialTargets","VariableNames");
```

This figure shows the internal structure of the `ModuleAssembly` object:



Create Module Assembly with Single Top Cooling Plate

Create a `Cell` object with a Cylindrical geometry and enable the modeling of its thermal effects.

```
battCell = Cell(Geometry = CylindricalGeometry);
battCell.CellModelOptions.BlockParameters.thermal_port = "model";
```

Create a `ParallelAssembly` object of 30 parallel cells in five rows with the default topology.

```
pSet = ParallelAssembly(Cell=battCell, NumParallelCells = 30, Rows = 5);
```

Use this `ParallelAssembly` object to create a `Module` object of two parallel assemblies connected in series.

```
module = Module(ParallelAssembly=pSet, NumSeriesAssemblies = 2);
```

Use this `Module` object to create a `ModuleAssembly` object of six identical modules connected in series.

```
moduleAssembly = ModuleAssembly(Module = repmat(module,1,6));
```

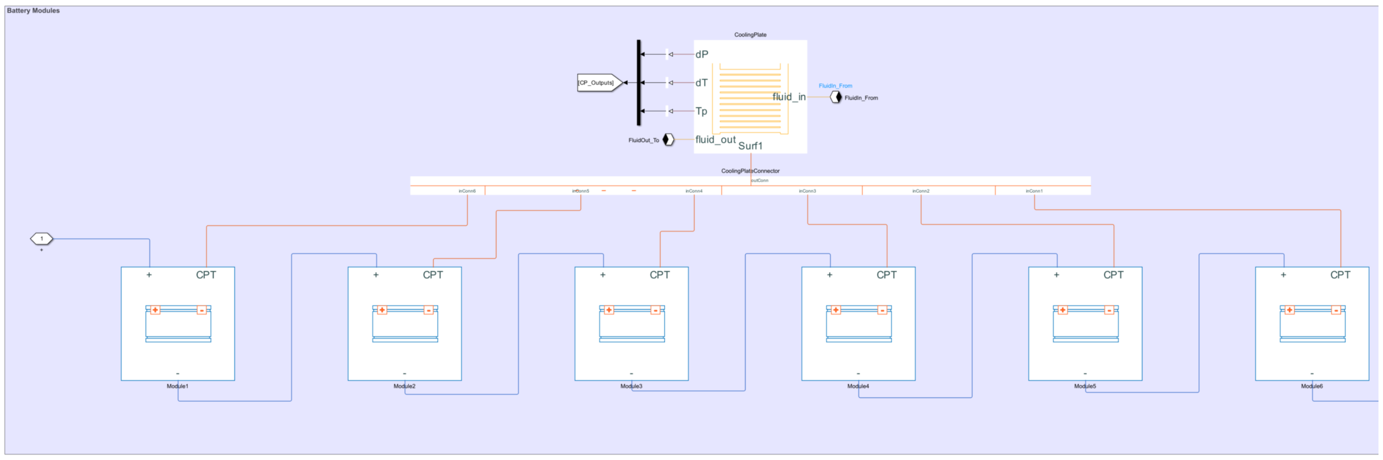
Assign a single cooling plate to the top of the module assembly and choose the Parallel Channels block as the cooling plate block.

```
moduleAssembly.CoolingPlate = "Top";
moduleAssembly.CoolingPlateBlockPath = "batt_lib/Thermal/Parallel Channels";
```

Build this `ModuleAssembly` object by calling the `buildBattery` function.

```
buildBattery(moduleAssembly, "LibraryName","Permutation3_SingleTopCoolingPlate",...
"MaskParameters","VariableNames","MaskInitialTargets","VariableNames");
```

This figure shows the internal structure of the `ModuleAssembly` object:



Create Module Assembly with Module-Specific Top Cooling Plates

Create a `Cell` object with a Cylindrical geometry and enable the modeling of its thermal effects.

```
battCell = Cell(Geometry = CylindricalGeometry);
battCell.CellModelOptions.BlockParameters.thermal_port = "model";
```

Create a `ParallelAssembly` object of 30 parallel cells in five rows with the default topology.

```
pSet = ParallelAssembly(Cell=battCell, NumParallelCells = 30, Rows = 5);
```

Use this `ParallelAssembly` object to create a `Module` object of two parallel assemblies connected in series.

```
module = Module(ParallelAssembly=pSet, NumSeriesAssemblies = 2);
```

Use this `Module` object to create a `ModuleAssembly` object of six identical modules connected in series.

```
moduleAssembly = ModuleAssembly(Module = repmat(module,1,6));
```

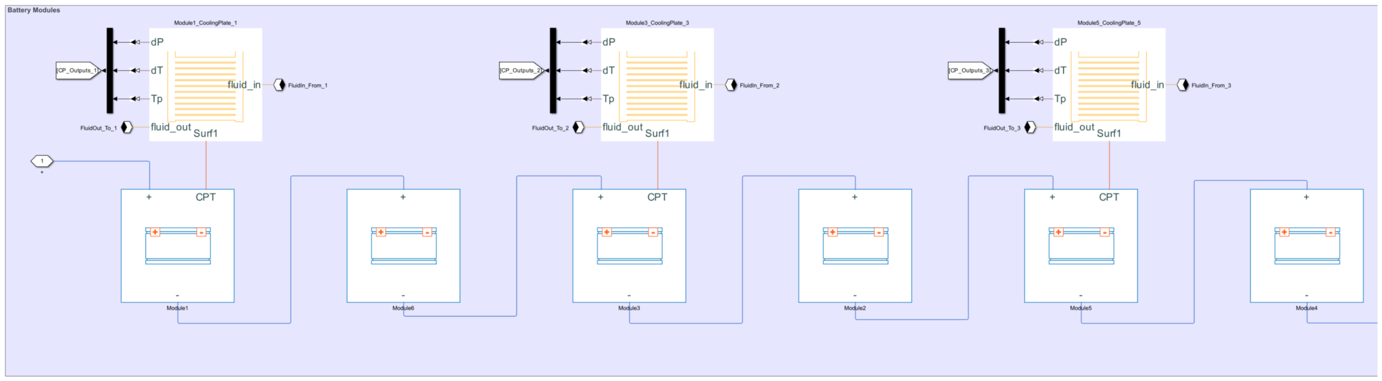
Assign a cooling plate to the top of the first, third, and fifth modules inside the module assembly and choose the Parallel Channels block as the cooling plate block.

```
moduleAssembly.Module(1).CoolingPlate = "Top";
moduleAssembly.Module(1).CoolingPlateBlockPath = "batt_lib/Thermal/Parallel Channels";
moduleAssembly.Module(3).CoolingPlate = "Top";
moduleAssembly.Module(3).CoolingPlateBlockPath = "batt_lib/Thermal/Parallel Channels";
moduleAssembly.Module(5).CoolingPlate = "Top";
moduleAssembly.Module(5).CoolingPlateBlockPath = "batt_lib/Thermal/Parallel Channels";
```

Build this `ModuleAssembly` object by calling the `buildBattery` function.

```
buildBattery(moduleAssembly, "LibraryName","Permutation4_ModuleSpecificTopCoolingPlate",...
"MaskParameters","VariableNames","MaskInitialTargets","VariableNames");
```

This figure shows the internal structure of the `ModuleAssembly` object:



Version History

Introduced in R2022b

See Also

Apps
Battery Builder

Objects
`Module` | `Pack` | `BatteryChart`

Functions
`buildBattery`

Simscape Blocks
ModuleAssembly (Generated Block)

Topics
“Battery Modeling Workflow”
“Build Simple Model of Battery Pack in MATLAB and Simscape”
“Build Detailed Model of Battery Pack From Pouch Cells”

Pack

Create pack of module assemblies

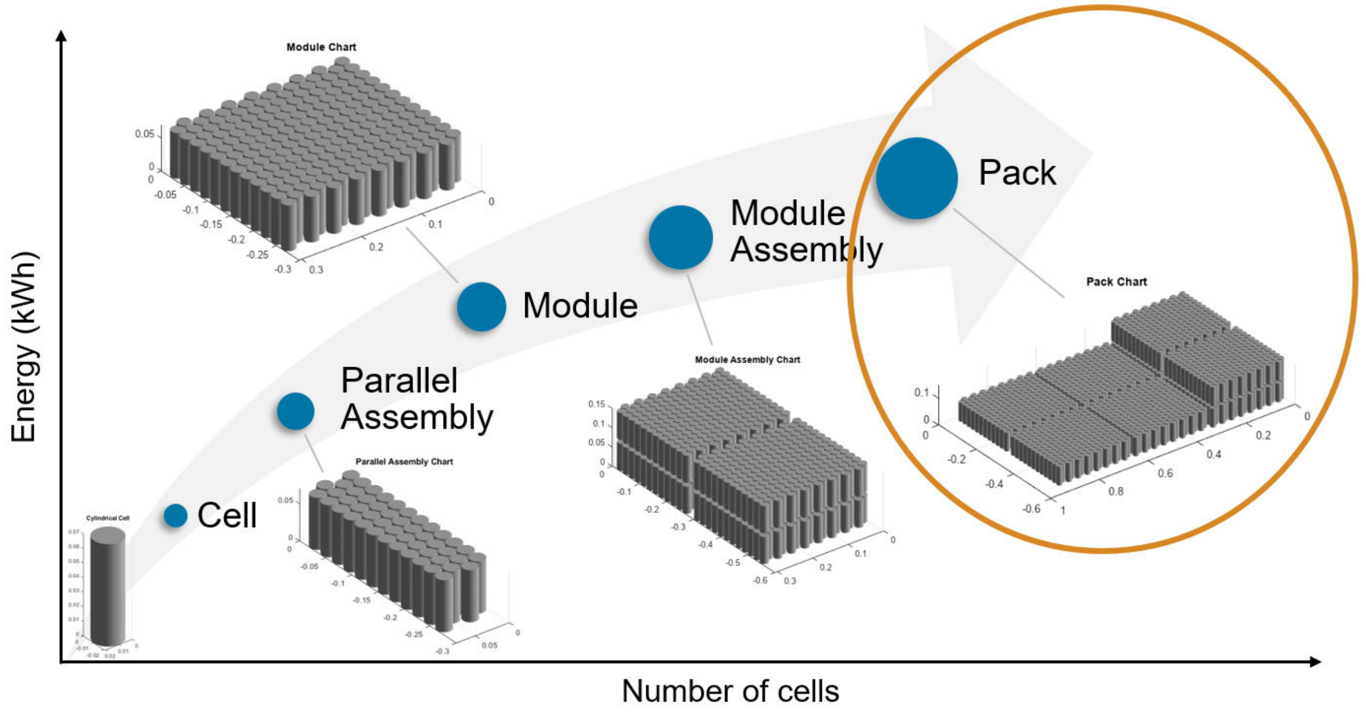
Description

Use `Pack` to create a battery pack object that represents a number of `ModuleAssembly` objects connected electrically in series or in parallel.

To generate a Simscape model of your `Pack` object, use the `buildBattery` function.

The `Pack` object only supports the definition of structural or design parameters. You can modify the run-time parameters for this object and for the models of its constituent module assemblies and modules after you create the model. The `Pack` object keeps track of its unique constituent module models and generates properties for these models instead of doing it for every model instance. You can generate a script that contains all the parameters of the `Pack` object by specifying the `MaskParameters` argument in the `buildBattery` function. A unique pack model is characterized by having the same properties (such as `NumSeriesAssemblies` and `ModelResolution`) and the same constituent cell properties (such as name, format, and weight).

The `Pack` object is the final stage of a battery pack system model in a bottom-up approach. Pack models are required for architecture evaluation in early development stages, software and hardware development, system integration and requirement evaluation, cooling system design, control strategy development hardware-in-the-loop, and many more applications.

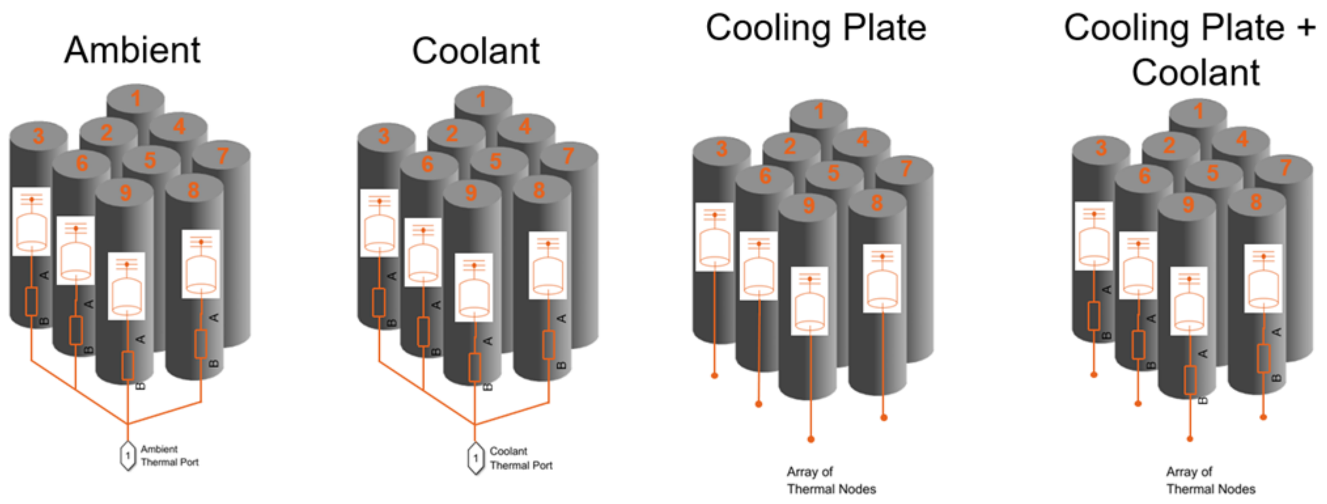


Thermal Boundary Conditions

Thermal boundary conditions define the specific heat transfer mechanisms that occur at each interface of a cell thermal model and its surroundings. In battery systems, cells are typically thermally coupled to different heat sources and sinks, all of which have an effect on the battery cell temperature. The number and type of thermal boundary conditions for a cell model depends on the thermal and mechanical design of the battery system.

For example, you can place cells on an aluminium cooling plate to enhance heat removal and, at the same time, join them together mechanically with a potting compound that effectively eliminates or decreases the inter-cell heat exchange path. The cell temperature has a direct impact on battery performance and lifetime. Therefore, it is crucial to predict this state in dynamic simulation.

Inside a battery object, you can set up a thermal network of lumped-thermal-mass cell models to simultaneously capture the thermal paths to the ambient, the coolant, and/or the cooling plate:

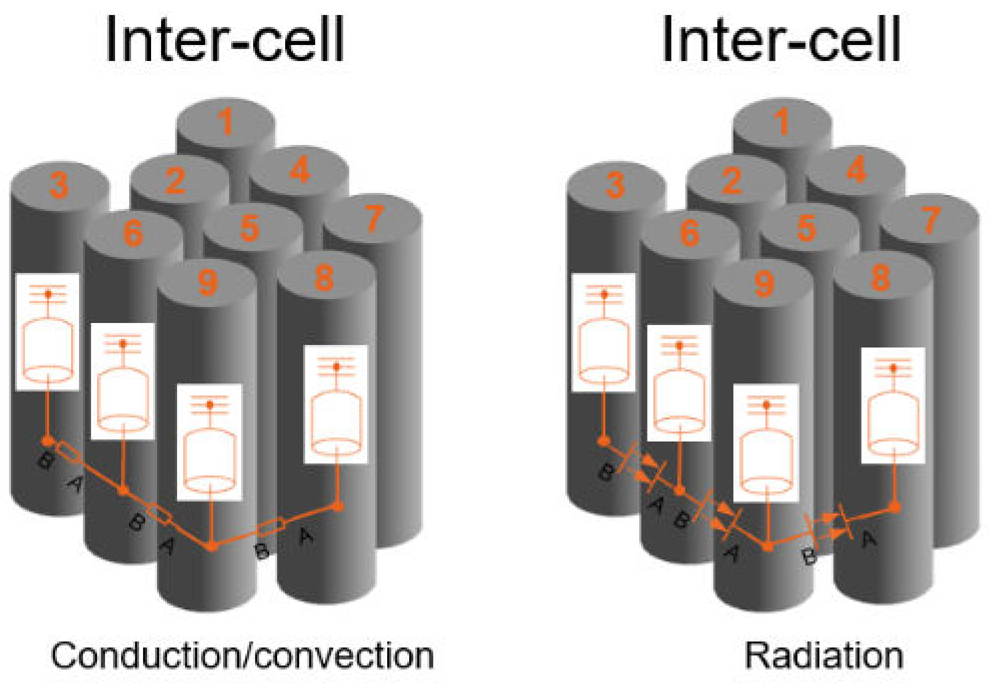


These options are not mutually exclusive. For example, your battery model can combine both the coolant thermal path and the cooling thermal plates to model individual thermal resistances between the individual cells and the sections of the cooling plate.

For more information about thermal paths, see the “AmbientThermalPath” on page 2-0 , “CoolantThermalPath” on page 2-0 , and “CoolingPlate” on page 2-0 properties.

You can also model direct cell-to-cell heat exchange. This is important when you want to simulate more detailed thermal management strategies or even thermal propagation scenarios where inter-cell heat transfer happens at faster rates than ambient or coolant rates. In the battery industry, you can link battery cells to each other through many different means. For example, you can link cylindrical cells by using potting compounds for mechanical rigidity, stability, and thermal isolation, or other types of thermal interface materials. You can also use dielectric fluids or other compounds to heat or cool down cylindrical cells, as well as forced air convection.

You can define the thermal parameters for the inter-cell heat exchange after you create the battery model. You can find these parameters from first principles calculations and more detailed 3D simulations.



These options are not mutually exclusive.

For more information about inter-cell thermal paths, see the “InterCellThermalPath” on page 2-0 and “InterCellRadiativeThermalPath” on page 2-0 properties.

Creation

Syntax

```
import simscape.battery.builder.*; batteryPack = Pack
import simscape.battery.builder.*; batteryPack = Pack(Name=Value)
```

Description

`import simscape.battery.builder.*; batteryPack = Pack` creates a battery pack that comprises module assemblies with default property values.

`import simscape.battery.builder.*; batteryPack = Pack(Name=Value)` sets “Properties” on page 2-76 using one or more name-value arguments. For example, create a pack with four default module assemblies connected in series, stacked along the *x*-axis, and with a gap between the module assemblies equal to 0.005 m.

```
batteryPack = Pack(...
    ModuleAssembly=repmat(ModuleAssembly=ModuleAssembly,1,4), ...
    StackingAxis="X",...
    InterModuleAssemblyGap=simscape.Value(0.005,"m"));
```

You can define the number and types of module assemblies in the `ModuleAssembly` property. If your pack comprises many module assemblies with exactly the same property values, you can use the repmat function to specify the `ModuleAssembly` property. Otherwise, specify an array of distinct module assemblies.

Properties

`ModuleAssembly` — Set of module assemblies in pack
`ModuleAssembly` object (default) | array of `ModuleAssembly` objects

Set of battery module assemblies in the battery pack, specified as a `ModuleAssembly` object or an array of `ModuleAssembly` objects. The `Pack` object electrically connects the module assemblies in series or in parallel according to the `CircuitConnection` property. If your pack comprises many module assemblies with the same property values, you can use the repmat function to specify this property. Otherwise, specify an array of distinct module assemblies.

Note This property does not affect how the `Pack` object stacks the modules. Only the `StackingAxis` property defines the stacking strategy.

Example: `batteryPack.ModuleAssembly = repmat(ModuleAssembly,1,2)`

Example: `batteryPack.ModuleAssembly = [repmat(moduleAssembly1,1,2),moduleAssembly2]`

`InterModuleAssemblyGap` — Shortest distance between module assemblies
`simscape.Value(0.001,"m")` (default) | `simscape.Value(positive scalar,"Length unit")` | positive scalar

Shortest distance between module assemblies inside the battery pack, specified as a positive scalar or a `simscape.Value` object that represents a positive scalar with a unit of length. The value of this property must be less than `0.1 m`.

If you set this property directly with a positive scalar value instead of using a `simscape.Value` object, this object converts the value to a `simscape.Value` object with meter as its physical unit.

Example: `batteryPack.InterModuleAssemblyGap = simscape.Value(0.01,"m")`

Example: `batteryPack.InterModuleAssemblyGap = 0.01`

`BalancingStrategy` — State-of-charge cell balancing strategy
`"None"` (default) | `"Passive`

State-of-charge cell balancing strategy for the pack, specified as `"None"` or `"Passive"`.

Note Setting this property automatically propagates its value to all the subcomponent battery objects inside this Pack object. However, this change does not propagate to the other battery objects in your MATLAB workspace.

Example: `batteryPack.BalancingStrategy = "Passive"`

`AmbientThermalPath` — Option to use simple thermal resistance block
`"None"` (default) | `"CellBasedThermalResistance"`

Option to use a simple thermal resistance block connected between the cells and a Simscape thermal network, specified as `"CellBasedThermalResistance"` or `"None"`.

Note Setting this property automatically propagates its value to all the subcomponent battery objects inside this Pack object. However, this change does not propagate to the other battery objects in your MATLAB workspace.

Example: `batteryPack.AmbientThermalPath = "CellBasedThermalResistance"`

`CoolantThermalPath` — Option to use simple thermal resistance block

`"None"` (default) | `"CellBasedThermalResistance"`

Option to use a simple thermal resistance block connected between the cells and a Simscape thermal network, specified as `"CellBasedThermalResistance"` or `"None"`. If you use a cooling plate, the object connects the thermal resistance block between the cells and the cooling plate block by using an array of thermal nodes.

Note Setting this property automatically propagates its value to all the subcomponent battery objects inside this Pack object. However, this change does not propagate to the other battery objects in your MATLAB workspace.

Example: `batteryPack.CoolantThermalPath = "CellBasedThermalResistance"`

`CoolingPlate` — Option to use cooling plate component

`"None"` (default) | `"Top"` | `"Bottom"`

Option to use a cooling plate component at a specific surface boundary, specified as `"Top"`, `"Bottom"`, or `"None"`.

If you want the Battery Pack Builder to automatically add a cooling plate in your generated model, specify the path of the Cooling Plate block by using the `CoolingPlateBlockPath` property.

Note Setting this property automatically propagates its value to all the subcomponent battery objects inside this Pack object. However, this change does not propagate to the other battery objects in your MATLAB workspace.

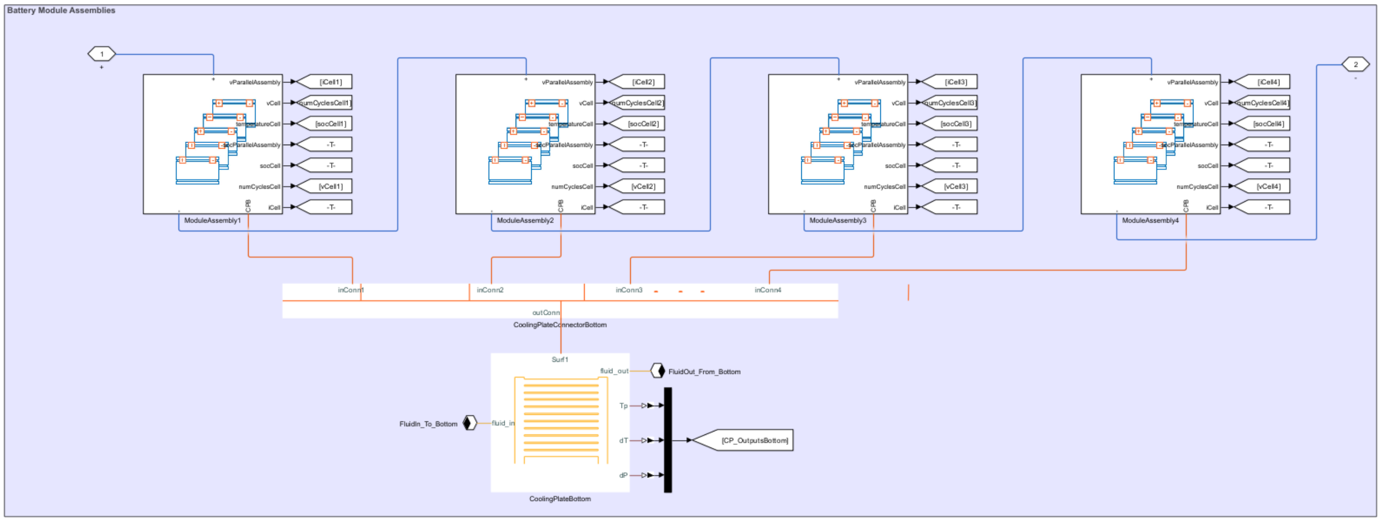
Example: `batteryPack.CoolingPlate = "Top"`

`CoolingPlateBlockPath` — Option to specify which cooling plate block to assign

`"None"` (default) | `"batt_lib/Thermal/Edge Cooling"` | `"batt_lib/Thermal/Parallel Channels"` | `"batt_lib/Thermal/U-Shaped Channels"`

Option to specify which cooling plate block you want to assign to the Pack object at the boundary defined by the `CoolingPlate` property, specified as `"batt_lib/Thermal/Edge Cooling"`, `"batt_lib/Thermal/Parallel Channels"`, `""batt_lib/Thermal/U-Shaped Channels""`, or `"None"`.

This figure shows the internal structure of a pack when you set the `CoolingPlate` property to `"Bottom"`:



Note Setting this property automatically propagates its value to all the subcomponent battery objects inside this Pack object. However, this change does not propagate to the other battery objects in your MATLAB workspace.

Example: `batteryPack.CoolingPlateBlockPath = "batt_lib/Thermal/Edge Cooling"`

`Position` — Location of battery object
`[0 0 0]` (default) | vector of real and finite entries

Location of the battery object in a 3-D Cartesian coordinate system, specified as a vector of real and finite entries.

Example: `batteryPack.Position = [0 0 0]`

`Name` — Name of battery pack
`"Pack1"` (default) | string

Name of the battery pack, specified as a string.

Example: `batteryPack.Name = "ModuleAssembly2"`

`CircuitConnection` — Type of electrical connection
`"Series` (default) | `"Parallel"`

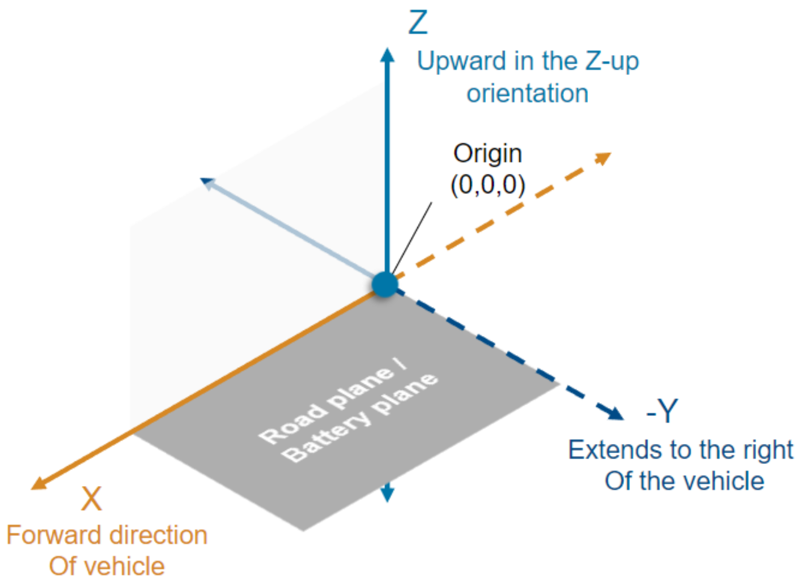
Type of electrical connection between module assemblies, specified as `"Series` or `"Parallel"`.

Example: `batteryPack.CircuitConnection = "Series"`

`StackingAxis` — Preferential stacking direction
`"X"` (default) | `"Y"`

Preferential stacking direction for the arrangement of battery cells in a 2-D Cartesian coordinate system, specified as `"X"` or `"Y"`.

This figure shows the global coordinate system for batteries.


Z
Upward in the Z-up orientation
Origin (0,0,0)
Road plane / Battery plane
-Y
Extends to the right Of the vehicle
X
Forward direction Of vehicle

To plot the `Pack` object in the direction of the *y*-axis, set this property to `"Y"` before creating the `BatteryChart` object.

Example: `batteryPack.StackingAxis = "Y"`

`MassFactor` — Additional non-cell-related mass
`1` (default) | positive double

Additional non-cell-related mass added to the pack by components such as busbars, tabs, and collector plates, specified as a strictly positive double greater than or equal to 1.

Example: `batteryPack.MassFactor = 1.2`

`NonCellResistance` — Option to use electrical resistance blocks
`'off'` (default) | on/off logical value

Option to use electrical resistance blocks to represent additional electrical resistances from non-cell components, specified as `'off'`, `'on'`, or as numeric or logical `1` (`true`) or `0` (`false`). A value of `'on'` is equivalent to `true`, and `'off'` is equivalent to `false`. Thus, you can use the value of this property as a logical value. The value is stored as an on/off logical value of type `matlab.lang.OnOffSwitchState`.

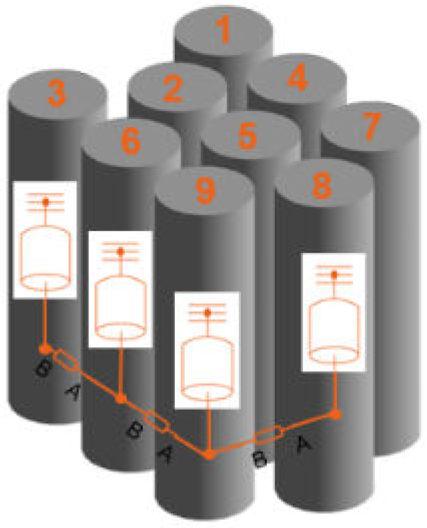
Example: `batteryPack.NonCellResistance = 'on'`

`InterCellThermalPath` — Option to use simple thermal resistance block for inter cell heat
`'off'` (default) | on/off logical value

Option to use thermal resistance blocks to represent a linear cell-to-cell heat transfer path between adjacent battery cells, specified as `'off'`, `'on'`, or as numeric or logical `1` (`true`) or `0` (`false`). A value of `'on'` is equivalent to `true`, and `'off'` is equivalent to `false`. Thus, you can use the value of this property as a logical value. The value is stored as an on/off logical value of type `matlab.lang.OnOffSwitchState`.

Enabling the inter-cell thermal path is useful only when you have selected a cell thermal model and more than one cell or cell model exists inside the battery.

You can define the value for the thermal resistance parameter after model creation.



If you set this property to `'on'`, you cannot set the **InterCellRadiativeThermalPath** property to `'on'`.

Note Setting this property automatically propagates its value to all the subcomponent battery objects inside this Pack object. However, this change does not propagate to the other battery objects in your MATLAB workspace.

Example: `batteryPack.InterCellThermalPath = "on"`

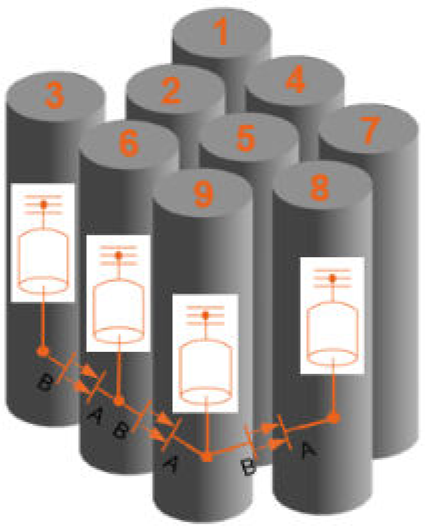
`InterCellRadiativeThermalPath` — Option to use simple thermal resistance block for inter cell heat

`'off'` (default) | on/off logical value

Option to use thermal resistance blocks to represent a radiative cell-to-cell heat transfer path between adjacent battery cells, specified as `'off'`, `'on'`, or as numeric or logical `1` (`true`) or `0` (`false`). A value of `'on'` is equivalent to `true`, and `'off'` is equivalent to `false`. Thus, you can use the value of this property as a logical value. The value is stored as an on/off logical value of type `matlab.lang.OnOffSwitchState`.

Enabling the inter-cell radiative thermal path is useful only when you have selected a cell thermal model and more than one cell or cell model exists inside the battery.

You can define the value for the radiation heat transfer parameters after model creation.



If you set this property to `'on'`, you cannot set the **InterCellThermalPath** property to `'on'`.

Note Setting this property automatically propagates its value to all the subcomponent battery objects inside this Pack object. However, this change does not propagate to the other battery objects in your MATLAB workspace.

Example: `batteryPack.InterCellRadiativeThermalPath = "on"`

`PackagingVolume` — Volume of battery
`simscape.Value([],"m^3")` (default) | `simscape.Value` object

This property is read-only.

Volume of the battery, returned as a `simscape.Value` object with a unit of volume.

`CumulativeMass` — Cumulative mass of battery
`simscape.Value` object

This property is read-only.

Cumulative mass of the battery, returned as a `simscape.Value` object with a unit of mass.

`CumulativeCellCapacity` — Cumulative cell capacity of battery
`simscape.Value` object

This property is read-only.

Cumulative cell capacity of the battery, returned as a `simscape.Value` object with unit of current multiplied by time.

The value of this property is equal to the individual cell capacity multiplied by the number of cells electrically connected in parallel inside this battery object.

`CumulativeCellEnergy` — Cumulative cell energy of battery
`simscape.Value` object

This property is read-only.

Cumulative cell energy of the battery, returned as a `simscape.Value` object with unit of energy.

The value of this property is equal to the individual cell energy multiplied by the total number of cells inside this battery object.

`NumModels` — Number of cell model blocks
double

This property is read-only.

Number of cell model blocks in simulation, returned as a double.

`CellNumbering` — Numbering for all cells
structure

This property is read-only.

Numbering for all of the cells inside of the battery, returned as a structure.

`ThermalNodes` — Vectorized thermal node information
structure

This property is read-only.

Vectorized thermal node information for external boundary conditions, returned as a structure. This information comprise XY location, XY dimensions, and number of thermal nodes. If you do not define a battery cell geometry, this property is an empty structure.

Type — Type of battery
`"Pack"`

This property is read-only.

Type of battery, returned as `"Pack"`.

Examples

Create Cylindrical Cell Pack with Two Series Module Assemblies

Create a `Cell` object with a cylindrical geometry.

```
batteryCell = Cell(Geometry=CylindricalGeometry)
```

Create a `ParallelAssembly` object of three parallel cells with the default topology.

```
pSet = ParallelAssembly(Cell=batteryCell,NumParallelCells=3)
```

Use this `ParallelAssembly` object to create a `Module` object of 10 parallel assemblies connected in series and stack them along the *x*-axis.

```
batteryModule = Module(ParallelAssembly=pSet,NumSeriesAssemblies=10,StackingAxis="X")
```

Use the `Module` object to create a `ModuleAssembly` object of four identical modules connected in series.

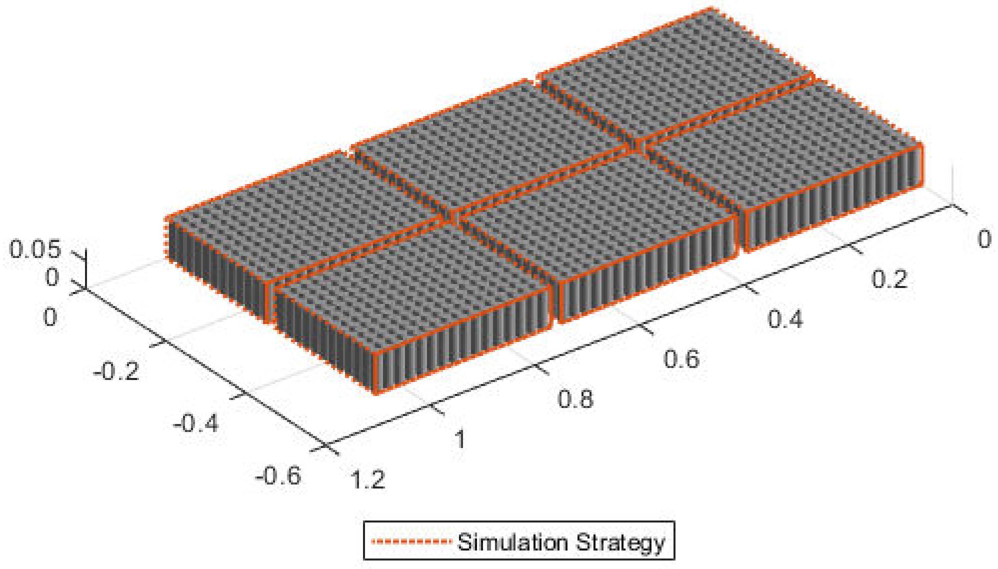
```
batteryModuleAssembly = ModuleAssembly(Module=repmat(batteryModule,1,4))
```

Use the `ModuleAssembly` object to create a `Pack` object of two identical module assemblies connected in series.

```
batteryPack = Pack(ModuleAssembly=repmat(batteryModuleAssembly,1,2))
```

Visualize the pack by using a `BatteryChart` object.

```
packChart = BatteryChart(Battery=batteryPack);
```



Define Cell Balancing Strategy for Pack

To define the cell balancing strategy for a pack, follow the steps in “Create Cylindrical Cell Pack with Two Series Module Assemblies” on page 2-82 to create a pack.

Set a common cell balancing strategy.

```
batteryPack.BalancingStrategy = "Passive"
```

To maintain consistency between the `BalancingStrategy` properties of the pack and all of its underlying components, setting the pack balancing strategy automatically modifies all the `BalancingStrategy` property in each of the underlying module components of the pack.

```
disp(batteryPack.ModuleAssembly(1).Module(1).BalancingStrategy);
disp(batteryPack.ModuleAssembly(2).Module(1).BalancingStrategy);
```

```
Passive
Passive
```

Version History

Introduced in R2022b

See Also

Apps
Battery Builder

Objects
`ModuleAssembly` | `BatteryChart`

Functions
`buildBattery`

Simscape Blocks
Pack (Generated Block)

Topics
“Battery Modeling Workflow”
“Build Simple Model of Battery Pack in MATLAB and Simscape”
“Build Detailed Model of Battery Pack From Pouch Cells”

ParallelAssembly

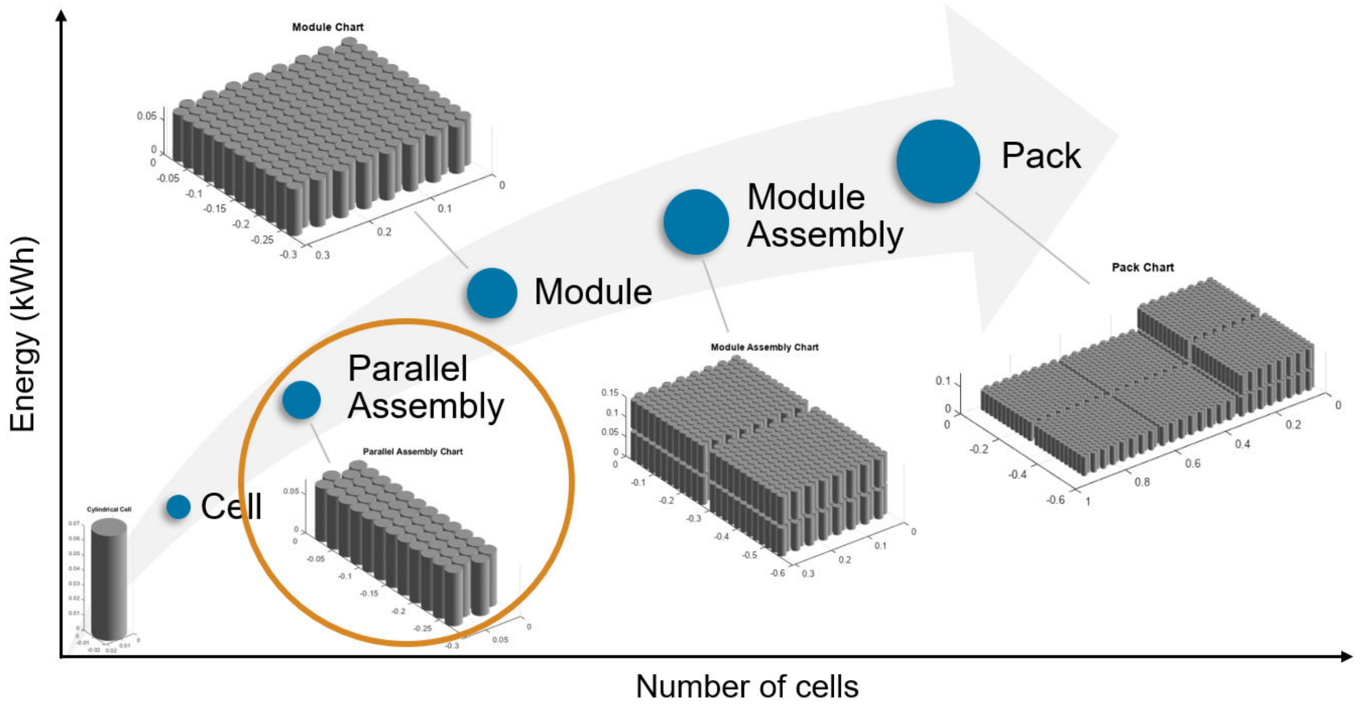
Create parallel assembly of battery cells

Description

Use `ParallelAssembly` to create a battery parallel assembly object that represents a number of cells connected electrically in parallel under a specific topological configuration or geometrical arrangement. You can use this object as an input to the `Module` object.

To specify the number of cells connected in parallel, use the `NumParallelCells` property. To generate a Simscape model of your `ParallelAssembly` object, use the `buildBattery` function. This object only supports the definition of structural or design parameters. You can modify the run-time parameters for this model block and its constituent cells after you create the model.

The `ParallelAssembly` object is the second stage of a battery pack system model in a bottom-up approach. Pack models are required for architecture evaluation in early development stages, software and hardware development, system integration and requirement evaluation, cooling system design, control strategy development hardware-in-the-loop, and many more applications.

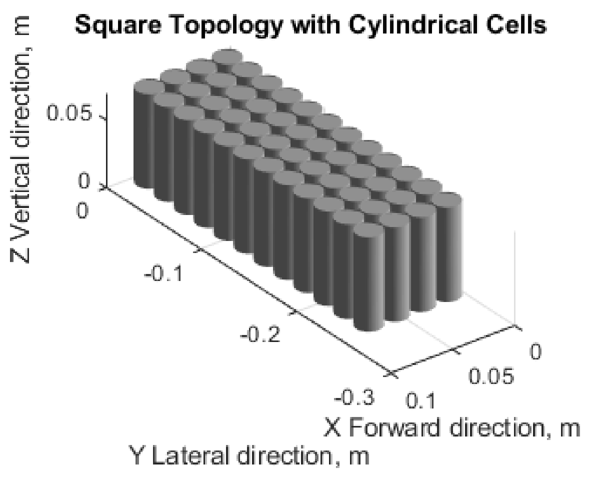


Topology of Parallel Assembly

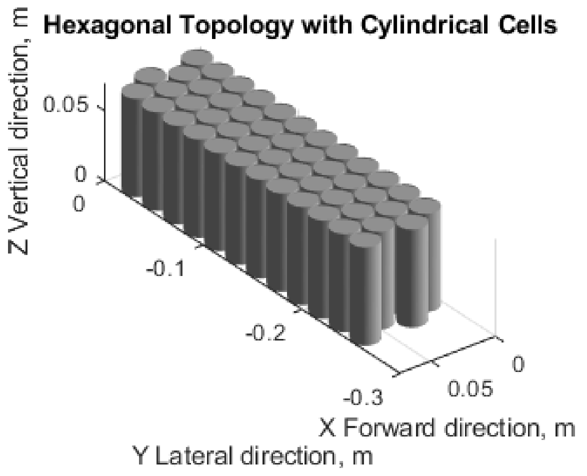
To meet battery system packaging and space requirements, you can arrange the battery cells in many different topologies or geometrical configurations. Use the `Topology` property of the `ParallelAssembly` object to define the geometrical arrangement of your battery in a 3-D Cartesian coordinate system.

The topology of the parallel assembly depends on the format of the battery cells that comprise the assembly. Depending on the value of the `Topology` property, you can modify the `Rows` property to best represent your specific design. The `Rows` property is defined with respect to the *x*-axis of the Cartesian coordinate system. By default, the cells are stacked along the *y*-axis.

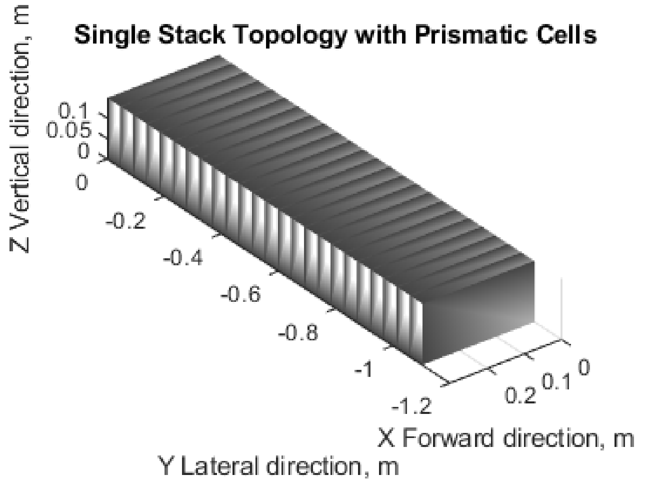
This figure shows the different topologies of a parallel assembly according to the format of the battery cells that comprise the assembly:



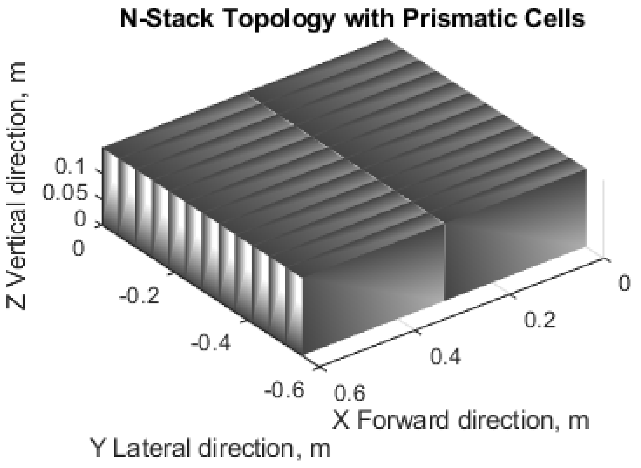
Square Topology with Cylindrical Cells
Z Vertical direction, m
0.05
0
0
-0.1
-0.2
-0.3
0.1
0.05
0
X Forward direction, m
Y Lateral direction, m



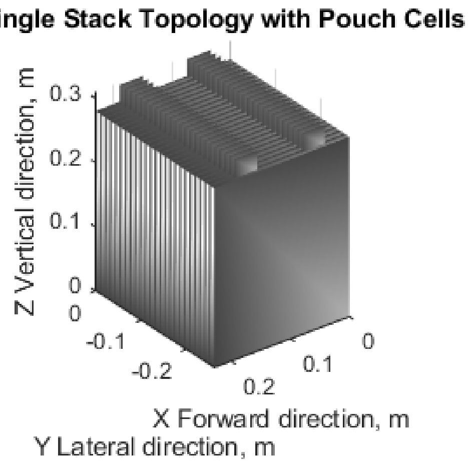
Hexagonal Topology with Cylindrical Cells
Z Vertical direction, m
0.05
0
0
-0.1
-0.2
-0.3
0.05
0
X Forward direction, m
Y Lateral direction, m



Single Stack Topology with Prismatic Cells
Z Vertical direction, m
0.1
0.05
0
0
-0.2
-0.4
-0.6
-0.8
-1
-1.2
0.2
0.1
0
X Forward direction, m
Y Lateral direction, m



N-Stack Topology with Prismatic Cells
Z Vertical direction, m
0.1
0.05
0
0
-0.2
-0.4
-0.6
0.6
0.4
0.2
0
X Forward direction, m
Y Lateral direction, m



Single Stack Topology with Pouch Cells
Z Vertical direction, m
0.3
0.2
0.1
0
0
-0.1
-0.2
0.2
0.1
0
X Forward direction, m
Y Lateral direction, m

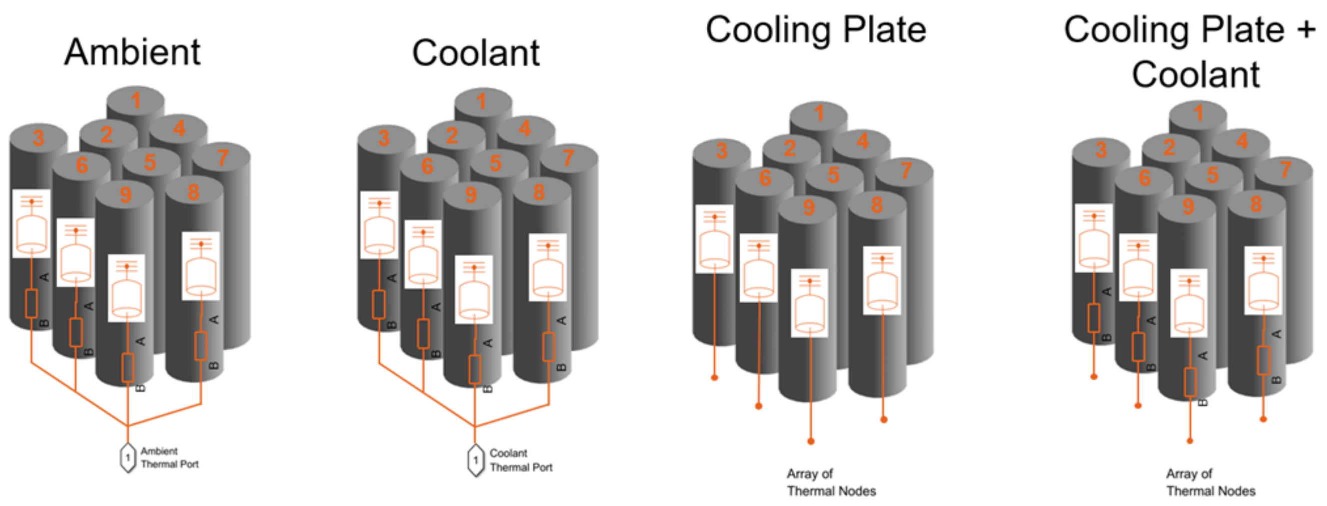
To modify the number of cells connected in parallel, use the `NumParallelCells` property. You can modify the topology of the parallel assembly by using the `Topology` property, which depends on the cell format. The `Rows` property is defined with respect to the *x*-axis of the Cartesian system. By default, cells are stacked along the *y*-axis.

Thermal Boundary Conditions

Thermal boundary conditions define the specific heat transfer mechanisms that occur at each interface of a cell thermal model and its surroundings. In battery systems, cells are typically thermally coupled to different heat sources and sinks, all of which have an effect on the battery cell temperature. The number and type of thermal boundary conditions for a cell model depends on the thermal and mechanical design of the battery system.

For example, you can place cells on an aluminium cooling plate to enhance heat removal and, at the same time, join them together mechanically with a potting compound that effectively eliminates or decreases the inter-cell heat exchange path. The cell temperature has a direct impact on battery performance and lifetime. Therefore, it is crucial to predict this state in dynamic simulation.

Inside a battery object, you can set up a thermal network of lumped-thermal-mass cell models to simultaneously capture the thermal paths to the ambient, the coolant, and/or the cooling plate:

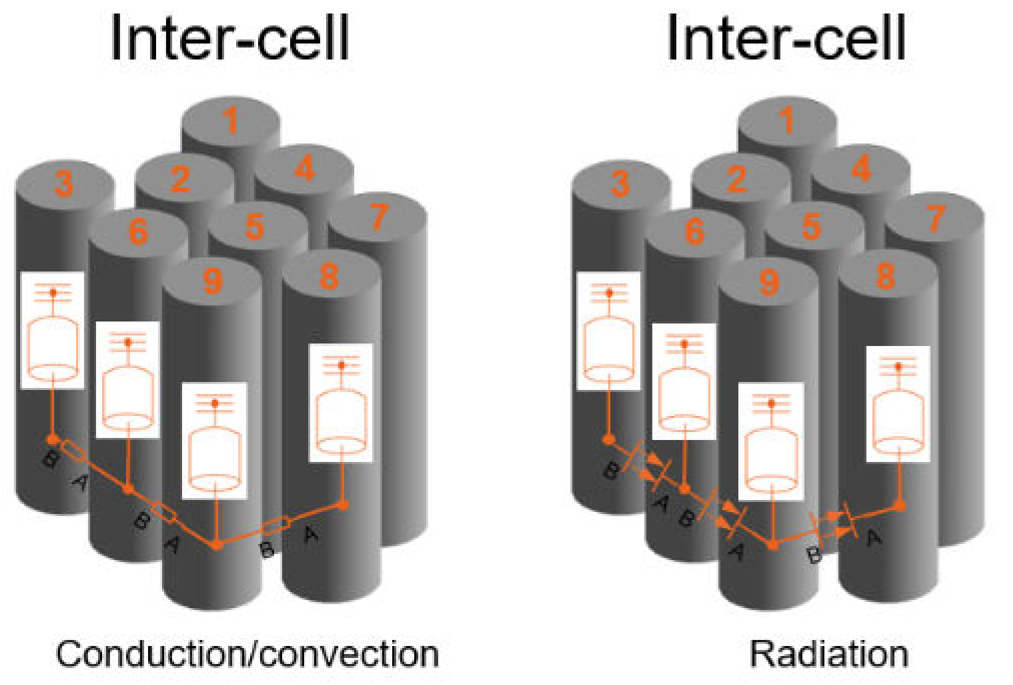


These options are not mutually exclusive. For example, your battery model can combine both the coolant thermal path and the cooling thermal plates to model individual thermal resistances between the individual cells and the sections of the cooling plate.

For more information about thermal paths, see the “AmbientThermalPath” on page 2-0 , “CoolantThermalPath” on page 2-0 , and “CoolingPlate” on page 2-0 properties.

You can also model direct cell-to-cell heat exchange. This is important when you want to simulate more detailed thermal management strategies or even thermal propagation scenarios where inter-cell heat transfer happens at faster rates than ambient or coolant rates. In the battery industry, you can link battery cells to each other through many different means. For example, you can link cylindrical cells by using potting compounds for mechanical rigidity, stability, and thermal isolation, or other types of thermal interface materials. You can also use dielectric fluids or other compounds to heat or cool down cylindrical cells, as well as forced air convection.

You can define the thermal parameters for the inter-cell heat exchange after you create the battery model. You can find these parameters from first principles calculations and more detailed 3D simulations.



These options are not mutually exclusive.

For more information about inter-cell thermal paths, see the “InterCellThermalPath” on page 2-0 and “InterCellRadiativeThermalPath” on page 2-0 properties.

Creation

Syntax

`import simscape.battery.builder.*; batteryParallelAssembly = ParallelAssembly`
`import simscape.battery.builder.*; batteryParallelAssembly = ParallelAssembly(Name=Value)`

Description

`import simscape.battery.builder.*; batteryParallelAssembly = ParallelAssembly` creates a parallel assembly that comprises battery cells with default property values.

`import simscape.battery.builder.*; batteryParallelAssembly = ParallelAssembly(Name=Value)` sets “Properties” on page 2-90 using one or more name-value arguments. For example, create a parallel assembly with 48 cylindrical cells stacked in a square topology over four rows with an intercell gap equal to 0.001 m.

```
parallelAssembly = ParallelAssembly(...
    NumParallelCells=48, ...
    Cell=Cell(Geometry=CylindricalGeometry), ...
    Topology="Square", ...
    Rows=4, ...
    InterCellGap=simscape.Value(0.001,"m"));
```

Properties

`Cell` — Cell component in parallel assembly

`Cell` object (default)

Cell component in the parallel assembly, specified as a `Cell` object. The `ParallelAssembly` object creates this component and then electrically connects it in parallel a number of times equal to the value of the `NumParallelCells` property.

Example: `batteryParallelAssembly.Cell = Cell(Geometry=CylindricalGeometry)`

`NumParallelCells` — Number of cells connected in parallel

1 (default) | positive integer

Number of cells connected in parallel inside the parallel assembly, specified as a strictly positive and finite integer. The value of this property must be less than `150`.

Example: `batteryParallelAssembly.NumParallelCells = 48`

`Rows` — Number of rows of parallel assembly

1 (default) | positive integer

Number of rows of the parallel assembly parallel to the stacking axis, specified as a strictly positive integer. The value of this property must be less than `50` and less than the value of the `NumParallelCells` property.

Example: `batteryParallelAssembly.Rows = 4`

`Topology` — Geometrical arrangement of cells

`"None"` (default) | `"Square"` | `"Hexagonal"` | `"SingleStack"` | `"NStack"` | `"SingleStack"`

Geometrical arrangement of the cells relative to the cell format, specified as either `"Square"` or `"Hexagonal"` for cylindrical cells, `"SingleStack"` or `"NStack"` for prismatic cells, or `"SingleStack"` for pouch cells.

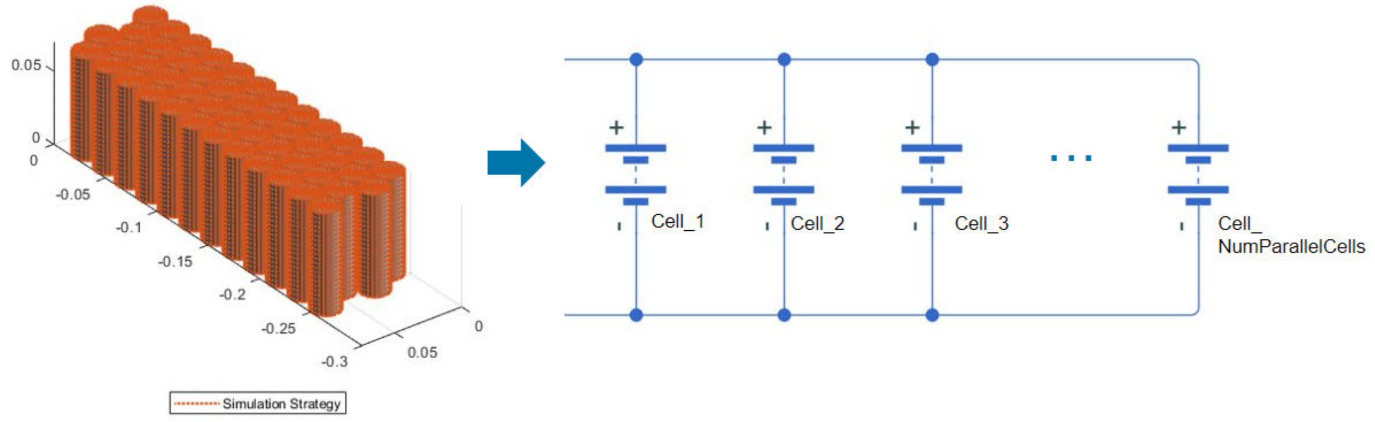
Example: `batteryParallelAssembly.Topology = "Hexagonal"`

`ModelResolution` — Model resolution in simulation

`"Lumped"` (default) | `"Detailed"`

Model resolution or fidelity in the simulation, specified as:

- `"Lumped"` — Choose this value for the lowest fidelity. The assembly uses only one electrical model. To obtain the fastest model compilation and running time, set the model resolution to this value.
- `"Detailed"` — Choose this value for the highest fidelity. The assembly uses one electrical model and one thermal model for each battery cell in the parallel assembly.

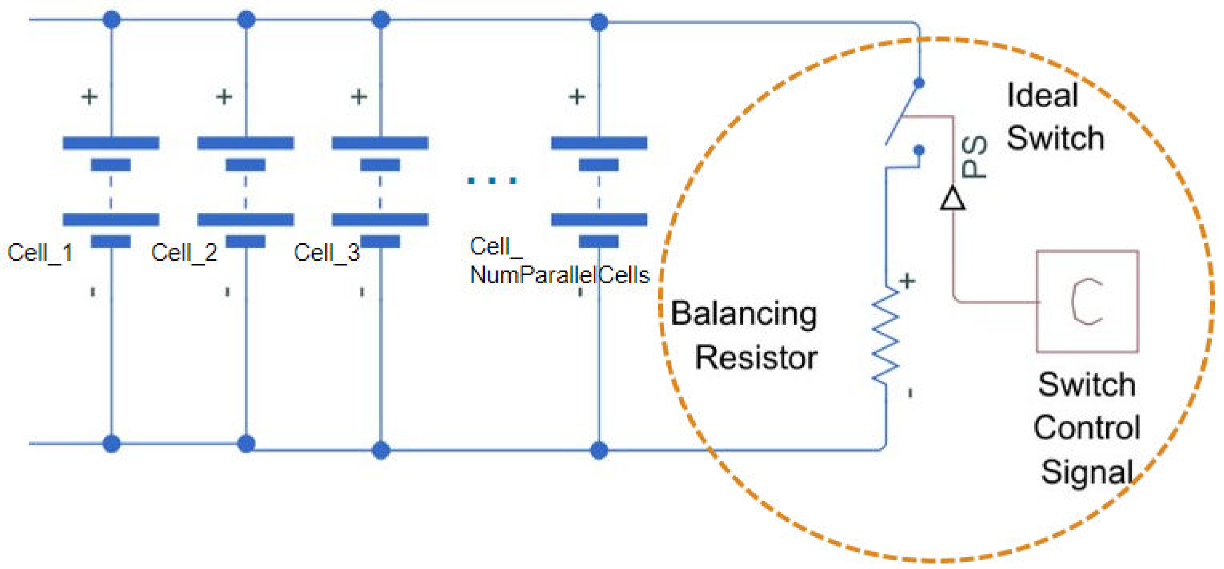


A number of cell model blocks equal to the value of the `NumParallelCells` property represents each cell component.

Example: `batteryParallelAssembly.ModelResolution = "Detailed"`

`BalancingStrategy` — State-of-charge balancing strategy
`"None"` (default) | `"Passive"`

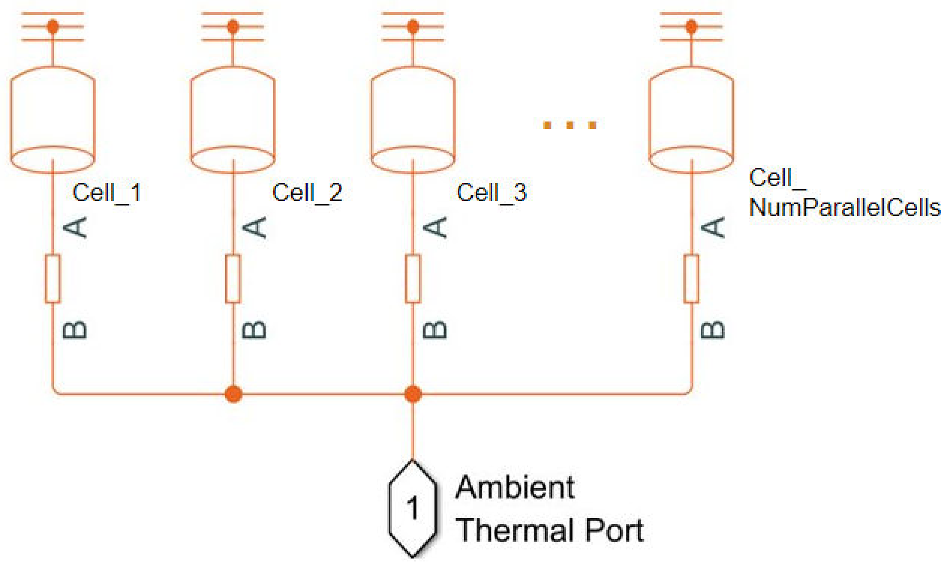
State-of-charge balancing strategy for the parallel assembly, specified as `"None"` or `"Passive"`. Set this property to `"Passive"` to add an ideal balancing circuit connected in parallel to the `ParallelAssembly` Simscape model and a physical port used for switch control. The switch is open when the control signal is equal to 0. The switch is closed when the control signal is equal to 1.



Example: `batteryParallelAssembly.BalancingStrategy = "Passive"`

`AmbientThermalPath` — Option to use simple thermal resistance block
`"None"` (default) | `"CellBasedThermalResistance"`

Option to use a simple thermal resistance block connected between the cells and a Simscape thermal network, specified as `"CellBasedThermalResistance"` or `"None"`.



Note Setting this property automatically propagates its value to all the subcomponent battery objects inside this ParallelAssembly object. However, this change does not propagate to the other battery objects in your MATLAB workspace.

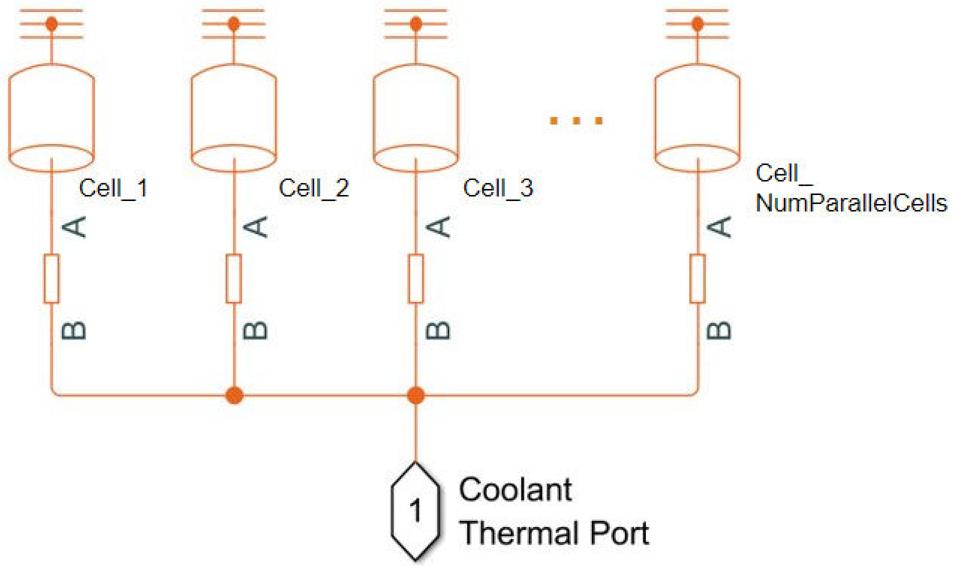
Example: `batteryParallelAssembly.AmbientThermalPath = "CellBasedThermalResistance"`

`CoolantThermalPath` — Option to use simple thermal resistance block

`"None"` (default) | `"CellBasedThermalResistance"`

Option to use a simple thermal resistance block connected between the cells and a Simscape thermal network, specified as `"CellBasedThermalResistance"` or `"None"`.

If you also define a cooling plate surface, the object connects the thermal resistance block between each battery thermal model and its corresponding element in the array of thermal nodes. You can use this thermal resistance to capture conductive and convective heat transfer mechanisms related to the cell and cooling system design.



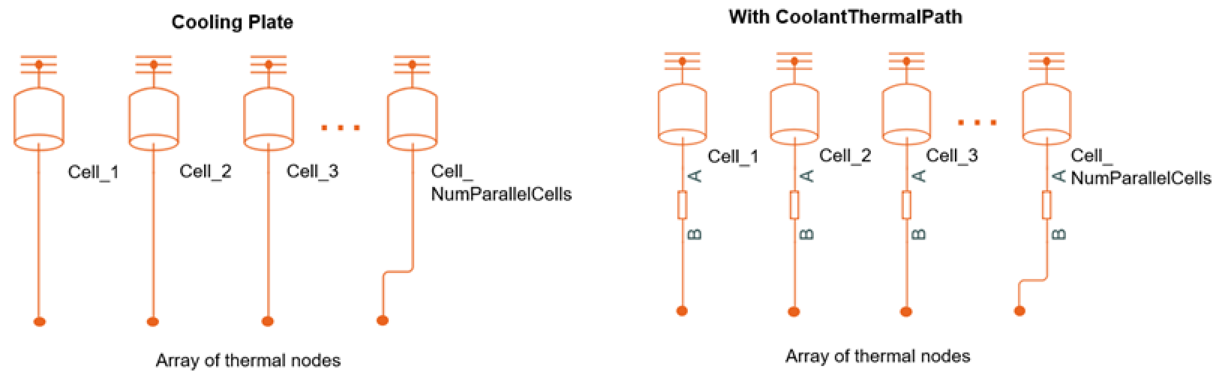
Note Setting this property automatically propagates its value to all the subcomponent battery objects inside this ParallelAssembly object. However, this change does not propagate to the other battery objects in your MATLAB workspace.

Example: `batteryParallelAssembly.CoolantThermalPath = "CellBasedThermalResistance"`

`CoolingPlate` — Option to use cooling plate component
`"None"` (default) | `"Top"` | `"Bottom"`

Option to use a cooling plate component at a specific surface boundary, specified as `"Top"`, `"Bottom"`, or `"None"`.

This option adds a vectorized array of thermal nodes connector to the `ParallelAssembly` object. The object connects each element of the array of thermal nodes to each thermal model. If you also select a coolant thermal path, the object connects a Thermal Resistance block between each battery thermal model and its corresponding element in the array of thermal nodes. If you define a cell geometry, the `ThermalNodes` property contains the number of thermal nodes, their dimensions, and their locations in a 2-D Cartesian plane.



Note Setting this property automatically propagates its value to all the subcomponent battery objects inside this ParallelAssembly object. However, this change does not propagate to the other battery objects in your MATLAB workspace.

Example: `batteryParallelAssembly.CoolingPlate = "Top"`

`CoolingPlateBlockPath` — Option to specify which cooling plate block to assign
`"None"` (default) | `"batt_lib/Thermal/Edge Cooling"` | `"batt_lib/Thermal/Parallel Channels"` | `"batt_lib/Thermal/U-Shaped Channels"`

Option to specify which cooling plate block you want to assign to the ParallelAssembly object at the boundary defined by the `CoolingPlate` property, specified as `"batt_lib/Thermal/Edge Cooling"`, `"batt_lib/Thermal/Parallel Channels"`, `""batt_lib/Thermal/U-Shaped Channels""`, or `"None"`.

Note Setting this property automatically propagates its value to all the subcomponent battery objects inside this ParallelAssembly object. However, this change does not propagate to the other battery objects in your MATLAB workspace.

Example: `batteryParallelAssembly.CoolingPlateBlockPath = "batt_lib/Thermal/Edge Cooling"`

`InterCellGap` — Shortest distance between cells
`simscape.Value(0.001,"m")` (default) | `simscape.Value(positive scalar,"Length unit")` | positive scalar

Shortest distance between the cells inside the parallel assembly, specified as a positive scalar or a `simscape.Value` object that represents a positive scalar with a unit of length. The value of this property must be less than 0.1 m.

If you set this property directly with a positive scalar value instead of using a `simscape.Value` object, the object converts the value to a `simscape.Value` object with meter as its physical unit.

Example: `batteryParallelAssembly.InterCellGap = simscape.Value(0.01,"m")`

Example: `batteryParallelAssembly.InterCellGap = 0.01`

`Position` — Location of battery object
`[0 0 0]` (default) | vector of real and finite entries

Location of the battery object in a 3-D Cartesian coordinate system, specified as a vector of real and finite entries.

Example: `batteryParallelAssembly.Position = [0 0 0]`

Name — Name of parallel assembly
`"ParallelAssembly1"` (default) | string

Name of the parallel assembly, specified as a string.

Example: `batteryParallelAssembly.Name = "ParallelAssembly2"`

`MassFactor` — Additional non-cell-related mass
`1` (default) | positive double

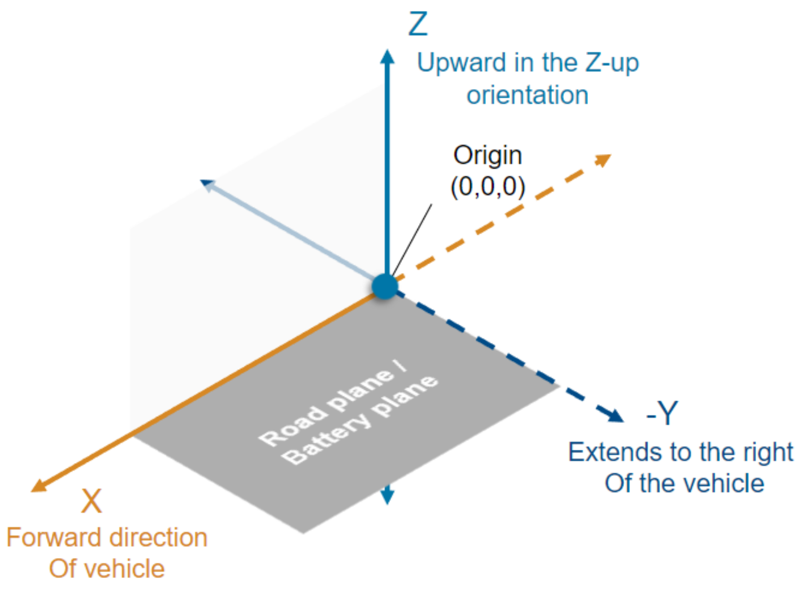
Additional non-cell-related mass added to the parallel assembly by components such as busbars, tabs, and collector plates, specified as a strictly positive double greater than or equal to 1.

Example: `batteryParallelAssembly.MassFactor = 1.2`

`StackingAxis` — Preferential stacking direction
`"Y"` (default) | `"X"`

Preferential stacking direction for the arrangement of battery cells in a 2-D Cartesian coordinate system, specified as either `"X"` or `"Y"`.

This figure shows the global coordinate system for batteries.


Z
Upward in the Z-up orientation
Origin (0,0,0)
Road plane / Battery plane
-Y
Extends to the right Of the vehicle
X
Forward direction Of vehicle

To plot the parallel assembly object in the direction of the *x*-axis, set this property to `"X"` before creating the `BatteryChart` object.

Example: `batteryParallelAssembly.StackingAxis = "Y"`

`NonCellResistance` — Option to use electrical resistance blocks
`'off'` (default) | on/off logical value

Option to use electrical resistance blocks to represent additional electrical resistances from non-cell components, specified as `'off'`, `'on'`, or as numeric or logical `1` (`true`) or `0` (`false`). A value of `'on'` is equivalent to `true`, and `'off'` is equivalent to `false`. Thus, you can use the value of this property as a logical value. The value is stored as an on/off logical value of type `matlab.lang.OnOffSwitchState`.

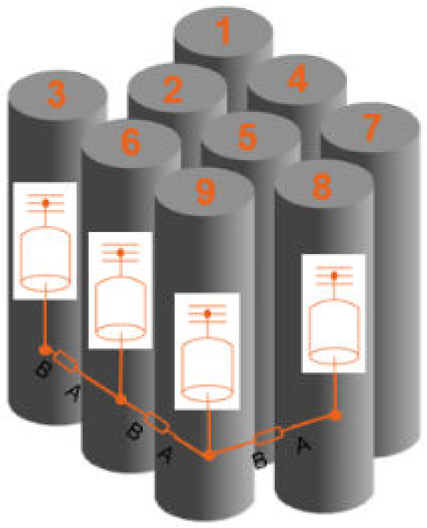
Example: `batteryParallelAssembly.NonCellResistance = 'on'`

`InterCellThermalPath` — Option to use simple thermal resistance block for inter cell heat
`'off'` (default) | on/off logical value

Option to use thermal resistance blocks to represent a linear cell-to-cell heat transfer path between adjacent battery cells, specified as `'off'`, `'on'`, or as numeric or logical `1` (`true`) or `0` (`false`). A value of `'on'` is equivalent to `true`, and `'off'` is equivalent to `false`. Thus, you can use the value of this property as a logical value. The value is stored as an on/off logical value of type `matlab.lang.OnOffSwitchState`.

Enabling the inter-cell thermal path is useful only when you have selected a cell thermal model and more than one cell or cell model exists inside the battery.

You can define the value for the thermal resistance parameter after model creation.



If you set this property to `'on'`, you cannot set the **InterCellRadiativeThermalPath** property to `'on'`.

Note Setting this property automatically propagates its value to all the subcomponent battery objects inside this ParallelAssembly object. However, this change does not propagate to the other battery objects in your MATLAB workspace.

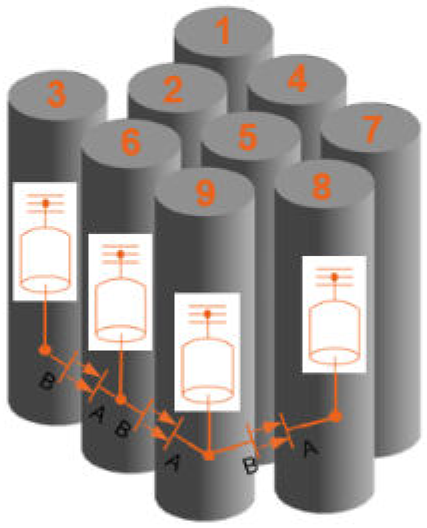
Example: `batteryParallelAssembly.InterCellThermalPath = "on"`

`InterCellRadiativeThermalPath` — Option to use simple thermal resistance block for inter cell heat
`'off'` (default) | on/off logical value

Option to use thermal resistance blocks to represent a radiative cell-to-cell heat transfer path between adjacent battery cells, specified as `'off'`, `'on'`, or as numeric or logical `1` (`true`) or `0` (`false`). A value of `'on'` is equivalent to `true`, and `'off'` is equivalent to `false`. Thus, you can use the value of this property as a logical value. The value is stored as an on/off logical value of type `matlab.lang.OnOffSwitchState`.

Enabling the inter-cell radiative thermal path is useful only when you have selected a cell thermal model and more than one cell or cell model exists inside the battery.

You can define the value for the radiation heat transfer parameters after model creation.



If you set this property to `'on'`, you cannot set the **InterCellThermalPath** property to `'on'`.

Note Setting this property automatically propagates its value to all the subcomponent battery objects inside this ParallelAssembly object. However, this change does not propagate to the other battery objects in your MATLAB workspace.

Example: `batteryParallelAssembly.InterCellRadiativeThermalPath = "on"`

`PackagingVolume` — Volume of battery
`simscape.Value([],"m^3")` (default) | `simscape.Value` object

This property is read-only.

Volume of the battery, returned as a `simscape.Value` object with a unit of volume.

`CumulativeMass` — Cumulative mass of battery
`simscape.Value` object

This property is read-only.

Cumulative mass of the battery, returned as a `simscape.Value` object with a unit of mass.

`CumulativeCellCapacity` — Cumulative cell capacity of battery
`simscape.Value` object

This property is read-only.

Cumulative cell capacity of the battery, returned as a `simscape.Value` object with unit of current multiplied by time.

The value of this property is equal to the individual cell capacity multiplied by the number of cells electrically connected in parallel inside this battery object.

`CumulativeCellEnergy` — Cumulative cell energy of battery
`simscape.Value` object

This property is read-only.

Cumulative cell energy of the battery, returned as a `simscape.Value` object with unit of energy.

The value of this property is equal to the individual cell energy multiplied by the total number of cells inside this battery object.

`NumModels` — Number of cell model blocks
double

This property is read-only.

Number of cell model blocks in the simulation, returned as a double.

`NumInterCellThermalConnections` — Number of internal thermal connections between cells
integer

This property is read-only.

Number of the internal thermal connections between the cells, returned as an integer.

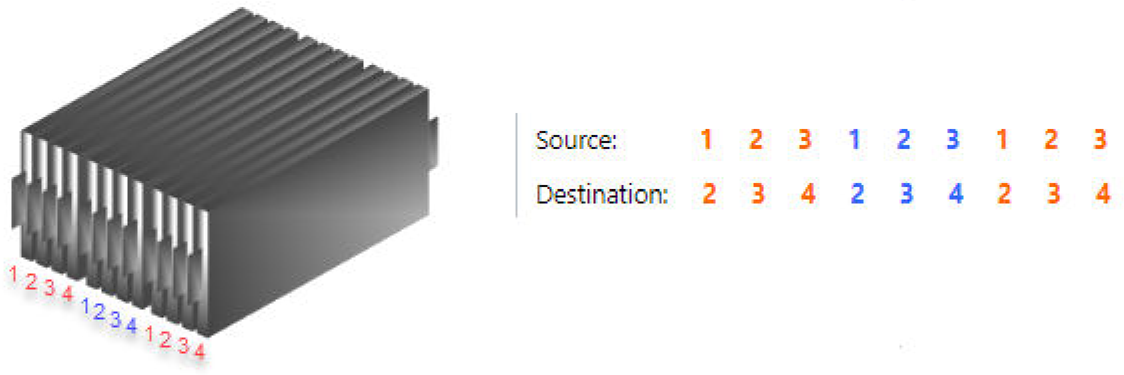
`InterCellConnectionsMapping` — 2-D map of inter-cell connections
matrix of integer

This property is read-only.

2-D map of the internal connections between the cells, returned as a matrix of integer.

The cell numbering convention refers to the `Layout` property. To understand the number assigned to each cell and how they relate to the their neighbors,see the `Layout` property.

For example, for a battery module with three parallel assemblies of four pouch cells each, this property is equal to:



CellNumbering — Numbering for all cells
structure

This property is read-only.

Numbering for all of the cells inside of the battery, returned as a structure.

ThermalNodes — Vectorized thermal node information
structure

This property is read-only.

Vectorized thermal node information for external boundary conditions, returned as a structure. This information comprises the XY location, XY dimensions, and number of thermal nodes. If you do not define a battery cell geometry, this property is an empty structure.

Layout — 2-D distribution of sub-components in parent
matrix of integer

This property is read-only.

2-D distribution of sub-components or cells within the parent, returned as a matrix of integer.

This property depends on the `StackingAxis` property and on the number of cells.

Type — Type of battery
`"ParallelAssembly"`

This property is read-only.

Type of battery object, returned as `"ParallelAssembly"`.

Examples

Create Parallel Assembly Object from Three Pouch Cells

Create a `Cell` object with a pouch geometry.

```
batteryCell = Cell(Geometry=PouchGeometry)
```

Create a `ParallelAssembly` object of three cells with the default topology.

```
pSet = ParallelAssembly(Cell=batteryCell,NumParallelCells=3)
```

Create Parallel Assembly of 46 Cylindrical Cells with Custom Topology

Create a `Cell` object with a cylindrical geometry.

```
batteryCell = Cell(Geometry=CylindricalGeometry)
```

Create a `ParallelAssembly` object of 46 cells stacked in 14 rows .

```
pSet = ParallelAssembly(Cell=batteryCell,NumParallelCells=46,Rows=14)
```

Set the topology of the parallel assembly to `"Square"`.

```
pSet.Topology = "Square";
```

Visualize the parallel assembly by using a `BatteryChart` object.

```
pSetChart = BatteryChart(Battery=pSet)
```

Version History

Introduced in R2022b

See Also

Apps
Battery Builder

Objects
`Cell` | `Module` | `BatteryChart`

Functions
`buildBattery`

Simscape Blocks
ParallelAssembly (Generated Block)

Topics
“Battery Modeling Workflow”
“Build Simple Model of Battery Pack in MATLAB and Simscape”
“Build Detailed Model of Battery Pack From Pouch Cells”

PouchGeometry

Pouch geometry for battery cell

Description

Use `PouchGeometry` to represent the geometry of a pouch battery cell. To specify the dimensions of a pouch geometry, set the object properties.

Creation

Syntax

```
import simscape.battery.builder.*; pouch = PouchGeometry
import simscape.battery.builder.*; pouch = PouchGeometry(Name=Value)
```

Description

`import simscape.battery.builder.*; pouch = PouchGeometry` creates a pouch geometry with default property values.

`import simscape.battery.builder.*; pouch = PouchGeometry(Name=Value)` sets properties using one or more name-value arguments.

Properties

`Length` — Length of pouch
`simscape.Value(0.23,"m")` (default) | `simscape.Value(positive scalar,"Length unit")` | positive scalar

Length of the pouch, specified as a positive scalar or a `simscape.Value` object that represents a positive scalar with a unit of length. This value must be less than `5` meters.

If you set this property directly with a positive scalar value instead of using a `simscape.Value` object, the object converts the value to a `simscape.Value` object with meter as its physical unit.

Example: `pouch.Length = simscape.Value(0.23,"m")`

Example: `pouch.Length = 0.23`

`Thickness` — Thickness of pouch
`simscape.Value(0.01,"m")` (default) | `simscape.Value(positive scalar,"Length unit")` | positive scalar

Thickness of the pouch, specified as a positive scalar or a `simscape.Value` object that represents a positive scalar with a unit of length. This value must be less than `0.5` meters.

If you set this property directly with a positive scalar value instead of using a `simscape.Value` object, the object converts the value to a `simscape.Value` object with meter as its physical unit.

Example: `pouch.Thickness = simscape.Value(0.045,"m")`

Example: `pouch.Thickness = 0.01`

`Height` — Height of pouch
`simscape.Value(0.28,"m")` (default) | `simscape.Value(positive scalar,"Length unit")` | positive scalar

Height of the pouch, specified as a positive scalar or a `simscape.Value` object that represents a positive scalar with a unit of length. The height of the pouch is aligned with the *z*-axis of the reference frame. This value must be strictly positive.

If you set this property directly with a positive scalar value instead of using a `simscape.Value` object, the object converts the value to a `simscape.Value` object with meter as its physical unit.

Example: `pouch.Height = simscape.Value(0.28,"m")`

Example: `pouch.Height = 0.28`

`TabLocation` — Location of pouch tabs
`"Standard"` (default) | `"Opposed"`

Location of the pouch tabs, specified as `"Standard"` or `"Opposed"`.

Example: `pouch.tabLocation = "Standard"`

`TabWidth` — Width of pouch tabs
`simscape.Value(0.04,"m")` (default) | `simscape.Value(positive scalar,"Length unit")` | positive scalar

Width of the pouch tabs, specified as a positive scalar or a `simscape.Value` object that represents a positive scalar with a unit of length. This value must be less than the value of the `Length` property.

If you set this property directly with a positive scalar value instead of using a `simscape.Value` object, the object converts the value to a `simscape.Value` object with meter as its physical unit.

Example: `pouch.TabWidth = simscape.Value(0.04, "m")`

Example: `pouch.TabWidth = 0.04`

`TabHeight` — Height of pouch tabs
`simscape.Value(0.03,"m")` (default) | `simscape.Value(positive scalar,"Length unit")` | positive scalar

Height of the pouch tabs, specified as a positive scalar or a `simscape.Value` object that represents a positive scalar with a unit of length. This value must be less than the value of the `Height` property.

If you set this property directly with a positive scalar value instead of using a `simscape.Value` object, the object converts the value to a `simscape.Value` object with meter as its physical unit.

Example: `pouch.TabHeight = simscape.Value(0.03, "m")`

Example: `pouch.TabHeight = 0.03`

Version History

Introduced in R2022b

See Also

Apps
Battery Builder

Objects
`CylindricalGeometry` | `PrismaticGeometry` | `Cell` | `ParallelAssembly` | `Module` | `ModuleAssembly` | `Pack`

Topics
"Build Simple Model of Battery Pack in MATLAB and Simscape"
"Build Detailed Model of Battery Pack From Pouch Cells"

PrismaticGeometry

Prismatic geometry for battery cell

Description

Use `PrismaticGeometry` to represent the geometry of a prismatic battery cell. To specify the dimensions of a prismatic geometry, use the `Length`, `Thickness`, and `Height` properties.

Creation

Syntax

```
import simscape.battery.builder.*; prism = PrismaticGeometry
import simscape.battery.builder.*; prism = PrismGeometry(Name=Value)
```

Description

`import simscape.battery.builder.*; prism = PrismaticGeometry` creates a prismatic geometry with default property values.

`import simscape.battery.builder.*; prism = PrismGeometry(Name=Value)` creates a prism geometry with a specified length, height, and thickness using name-value arguments.

Properties

`Length` — Length of prism
`simscape.Value(0.3,"m")` (default) | `simscape.Value(positive scalar,"Length unit")` | positive scalar

Length of the prism, specified as a positive scalar or a `simscape.Value` object that represents a positive scalar with a unit of length. This value must be less than 5 meters.

If you set this property directly with a positive scalar value instead of using a `simscape.Value` object, the object converts the value to a `simscape.Value` object with meter as its physical unit.

Example: `prism.Length = simscape.Value(0.3,"m")`

Example: `prism.Length = 0.3`

`Thickness` — Thickness of prism
`simscape.Value(0.045,"m")` (default) | `simscape.Value(positive scalar,"Length unit")` | positive scalar

Thickness of the prism, specified as a positive scalar or a `simscape.Value` object that represents a positive scalar with a unit of length. This value must be less than 0.5 meters.

If you set this property directly with a positive scalar value instead of using a `simscape.Value` object, the object converts the value to a `simscape.Value` object with meter as its physical unit.

Example: `prism.Thickness = simscape.Value(0.045,"m")`

Example: `prism.Thickness = 0.045`

`Height` — Height of prism
`simscape.Value(0.15,"m")` (default) | `simscape.Value(positive scalar,"Length unit")` | positive scalar

Height of the prism, specified as a positive scalar or a `simscape.Value` object that represents a positive scalar with a unit of length. The height of the prism is aligned with the *z*-axis of the reference frame. This value must be strictly positive.

If you set this property directly with a positive scalar value instead of using a `simscape.Value` object, the object converts the value to a `simscape.Value` object with meter as its physical unit.

Example: `prism.Height = simscape.Value(0.15,"m")`

Example: `prism.Height = 0.15`

Version History
Introduced in R2022b

See Also

Apps
Battery Builder

Objects
`CylindricalGeometry` | `PouchGeometry` | `Cell` | `ParallelAssembly` | `Module` | `ModuleAssembly` | `Pack`

Topics
“Build Simple Model of Battery Pack in MATLAB and Simscape”
“Build Detailed Model of Battery Pack From Pouch Cells”

Tools and Apps

Battery Builder

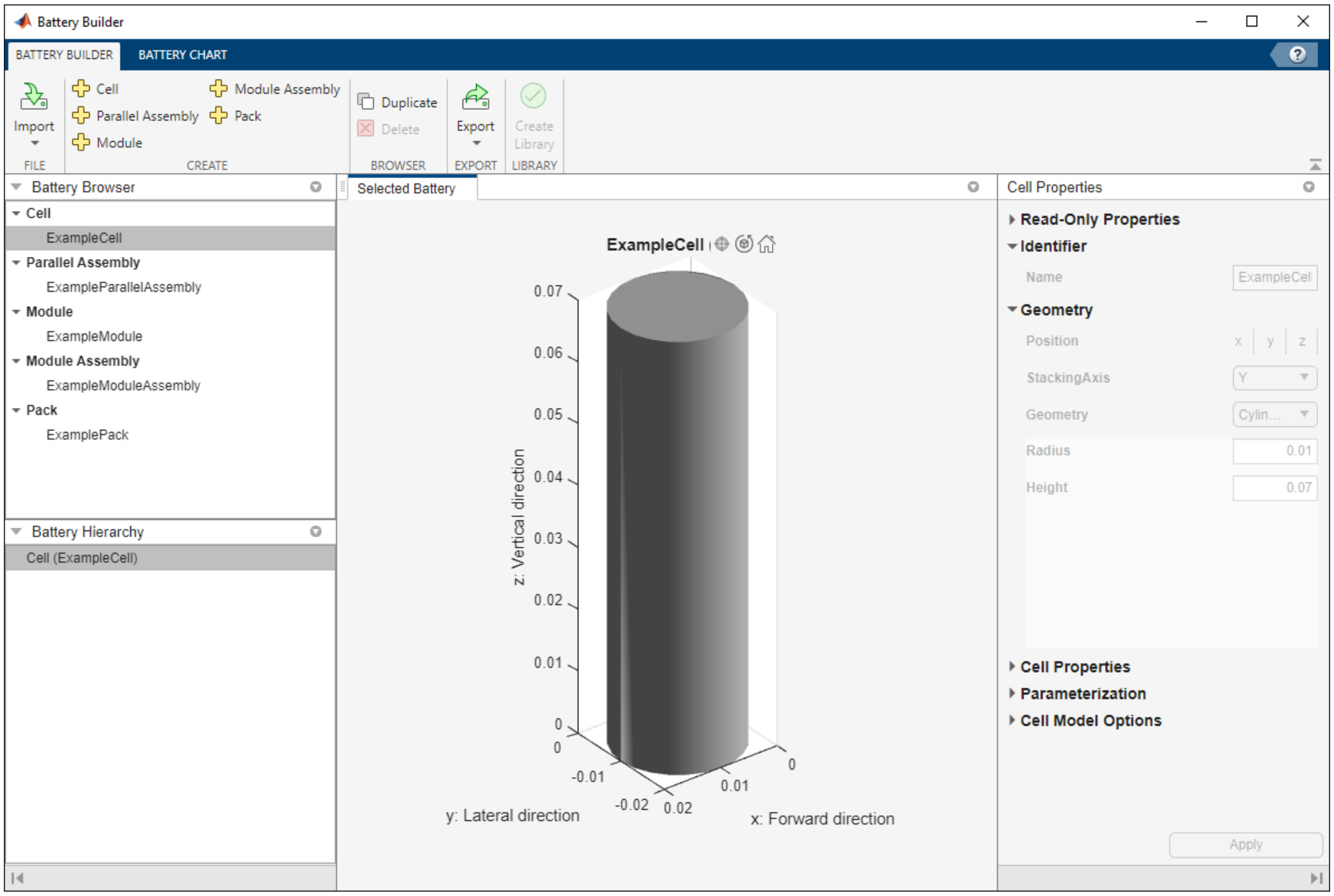
Define battery system and automatically generate dynamic models

Description

Use the **Battery Builder** app to create Simscape battery models by interactively creating, modifying, and visualizing MATLAB battery objects. You can use these MATLAB objects to define your own battery design specifications, visualize your battery in a 3-D plot, customize the modeling resolution during simulation, and generate a Simulink library that contains your custom battery blocks. You can use these blocks to assist with virtual battery design and verification, develop battery control algorithms using Simulink, explore design sensitivities, and design thermal management strategies.

With the **Battery Builder** app, you can:

- Import existing battery objects from your workspace or MAT file.
- Create new `Cell`, `ParallelAssembly`, `Module`, `ModuleAssembly`, and `Pack` objects.
- Generate a 3-D plot of the battery object, edit the plotting options, and export the plot to a FIG file.
- Inspect the hierarchy of a battery object and visualize all of its subcomponents.
- Edit the properties of a battery object, such as geometrical data and thermal boundary conditions.
- Export objects you create in the app to your workspace or a MAT file.
- Create a Simscape battery model from an object.



Open the Battery Builder App

- MATLAB® Toolstrip: On the **Apps** tab, under **Simscape**, click the **Battery Builder** icon.
- MATLAB command prompt: Enter `batteryBuilder`.

Examples

Import Battery Objects from Workspace or MAT File

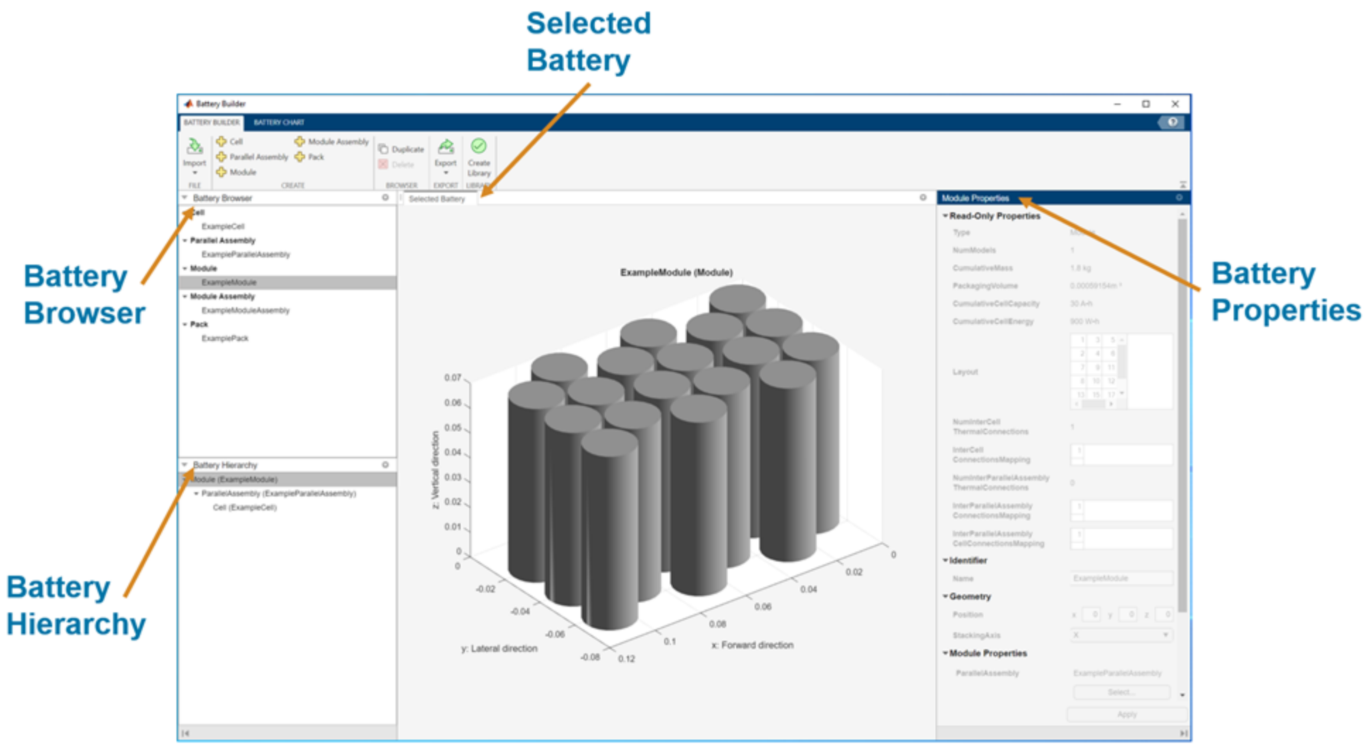
To import the battery objects from your workspace or from a MAT file, under the **Battery Builder** tab, in the **Import** section of the toolstrip, click **Import**. Then click **Import from Workspace** or **Import from MAT-file**.

Note When you import an object, the app also automatically imports all its subcomponent battery objects.

Create New Battery Object

To create a new battery object, under the **Battery Builder** tab, in the **Create** section of the toolstrip, click **Cell**, **Parallel Assembly**, **Module**, **Module Assembly**, or **Pack** to create the battery object. This action creates the corresponding battery object with default property values.

The **Battery Browser** panel on the left of the app now contains the new object. You can select this object, visualize it in the **Selected Battery** tab, check its hierarchy and child objects in the **Battery Hierarchy** panel, and edit its properties in the **Properties** panel on the right of the app.



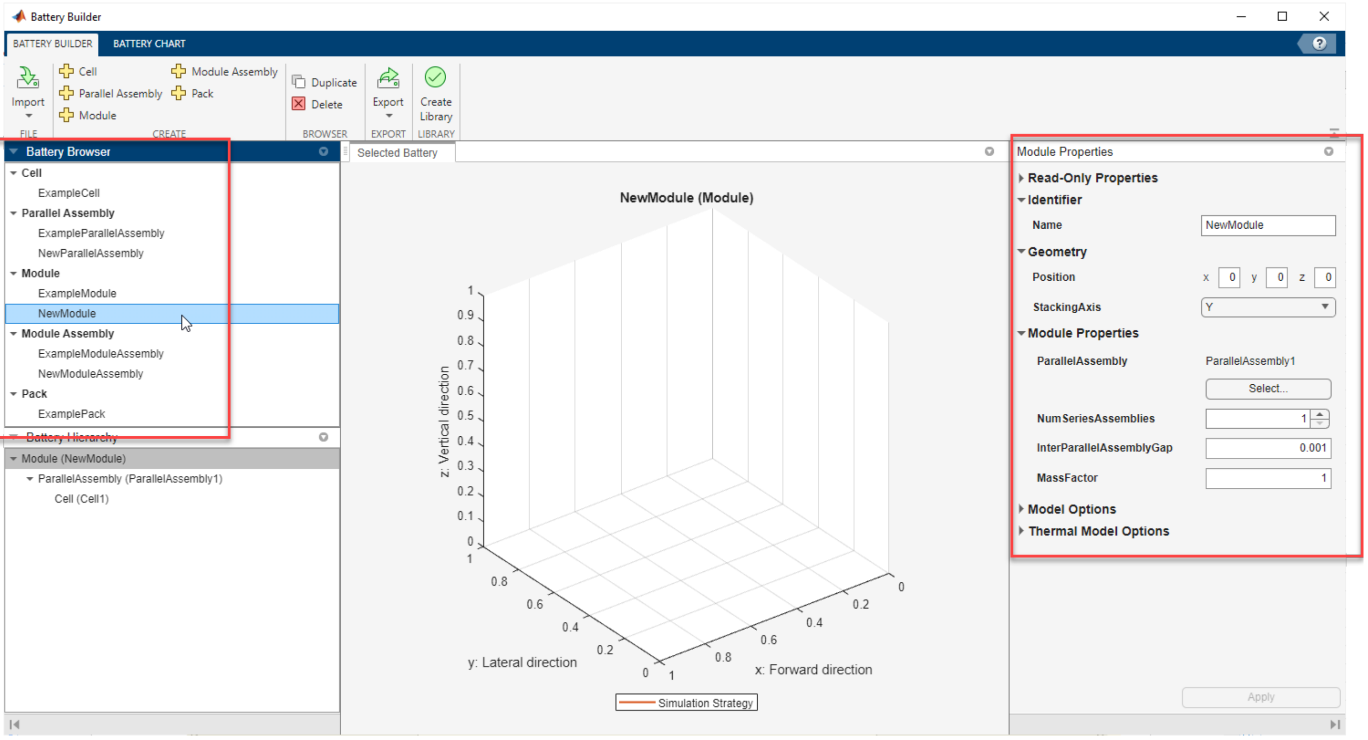
Visualize Battery Object and Modify Plotting Options

To visualize a battery object, under the **Battery Builder** tab, in the **Battery Browser** panel, select the object you want to visualize. The **Selected Battery** tab now displays a 3-D plot of the object.

You can edit multiple properties of the plot under the **Battery Chart** tab, such as the axes labels, axes direction, title of the plot, and lights. You can also check the current simulation strategy and model resolution of the selected battery object. To visualize the simulation strategy in the plot, in the **Simulation Strategy** section of the toolstrip, check the **Visible** box.

Edit Battery Object

To edit a battery object, select it in the left **Battery Browser** panel of the app. The **Properties** panel on the right of the app now displays all the editable properties of the object.



Name: NewModule
Position: x 0 y 0 z 0
StackingAxis: Y
ParallelAssembly: ParallelAssembly1
NumSeriesAssemblies: 1
InterParallelAssemblyGap: 0.001
MassFactor: 1

To apply any changes, you must refresh the object by clicking **Apply**.

Each battery object has its own properties and parameters. For information about the properties of the battery objects, see the `Cell`, `ParallelAssembly`, `Module`, `ModuleAssembly`, and `Pack` documentation pages.

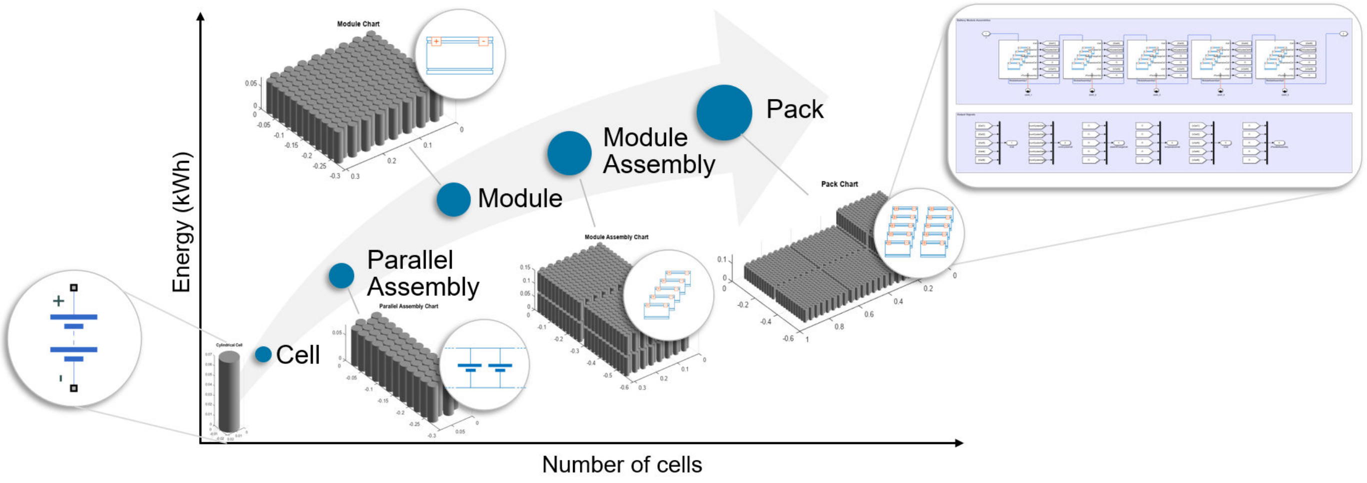
Note To display the relevant unit for relevant properties, including those of type `Simscape.Value`, point on the edit field of the property.

The **Properties** panel of the app contains the properties of the `ParallelAssembly`, `Module`, `ModuleAssembly`, and `Pack` objects in six sections.

- Read-Only Properties — Data about the total mass, volume, energy, and capacity of the battery. You cannot modify these properties.
- Identifier — Name of the object. Modify this property to modify the name of the object as it appears in the left **Battery Browser** panel of the app.
- Geometry — Geometrical information about the object, such as the position inside the chart and the stacking axis of the battery.
- Properties — Structural information about the object, including the subcomponent object from which the parent object is created, how these children objects are stacked, and the gap between the objects.
- Model Options — Modeling information about the object, such as options to enable non-cell resistances or a balancing strategy.
- Thermal Model Options — Information that defines the thermal boundary conditions of the object.

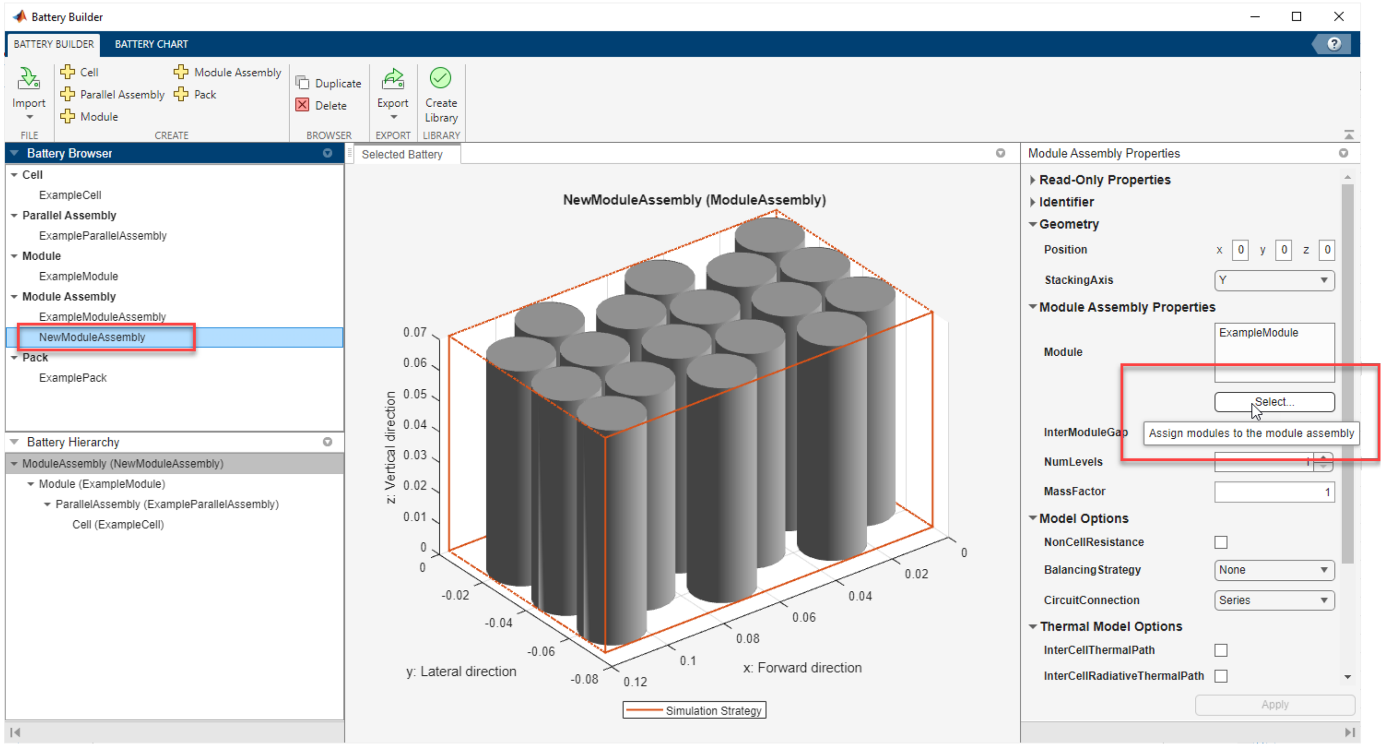
Link Child Components to Parent Components

When you create a `ParallelAssembly`, `Module`, `ModuleAssembly`, or `Pack` object, you must link it to its defining subcomponent in parallel and/or in series and scale it up to generate larger battery system models.



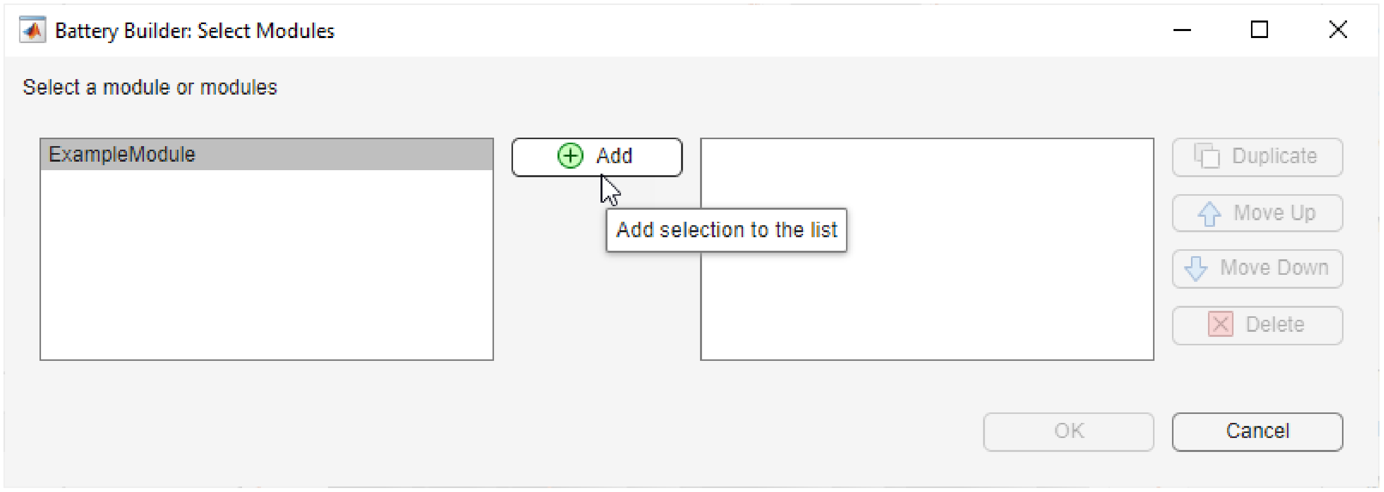
For example, a battery module comprises multiple parallel assemblies in series. When you create a `Module` object, you must link it to the associated `ParallelAssembly` subcomponent object that forms the parent module.

To link a child component to its parent component, first select the parent component to which you want to link a child component in the **Battery Browser** panel. Then, in the **Properties** panel on the right of the app, in the **Properties** section, search for the property with the name of the child component you want to link and click **Select**.



Position x 0 y 0 z 0
StackingAxis Y
Module ExampleModule
InterModuleGap
NumLevels
MassFactor 1
NonCellResistance ☐
BalancingStrategy None
CircuitConnection Series
InterCellThermalPath ☐
InterCellRadiativeThermalPath ☐

In the new window, select all the child components and click **Add**.

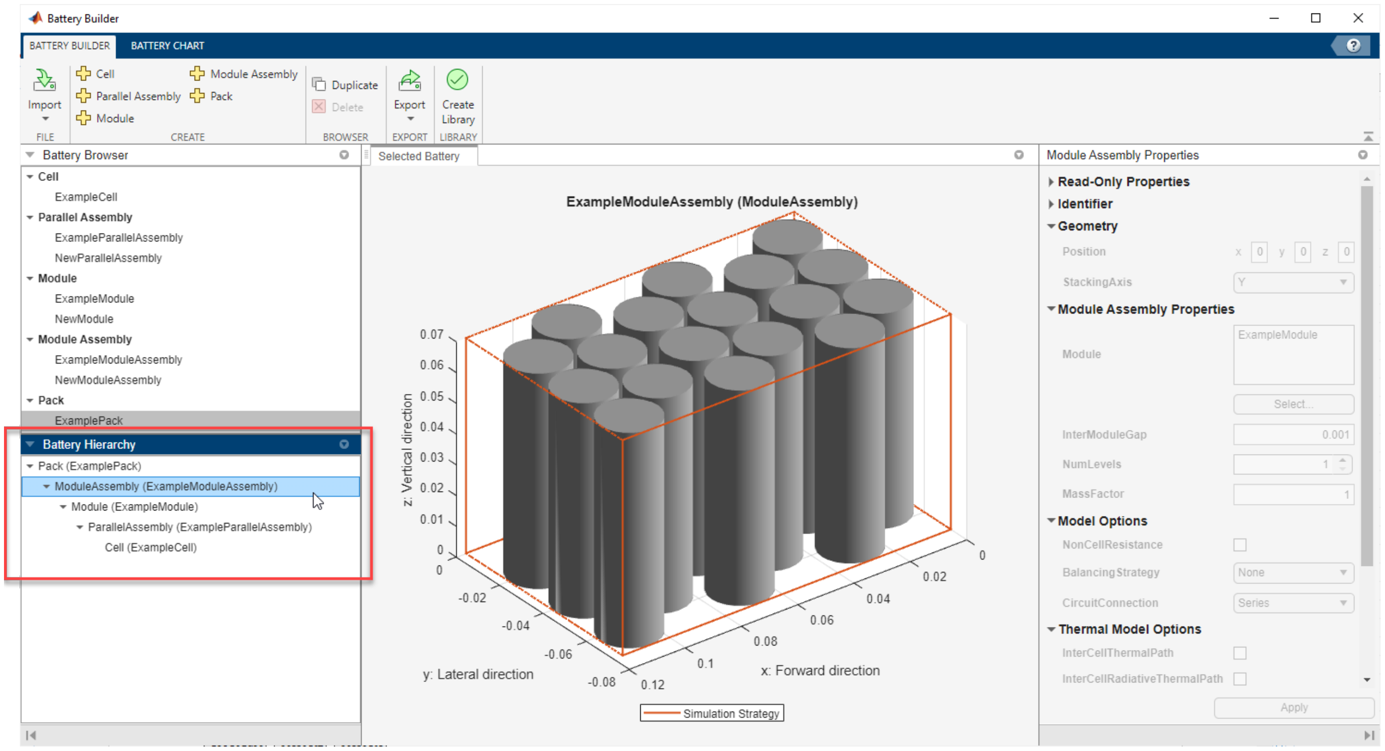

Battery Builder: Select Modules
Select a module or modules
ExampleModule
Add
Add selection to the list
Duplicate
Move Up
Move Down
Delete
OK
Cancel

You can duplicate, delete, or move the components you add by clicking the respective buttons on the right of the window.

Once you add the child component, click **OK**. This action links the child component to the parent component. You can visualize the hierarchy of your battery object in the **Battery Hierarchy** panel on the bottom left of the app.

Visualize Subcomponents of Battery Object in Battery Hierarchy Panel

You can visualize all the subcomponents of a specific battery object in the **Battery Hierarchy** panel of the app. Select a subcomponent inside the **Battery Hierarchy** panel to show a 3-D plot of the object in the **Selected Battery** panel and its properties in the **Properties** panel on the right of the app.



You cannot modify any of the properties of the subcomponent objects that you select in the **Battery Hierarchy** panel. To modify the objects, select them in the **Battery Browser** panel first.

Duplicate, Delete, or Recover Battery Object

The **Battery Builder** app allows you to manage your work by duplicating, deleting, or recovering your battery objects.

- To duplicate a battery object, first select it in the **Battery Browser** panel on the left of the app. Then, under the **Battery Builder** tab, in the **Browser** section of the toolstrip, click **Duplicate**. This action creates an identical copy of the selected object inside your battery browser. This duplicate object is not linked to any of the parent objects of the original object.

  Alternatively, in the **Battery Hierarchy** panel on the left of the app, right-click the object that you want to duplicate, and click **Copy to browser**.
- To delete a battery object, first select it in the **Battery Browser** panel on the left of the app. Then, under the **Battery Builder** tab, in the **Browser** section of the toolstrip, click **Delete**. This action deletes the object from the battery browser but does not unlink that object from any parent object that is using it.

- You can recover an object that you previously deleted if this object is a subcomponent of an existing parent object in your battery browser. To recover a deleted battery object that is still linked inside another parent object, first select the parent object that still contains this deleted subcomponent in the **Battery Browser** panel on the left of the app. Then, in the **Battery Hierarchy** panel, right-click the object that you want to recover, and click **Copy to browser**.

Export Battery Objects or Battery Charts

To export the battery objects to your workspace or to a MAT file, under the **Battery Builder** tab, in the **Export** section of the toolstrip, click **Export**. Then, click **Export from Workspace** or **Export from MAT-file**.

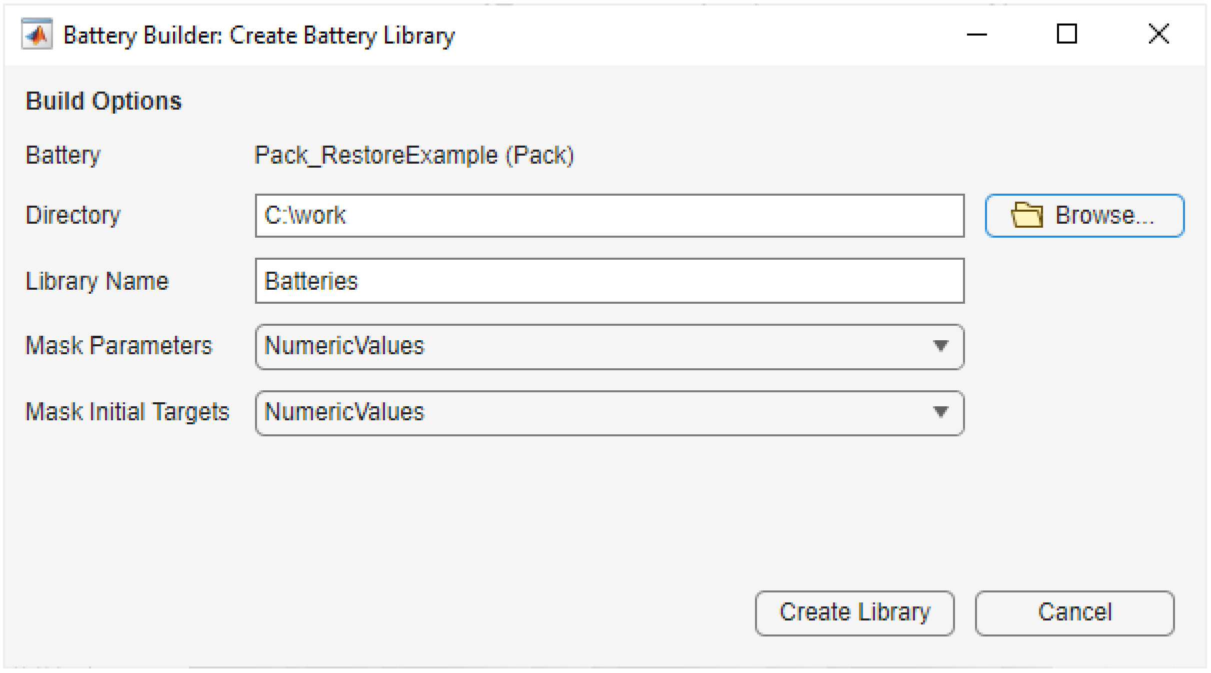
Note The battery objects that you edit in the app are not automatically saved in your workspace. If you close the app, you lose the session and the objects you create.

To export the battery chart of a battery object to a FIG file, first select the battery object in the **Battery Browser** panel. Then, under the **Battery Chart** tab, in the **Export** section of the toolstrip, click **Export Chart**.

Create Library Model from Battery Object

To create a library model from the `ParallelAssembly`, `Module`, `ModuleAssembly`, or `Pack` objects, under the **Battery Builder** tab, in the **Library** section of the toolstrip, click **Create Library**.

In the new window, specify the folder in which you want to save the library, the library name, and whether to use numeric values or variable names for the mask parameters and mask initial targets. For more information about each of these fields, see the `buildBattery` documentation page.



Build Options

Battery	Pack_RestoreExample (Pack)
Directory	C:\work
Library Name	Batteries
Mask Parameters	NumericValues
Mask Initial Targets	NumericValues

Click the **Create Library** button to generate the library model of your battery object in the specified folder. Open this model to access your battery objects as Simscape blocks that you can use as a starting point for architecture evaluation in early development stages, software and hardware development, system integration and requirement evaluation, cooling system design, control strategy development, hardware-in-the-loop, and many more applications.

- “Get Started with Battery Builder App”

Version History

Introduced in R2023a

See Also

Simscape Blocks

ParallelAssembly (Generated Block) | Module (Generated Block) | ModuleAssembly (Generated Block) | Pack (Generated Block)

Topics

“Get Started with Battery Builder App”
“Battery Modeling Workflow”
“Manage Battery Run-Time Parameters with Centralized Script”
“Connect Cooling Plate to Battery Blocks”