Simscape™ Release Notes

MATLAB®&SIMULINK®

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™ Release Notes

Contents

R2022b

R2022a

R2021b

R2016a

R2015aSP1

R2022b

Simscape Language

Array of Nodes: Define conserving ports as resizable arrays of elements

You can now use arrays and `for`-loop constructs when declaring component nodes. Follow these rules:

- Arrays, which previously applied only to the `components` member class, now also apply to the `nodes` member class.
- Array members must all belong to the same domain.
- The array size can be a parameter with the `ExternalAccess` attribute set to `modify`, which means that the block users can modify this value.
- Use `for`-loops to declare arrays of nodes and to connect members of the array to each other. You can also index into an array of nodes to specify textual connections between elements.
- Use nested `for`-loops to create multidimensional arrays of nodes.

An array of nodes (AON) is represented as a single port on the block icon. If you have two or more AON ports of the same domain type and identical size, you can establish element-wise connections between these arrays by drawing connection lines between the ports. When using AON ports:

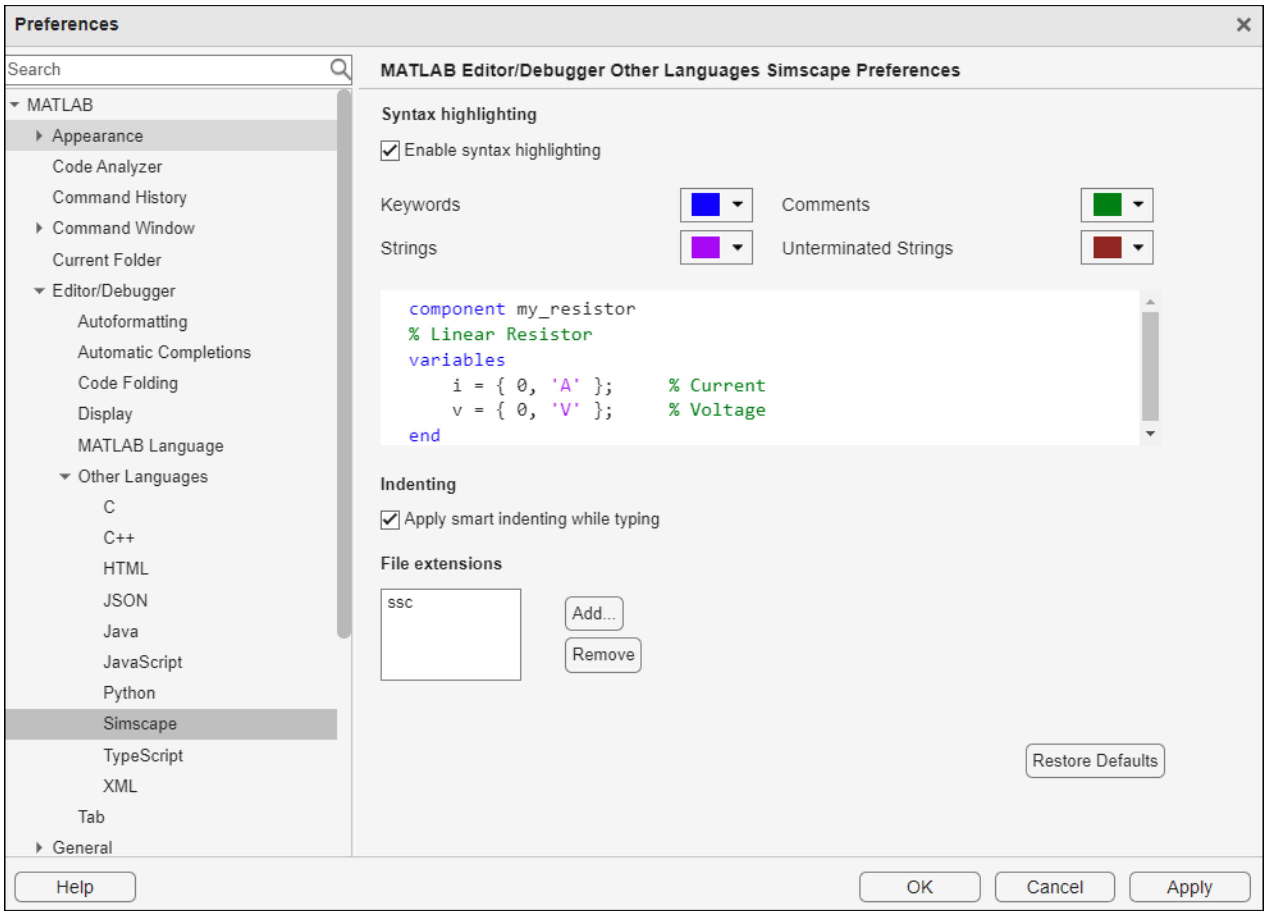
- Like with scalar ports, error checking for the domain type is performed at edit time. You cannot connect ports of different domain types.
- Error checking for the array size is performed at compile time. This means that, although you can graphically connect AON ports of different sizes, this connection generates an error when you compile the model or update the block diagram.
- Implicit scalar expansion is not supported. Connecting a scalar port to an AON port also generates a compile-time error.

For more information, see "Arrays of Nodes".

Simscape Language Preferences Panel in MATLAB Online: Change default colors and set other preferences for the Simscape language editor

In MATLAB® Online™, you can change the colors and other settings used by the Simscape language editor.

To access these settings, on the **Home** tab, in the **Environment** section, click **Preferences**. Select **MATLAB > Editor/Debugger > Other Languages > Simscape** to open the Simscape language preferences panel.



MATLAB Editor/Debugger Other Languages Simscape Preferences

Syntax highlighting

✓ Enable syntax highlighting

Keywords 🟦 Comments 🟩

Strings 🟪 Unterminated Strings 🟥

```
component my_resistor
% Linear Resistor
variables
    i = { 0, 'A' };     % Current
    v = { 0, 'V' };     % Voltage
end
```

Indenting

✓ Apply smart indenting while typing

File extensions

ssc

If you change the color for **Keywords**, **Comments**, **Strings**, or **Unterminated Strings**, the code sample inside the panel reflects your changes. You can also change other settings, such as enable or disable syntax highlighting or smart indenting. When you click **Apply**, the Editor window reflects these changes.

The **Restore Defaults** button restores the default Simscape language editor settings.

Foundation Library

Limited PS Integrator: Specify saturation bounds for PS Integrator block

The PS Integrator block can now function as a limited integrator, holding its output within the specified saturation bounds:

- When the integral is less than or equal to the lower saturation limit, the block holds the output at the lower saturation limit.
- When the integral is between the lower saturation limit and the upper saturation limit, the output is the integral.
- When the integral is greater than or equal to the upper saturation limit, the block holds the output at the upper saturation limit.

Use the **Limit output** parameter to specify whether to limit the output at the lower bound, upper bound, or both. The **Limit output source** parameter provides two ways of specifying the upper and lower bounds:

- `Internal` — Define the saturation bounds using the **Upper limit** and **Lower limit** parameters.
- `External` — Define the saturation bounds using the external physical signals at ports **U** and **L**.

PS Transfer Function Block Enhancement: Reset block output at events

The PS Transfer Function block is now resettable at events. You can specify that the block output resets to the initial output when the physical signal at port **R** triggers a reset. The initial output source can now be either a physical signal or a block parameter. If you set the **Initial output source** parameter to `External`, the physical signal at port **Y0** supplies the initial output value, which is also the output value after reset.

In the default configuration, with **External reset** parameter set to `None` and **Initial output source** set to `Internal`, the block functions as in previous releases.

Enhancements to Translational and Rotational Motion Sensors: Measure acceleration and parametrically control the visibility of output ports

The Ideal Translational Motion Sensor and Ideal Rotational Motion Sensor blocks have these enhancements:

- An additional physical signal output port that lets you measure acceleration.
- Additional parameters that control the visibility of physical signal output ports. Use these parameters to expose only the ports that are actually used for measurement in each particular instance. This enhancement helps improve the block diagram readability. Also, because exposing the acceleration measurement port involves additional computations, this port is hidden by default. Do not expose it unless you need to measure acceleration.

- In previous releases, the blocks measured only the relative velocity and position of port **R** with respect to port **C**. Now, the new **Measurement reference** parameter lets you disable port **C** and measure with respect to ground (internal reference node).

Conditional Port Visibility for Two-Phase Fluid Blocks: Expose additional ports in block variants

The source code for the Two-Phase Fluid library blocks has been updated to take advantage of the conditional port visibility and make this library consistent with other fluid block libraries. As a result, the block library has been streamlined by combining similar blocks that differed primarily by the number of ports:

- The Constant Volume Chamber (2P) block can now have between one and four ports. The four-port option is new. If a chamber has four ports, you can use it as a junction in a cross connection. Separate two-port and three-port blocks are no longer available. This change has no compatibility impact. All two-port and three-port chamber blocks in existing models are automatically replaced by a Constant Volume Chamber (2P) block with the appropriate number of conserving ports.
- The Local Restriction (2P) block can now have an optional input physical signal port that models the variable restriction of flow area. Variable Local Restriction (2P) block is no longer available. This change has no compatibility impact. All Variable Local Restriction (2P) blocks in existing models are automatically replaced by Local Restriction (2P) blocks with an exposed control port.

Pipe (MA) Block Enhancement: Model effect of condensation on wall surface

The Pipe (MA) block has a new parameter, **Condensation on wall surface**, that lets you model the effect of wall condensation on a cold pipe surface in contact with a moist air volume. Use this parameter when modeling HVAC systems that contain pipes and ducts. If these pipes and ducts are not well insulated, their surface could get cold, and condensation on wall surface occurs. When you set **Condensation on wall surface** to `On`, the convective heat transfer equation accounts for both sensible and latent heat, and the block also calculates the rate of water vapor condensation on the surface.

Thermal Library Enhancements: Expand heat transfer modeling capabilities and improve diagram layout

Several enhancements in the Thermal library provide additional options for modeling thermal effects within a Simscape network. These enhancements include:

- Thermal conductivity in the Conductive Heat Transfer block can be either constant, which you specify by the **Thermal conductivity** parameter, or variable. You can specify variable thermal conductivity either as a physical signal at port **K** or as a lookup table based on temperature.

  Additionally, you can set **Wall geometry** to `Planar` or `Cylindrical`. `Planar` refers to a flat, rectangular wall. `Cylindrical` refers to a round pipe wall.

  In the default configuration, with **Conductivity type** parameter set to `Constant` and **Wall geometry** set to `Planar`, the block functions as in previous releases.
- Heat transfer coefficient in the Convective Heat Transfer block can be either constant, which you specify by the **Heat transfer coefficient** parameter, or variable, which you specify as a physical signal at port **K**.

In the default configuration, with **Convection type** parameter set to `Constant`, the block functions as in previous releases.

- The Thermal Mass block can have one or two ports. In some applications, it is customary to display mass or inertia in series with other components in the block diagram layout. To support this use case, you can now display a second port on the block icon. The two-port variant is purely graphical: the two ports have the same temperature, so the block functions exactly the same whether it has one or two ports.

erf and erfc function support for simscape.Value objects

You can now use `erf` and `erfc` when working with `simscape.Value` objects.

For a complete list of supported functions, see “Core MATLAB Functions Supporting simscape.Value Arrays”.

Energy and charge units added to unit registry

Additional units in the default Unit Manager registry in this release include:

- New energy and charge units based on watt-hour and ampere-hour: `Ah`, `mAh`, `kAh`, `MAh`, `Wh`, `mWh`, `kWh`, `MWh`.
- Popular multipliers for other units currently in the registry, such as `MV`, `MA`, `GW`, `mOhm`, `uJ`, and `mJ`.

For a complete list of units in the Unit Manager registry, see “Unit Definitions”.

Simulation

State-Based Consistency Tolerance: Fine-tune the nonlinear solver tolerances used for computing initial conditions and transient initialization

In previous releases, the **Consistency tolerance** parameter of the Solver Configuration block had a numeric value, and the block used a nonlinear solver based on the equation residual tolerance to initialize the model. The default parameter value was applicable to most cases. However, the residual-based computation did not provide a clear way to adjust the parameter value and improve the model initialization.

Now, the block uses state-based absolute and relative consistency tolerances, multiplied by a scaling factor, to compute the initial conditions and for transient initialization. The **Consistency tolerance** parameter of the Solver Configuration block provides a choice:

- `Model AbsTol and RelTol` — Use the model tolerance settings, specified as **Absolute tolerance** and **Relative tolerance** parameters on the **Solver** pane of the Configuration Parameters dialog box.
- `Local tolerance settings` — Replace the model tolerance settings with local values. When you select this option, the **Absolute tolerance** and **Relative tolerance** parameters appear in the Solver Configuration block dialog box.

Independent of whether you use the model tolerances or the local tolerance settings, the new **Tolerance factor** parameter provides a scaling factor for these values. The resulting value determines how accurately the algebraic constraints are to be satisfied at the beginning of simulation and after every discrete event, such as a discontinuity resulting from a valve opening or a hard stop. Decrease the parameter value to obtain a more reliable time simulation. Increase the parameter value if solving for initial conditions failed to converge, or to reduce the computation time.

The new state-based method provides better robustness and efficiency, especially if used in conjunction with scaling the model by nominal values.

Compatibility Considerations

If you open an existing model where the **Consistency tolerance** parameter has a numeric value, the model continues to use the same residual-based computation method that it used in previous releases and your simulation results do not change.

To upgrade your existing models to use the new state-based method, use the **Check Simscape use of state-based consistency tolerances** check in the Upgrade Advisor.

Steady-State Enhancement: Consider mode charts and events for steady-state solution

When you enable steady-state initialization, the solver now selects the steady-state solution that is consistent with the mode charts and event variables present in the model. For more information, see “Finding an Initial Steady State”.

Compatibility Considerations

In previous releases, the solver held mode charts and events in their initial states and found a steady-state solution. The solver then updated mode charts and event variables as a subsequent step. Because the steady-state solution is now consistent with the modes and events, the result can be different than in previous releases.

Multithread Function Evaluation: Speed up fixed-cost simulations that use Backward Euler local solver

If your model uses the Backward Euler local solver, computing Newton iterations is time-consuming, which may present an issue for fixed-cost simulations. You can now use multithread function evaluation to speed up simulation on a multicore machine. The new **Maximum threads for function evaluation** parameter in the Solver Configuration block lets you specify the desired number of threads. The solver reduces the number you specify to the nearest power of 2. For example, if you specify `5` as the parameter value, the solver uses 4 threads. The default, `1`, corresponds to single-thread function evaluation and is equivalent to the algorithm used in previous releases.

This parameter is available only if you select the **Use local solver** check box, set the **Local solver** parameter to `Backward Euler`, and select the **Use fixed-cost runtime consistency iterations** check box.

Additionally, to use multithread function evaluation, you must clear the **Resolve indeterminate equations** check box. This check box is also new in this release. For details, see the Solver Configuration block reference page.

Other unsupported simulation modes for multithread function evaluation include frequency-and-time simulation, delay, scalable compilation, accelerator mode, and rapid accelerator mode. When using multithread function evaluation, you can generate code using Simulink® Real-Time™, but other types of code generation are not supported.

Statistics Viewer Enhancement: Provide a more complete reporting of constraint statistics

Typically, the presence of constraints indicates a high-index problem that requires applying index reduction techniques to simplify the system, in order for it to be solved. In previous releases, the Statistics Viewer, under **Number of dynamic variable constraints**, reported only the number of constraint equations detected during run-time index reduction.

This statistic has been renamed to **Number of constraints**. This statistic now reports the total number of primary constraint equations in the model, which are the equations the index reduction algorithm handles at compile-time or at run-time. If **Number of constraints** is greater than zero, the **Sources** section lists the differential variables involved in each constraint.

Simscape to HDL Compatibility Enhancements: Use averaged switching, input filtering, unit conversions, tablelookup, and mode charts in Simscape models

If you have an HDL Coder™ license, you can take advantage of more Simscape features when generating FPGA code. When you set **Solver type** to `Partitioning`, you can generate HDL code from models that contain:

- Converter blocks with the **Switching device** parameter set to `Averaged switch`. For more information, see *Release Notes for Simscape Electrical* (Simscape Electrical).
- The `tablelookup` function in the connection functions. You must use `interpolation = linear` and `extrapolation = linear`. The tables must have no more than four dimensions. To learn more about connection functions and systems of equations in the partitioning solver, visit "Understanding How the Partitioning Solver Works".
- Integer-valued events and mode charts.

When you set **Solver type** to `Backward Euler`, you can generate HDL code from models that contain Simulink-PS Converter blocks with the **Input filtering order** parameter set to either `First-order filtering` or `Second-order filtering`.

For both solver types, you can generate HDL code from models that contain PS-Simulink Converter blocks with **Output signal unit** set to any unit.

For more information, see *Release Notes for HDL Coder* (HDL Coder).

Functionality being removed or changed

exp function limiting to avoid nonfinite values

Behavior change

If a block uses the `exp` function in its equations, a very large value inside that function can cause the exponential to become infinite and thus introduce Inf or NaN values at initialization time or during simulation.

Starting in R2022b, exponential limiting has been introduced to avoid generating nonfinite values during initialization and simulation. However, exponential limiting is not performed when `exp` is used as an argument inside functions such as `isinf`, `isnan`, or `isfinite`. In these cases, the `exp` function can assume nonfinite values to ensure the correct predicate evaluation.

Compatibility Considerations

This change improves simulation robustness for most models. However, some models can now issue a warning during simulation about the limiting of the exponential.

Change to symbolic integration algorithm

Behavior change

In previous releases, if you simulated a frequency-and-time model that involved only differential variables that were dependent on each other, nonzero initial conditions of such variables were ignored during symbolic integration. In other words, all such variables were assumed to have an initial condition of 0.

Starting in R2022b, the symbolic integration algorithm has been enhanced to account for the initial conditions in this case.

Compatibility Considerations

As a result of this change, some models can now initialize differently than in previous releases.

R2022a

Simscape Language

Scalable Compilation for Textual Components: Reduce compilation time for models containing component arrays

Scalable compilation for models containing a large number of repeated components was introduced in R2021b. For more information, see Scalable Compilation. In the previous release, the reusable components had to be included in the model as either referenced subsystems or linked subsystems. This functionality has now been extended to include textual components. A new attribute, `CompileReuse`, lets you specify whether the components are reusable or not. This attribute is available for components only. If set to `true`, the compilation artifacts for these components are reusable. The default is `false`.

For example, if you model repeated components by using component arrays, you can use the `CompileReuse` attribute to specify which component arrays are reusable:

```
component battery_pack
...
for i =1:Ncells
    components(ExternalAccess=none,CompileReuse=true)
        battery_cell(i) = BatteryPack.battery_cell(cell_mass=cell_mass);
    end
    ...
end
...
for i=1:Ncells-1
    components(ExternalAccess=none)
        conduction(i) = foundation.thermal.elements.conduction(area={1e-3,'m^2'},...
            th_cond={200,'W/(m*K)'});
    end
    ...
end
...
```

If scalable compilation is enabled for a model containing this battery pack component, then the members of the first component array, `battery_cell`, are reusable. Members of the second array, `conduction`, are not designated as reusable because the conduction element in the Foundation library is not complex enough to benefit from scalable compilation.

LoggingUnit Annotation: Specify the preferred display unit for intermediates, variables, inputs, and outputs

A new annotation option, `LoggingUnit`, lets you specify a preferred data logging unit for component members, such as intermediates, variables, inputs, and outputs. If specified, the same unit is also used for other display purposes, such as in the Variable Viewer or in the operating point.

The specified logging unit must be commensurate with the intrinsic unit of the component member, such as the declared unit of a variable, typed input, or typed output. For intermediates and untyped inputs and outputs, the intrinsic unit is computed by the compiler. For example:

```
component mycomp

parameters
  p1 = {10,'lb'};
  p2 = {5, 'ft'};
end
```

```
intermediates
  i = p1*p2;
end

annotations
  i : LoggingUnit = 'J'
end

...
```

Error Function Support: Use erf and erfc in equations

Two more MATLAB functions, `erf` and `erfc`, are now supported in Simscape files.

For a complete list of supported functions, see `equations`.

Foundation Library

Standalone Property Inspector for Simscape Blocks: Unified look and feel of the block user interface

In previous releases, double-clicking a Simscape block opened the block dialog box, where you viewed the block description and modified its parameters. Alternatively, you could view and modify the block properties in the Property Inspector pane in the model window.

Starting in R2022a, when you double-click a Simscape block, the Property Inspector opens in a standalone window. This window has the same content and appearance as the Property Inspector pane in the model window.

To view and modify the block parameter values and initialization targets for the block variables, double-click the block, click the **Settings** tab, and modify the values of the parameters, variable priorities, and targets.

By default, changing a value in the Property Inspector immediately applies the new value. To remove a series of changes, use the Undo button and Redo button in the upper-right corner of the model window.

To manually apply parameter changes, clear the **Auto Apply** check box in the upper-right corner of the Property Inspector to enable the **Reset** and **Apply** buttons. You can use these buttons to apply or reset parameter changes, similar to using the **Apply** and **Cancel** buttons in the block dialog in previous releases.

To view the description of a block, click the **Description** tab. This tab also contains the **Source code** link. Click this link to open the Simscape source file for this block in the MATLAB Editor.

If a block has no parameters or variable targets that can be set, then the Property Inspector has only a **Description** tab.

To view the documentation for a block, click the Help button in the upper-right corner of the Property Inspector.

Note In previous releases, to make the run-time parameter settings visible in block dialogs, you had to set the **Show run-time parameter settings** preference. This check box has been removed from the **Simscape Preferences** pane because the `Compile-time`/`Run-time` drop-down is always visible in Property Inspector.

Network Couplers Library: Split your system into multiple coupled networks with different solver configurations

The new Network Couplers library blocks let you split a Simscape network in your model into multiple coupled networks. Each of these networks can then have its own solver settings. For example, you can use a variable solver for one of the coupled networks and a fixed-step solver for another, or use two fixed-step solvers with different step sizes.

The Network Couplers library is available as a sublibrary in the Utilities library. For more information, see Using Network Couplers to Split Physical Networks.

Vector Format Parameter in PS-Simulink Converter Block: Choose format for outputting vector physical signals

The new **Vector format** parameter in the PS-Simulink Converter block lets you specify how to output vector physical signals:

- `inherit` — Format the Simulink output signal to match the format of the physical signal: scalar, row or column vector, or 2-D matrix. This is the default setting for new models.
- `1-D array` — If the physical signal is a row or column vector, format the output signal as a Simulink 1-D array. This option corresponds to how the PS-Simulink Converter block handled vector physical signals in previous releases. To preserve backward compatibility, in models created prior to R2022a, the **Vector Format** parameter is automatically set to `1-D array`.

Compact Boundaries for Block Display: Improved model layout and readability

Simscape blocks that do not have a solid boundary line around the block icon, such as Resistor or Capacitor, now get a more compact boundary in block diagrams. This enhancement results in improved model layout and better automatic routing of lines. Blocks that have a solid boundary line, such as PS Ramp or Gear Box, are not affected by this change. For detailed information and compatibility considerations, see *Release Notes for Simulink*.

Comment Through Simscape Blocks: Exclude blocks from simulation without physically removing them from model

You can now comment through a Simscape block in a model to temporarily disable and short-circuit the block from simulation. To comment through a block, right-click on the block and select the **Comment Through** option. Check that there are exactly two connection ports and that both the ports are from the same domain.

Functionality being removed or changed

Default amount of entrained air changed for isothermal liquid domain
Behavior change

To improve simulation robustness, the default amount of entrained air for the isothermal liquid domain has been changed from 0 to 0.005. The new value applies to:

- The domain parameter `air_fraction`.
- The default value of the **Volumetric fraction of entrained air in mixture at atmospheric pressure** parameter in the Isothermal Liquid Properties (IL) block.
- The default value of the **Volumetric fraction of air that is entrained at atmospheric pressure** parameter in the Isothermal Liquid Predefined Properties (IL) (Simscape Fluids) block, available with a Simscape Fluids™ license.

Compatibility Considerations

If you specify the working fluid properties by using an Isothermal Liquid Properties (IL) or Isothermal Liquid Predefined Properties (IL) block, as recommended in Specifying the Working Fluid, then these blocks in existing models retain the parameter value saved with the block in the previous release, whether the default value of 0 or a custom value, and the simulation results stay the same.

However, if your model contains isothermal liquid circuits without a fluid properties block, these circuits use the default domain properties and the simulation results for the model might change. To preserve compatibility with previous releases, add an Isothermal Liquid Properties (IL) block to the isothermal liquid circuit and set its **Volumetric fraction of entrained air in mixture at atmospheric pressure** parameter to `0`.

Simulation

Scalable Compilation Enhancement: Reduce compilation time for models containing multiple instances of the same Simscape block

When your model contains a large number of repeated components, such as a transmission line or a battery pack, you can reduce its compilation time by enabling scalable compilation. In the previous release, the reusable components had to be included in the model as either referenced subsystems or linked subsystems. Now, if your model contains multiple instances of the same block, you can use the `simscape.scalable.setBlockConfig` function to make these instances reusable. For more information, see Scalable Compilation.

New Index Reduction Options: Choose nonlinear index reduction method best suited for each network

If your model contains high-index differential algebraic equations (DAEs), you now have a choice of nonlinear index reduction methods. Use the new **Index reduction method** parameter in the Solver Configuration block to select the method best suited for that network:

- `Derivative replacement` — In this method, parts of the DAE are differentiated analytically and appended to the original system. For each additional equation, a derivative is selected to be replaced by a new algebraic variable called a dummy derivative. For more information, see https://epubs.siam.org/doi/abs/10.1137/0914043?journalCode=sjoce3. This option corresponds to the nonlinear index reduction method used in previous releases. It is recommended that you start with this method. This is the default setting.
- `Projection` — Use this option if the `Derivative replacement` method fails due to issues with dynamic state selection.
- `None` — If your model does not contain nonlinear high-index DAEs, use this option to completely bypass nonlinear index reduction and remove the analysis overhead.

Multithread Linear Algebra: Speed up desktop simulations that use local solver and sparse linear algebra

If your model uses a local solver and sparse linear algebra, you can now use multithread linear algebra to speed up desktop simulation on a multicore machine. The new **Number of threads (specify n for 2^n)** parameter in the Solver Configuration block lets you specify the number of threads by providing an integer exponent for 2. The number of threads equals 2 to the power of the parameter value. The default, `0`, corresponds to single-thread linear algebra and is equivalent to the algorithm used in previous releases.

This parameter is available only if you select the **Use local solver** check box and set the **Linear algebra** parameter to `Sparse`. For a global solver, Simulink solves the equations without using Simscape linear algebra algorithms.

For small models, multithread algorithms that use numbers higher than 0 may be slower than single-thread.

simscape.op.Target Enhancement: Use simscape.Value objects to specify operating point targets

When creating or manipulating operating point targets, you no longer specify the value and unit separately. Instead, use `simscape.Value` objects to specify both the value and the unit.

Compatibility Considerations

The underlying operating point data has not changed. However, if you use scripts for programmatic target manipulation, you might need to update portions of the code related to accessing the `Value` property of a `simscape.op.Target` object.

Simscape Hardware-in-the-Loop Workflow Enhancement: Use sign(x) in Simscape models

If you have an HDL Coder license, you can now generate an HDL code from Simscape models that use `sign(x)` in component equations. In previous releases, the use of this function was not supported.

Simscape Variable Scaling Analyzer App: Identify issues with model scaling to improve performance

Now you can access the **Simscape Variable Scaling Analyzer** app from the **Simscape** section of the app library. Previously, you could only access the **Simscape Variable Scaling Analyzer** by using the `simscapeVariableScalingAnalyzer` command. The app performs an analysis of the model variables and equations, highlights problem areas that may cause issues with performance or accuracy, and provides scaling recommendations for nominal values. To learn more, see Select Nominal Values Using the Variable Scaling Analyzer.

New look and feel for Simscape Results Explorer and Variable Viewer

Simscape Results Explorer and Variable Viewer have a new look and feel and a streamlined interface, but all the workflows are similar to those in previous releases.

For more information, see About the Simscape Results Explorer and About Variable Viewer.

Code reuse support in code generation

In previous releases, code generated from Simscape models did not support code reuse. This limitation has now been removed.

New examples

Examples introduced in this version include:

- Configuring an EV Simulation for Multirate HIL
- Pendulum in Cartesian and Polar Coordinates
- PEM Electrolysis System

Functionality being removed or changed

New algorithm for physical signal variable elimination
Behavior change

To speed up simulations, the new algorithm eliminates variables generated by inputs and outputs of blocks from the Physical Signals library from the system of model equations. In most cases, these variables represent operations on known data and therefore including them in the overall system of model equations makes these equations unnecessarily complicated. Eliminating these variables shortens compilation time and increases robustness.

Compatibility Considerations

Because of the new algorithm, some models might require more input derivatives than in previous releases. This issue mostly affects models with conditional high-index equations. If, upon simulating an existing model, you get a request for additional derivatives for a Simulink-PS Converter block, you can:

- Use the input filtering option in the Simulink-PS Converter block.
- Add an additional input signal to the Simulink-PS Converter block to provide the derivative.
- Eliminate the Simulink-PS Converter block by using a physical signal as an input.

Improved simulation accuracy for Partitioning solver in Robust simulation mode
Behavior change

To improve simulation accuracy when using the Partitioning solver in `Robust simulation` mode, more partitions may be generated than in previous releases. The solver uses the new partitions to ensure that all the algebraic constraints are satisfied up to the consistency tolerance.

Compatibility Considerations

In R2021a, when improving the stability of the Partitioning solver in `Robust simulation` mode, one of the changes was to delay the computation of some nonlinear terms in algebraic equations. That change, while improving computation efficiency, in some cases may have led to lower simulation accuracy. The new algorithm does not delay the computation of nonlinear terms. However, because of this change, some models may slow down or fail during simulation. If this happens, you can:

- Use the Partitioning solver in `Fast simulation` mode.
- Use a different local solver.

R2021b

Simscape Language

Improved Code Reuse: Override base class members in derived classes

You can now override certain members of base class in derived classes. For example, you can:

- Override the default values of base class parameters
- Override the default initial values, priorities, and other attributes of base class variables
- Override intermediates declared in the base class
- Override annotation attributes declared in the base class, such as `Icon` or `ExternalAccess` of base class members

For more information, see Overriding Base Class Members in Derived Classes.

You cannot override the `Access` attribute of base class members. For example, if a base class member is declared as `protected`, it stays `protected` in all derived classes.

You cannot override values or attributes of base class members declared as `private`.

In previous releases, if a parameter or variable referenced other parameters, it had to be declared as `protected` or `private`. This restriction has now been removed. Instead, if the `ExternalAccess` attribute of a parameter or variable derived from other parameters is `modify`, the compiler issues a warning and sets the `ExternalAccess` attribute to `observe`. For example:

```
component Base
  parameters
    p1 = 1;
  end
  variables
    x1 = p1;
    x2 = p1*2;
  end
...
end

component Sub
< Base(x1 = 0)
...
end
```

Component `Sub` overrides the value of the base class variable `x1`, so it no longer depends on parameter `p1`. However, variable `x2` references parameter `p1` and cannot be modified independently. Therefore, for the component `Sub`, the compiler leaves the `ExternalAccess` attribute of `x1` as `modify`, but sets `ExternalAccess` of `x2` to `observe` and issues a warning.

Also, in previous releases, members of the base class with `Access=private` were always forced to have `ExternalAccess=none`, to avoid potential collision of names between the base class and the derived class. This restriction has now been relaxed. A member of the base class with `Access=private` is forced to have `ExternalAccess=none` only if a member from the derived class is declared with the same name. Lifting this limitation allows data logging for conditionally declared members of a base class. For more information, see Defining Conditional Visibility of Component Members.

Foundation Library

Reservoir (2P) and Controlled Reservoir (2P) Blocks: Specify fluid boundary conditions using extended set of options

The two-phase fluid reservoir blocks, Reservoir (2P) and Controlled Reservoir (2P), now have additional options for specifying the fluid boundary conditions with the quantities most appropriate for your applications. When modeling closed-loop systems, use these blocks to break the loop apart, to model and validate each segment separately before connecting the segments together.

hydraulicToIsothermalLiquid Conversion Tool Enhancement: Easily upgrade models containing customized hydraulic blocks

To facilitate conversion of models containing customized hydraulic blocks, such as masked library blocks or custom blocks written in Simscape language, the `hydraulicToIsothermalLiquid` conversion tool now additionally lets you specify two cell arrays of custom block names, `oldcustomblocks` and `newcustomblocks`, after all the other input arguments. `oldcustomblocks` contains the names of the customized hydraulic blocks to replace. `newcustomblocks` contains the names of customized isothermal liquid blocks to use as replacements. The two cell arrays must have the same number of elements. The respective blocks listed in each array must have the same number of ports, matching port order, and the same programmatic parameter names.

If, during conversion, the tool encounters a block listed in `oldcustomblocks`, then the tool replaces that block with the block listed as the respective element in `newcustomblocks`.

Before you can use this syntax, prepare the equivalent isothermal liquid version of the customized blocks:

- For custom library blocks and subsystems that contain blocks from the Foundation > Hydraulic library or Fluids > Hydraulics (Isothermal) library, run the conversion tool on these custom libraries. In previous releases, if your custom library blocks were masked, the conversion tool discarded the mask. Now, when the tool converts a masked hydraulic block or subsystem, it retains the mask. You still need to verify that the number of ports, port order, and the mask parameter names match between the hydraulic and the isothermal liquid versions.
- For custom hydraulic blocks written in Simscape language, manually create equivalent versions of these blocks that use the isothermal liquid domain. If the custom hydraulic blocks were masked, you can add masks to the custom isothermal liquid blocks as well. Make sure that the isothermal liquid blocks have the same number of ports, matching port order, and the same programmatic parameter names as the original hydraulic domain blocks.

simscape.Value and simscape.Unit Objects: Use MATLAB interface to manipulate physical values with units

In physical modeling, block parameters, variables, and physical signals are represented as a value with associated unit. Simscape unit manager automatically performs the necessary unit conversion operations when solving a physical network. However, if you wanted to write simple MATLAB programs to do physical computations (such as postprocessing simulation data), performing mathematical operations on values with units was not possible in previous releases. You had to strip the Simscape values from the associated units, perform computations, and then manually perform the necessary unit conversions and reattach the new unit to the data. This process was cumbersome and error-prone.

`simscape.Value` and `simscape.Unit` objects essentially implement a MATLAB interface that replicates the unit manager functionality outside of Simscape:

- `simscape.Value` binds arrays of arithmetic values to units and propagates those units through mathematical operations. All members of an array must have the same unit.
- `simscape.Unit` represents units of measure without an associated value, and therefore lets you write MATLAB functions that emulate the unit propagation behavior.

Use `simscape.Value` and `simscape.Unit` to:

- Preprocess or postprocess simulation data with units in MATLAB, for example, calculate total fluid mass or plot vehicle dynamics.
- Create value with unit objects in MATLAB and manipulate them during programmatic model construction.
- Write MATLAB functions that operate on values with units.

For more information, see Working with simscape.Value and simscape.Unit Objects.

You can also use `simscape.Value` objects to create operating point targets. For more information, see Use simscape.Value to Create an Operating Point Target.

Interface Specification for Simscape Connections: Lock down connection types for Simscape Bus and Connection Port blocks

A new Simulink object, `Simulink.ConnectionBus`, lets you design rigid interface specifications for conserving connections. When you apply such rigid specification to a Simscape Bus or Connection Port block, the block ports become typed by the interface and do not accept connections to a different domain type.

To construct `Simulink.ConnectionBus` objects, add `Simulink.ConnectionElement` objects and specify the names and domain types for these connection elements. You can construct or modify these objects:

- Programmatically
- Using the Simulink Bus Editor
- Using the Model Explorer

To apply an existing connection bus specification to a Simscape Bus or Connection Port block, use the **Connection type** parameter and select the bus name from the drop-down list.

To remove the rigid bus specification, set the **Connection type** parameter on the block to `Inherit: auto`.

For more information, see Design Rigid Interface Specifications for Conserving Connections.

System Composer Support for Simscape Models: Create physical interfaces, ports, and connections on architecture components

If you have a System Composer™ license, you can now create physical interfaces, ports, and connections on architecture components and implement physical behaviors using the System Composer software together with the Simscape family of products.

Simulation

Scalable Compilation: Reduce compilation time for models containing repeated reusable components

When your model contains a large number of repeated components, such as a transmission line or a battery pack, you can reduce its compilation time by enabling scalable compilation. In order to benefit from scalable compilation, the repeated components must be included in the model as either referenced subsystems or linked subsystems. Scalable compilation helps reduce compilation time for such models by compiling a repeated component once, and then reusing these compilation artifacts for other instances of the same component. For more information, see Scalable Compilation.

Simscape Variable Scaling Analyzer Tool: Identify issues with model scaling to improve performance

The **Simscape Variable Scaling Analyzer** tool performs an analysis of the model variables and equations, highlights problem areas that may cause issues with performance or accuracy, and provides scaling recommendations for nominal values. To learn more, see Select Nominal Values Using the Variable Scaling Analyzer.

Stiffness Impact Analysis Tool Enhancement: Perform stiffness analysis at multiple time points

In previous releases, the Stiffness Impact Analysis tool performed the stiffness analysis of a model at initialization time only. Now you can specify multiple time points during simulation.

simscape.getLocalSolverFixedCostInfo function: Expedite model conversion to fixed-cost

When converting your model to fixed-cost, you can now easily determine the value for the **Nonlinear iterations** parameter of the Solver Configuration block in your model using the `simscape.getLocalSolverFixedCostInfo` function.

Data Logging Support for Rapid Accelerator Mode: Use rapid accelerator mode to simulate models with data logging enabled

In previous releases, data logging was available only in normal simulation mode or accelerator mode. If you wanted to simulate your model in rapid accelerator mode, you had to turn data logging off.

Now you can log simulation data when running the simulation in rapid accelerator mode. All the data logging workflows available in the normal simulation mode are now available in rapid accelerator mode as well, with one exception. Simulink Compiler does not support Simscape data logging.

Statistics Viewer Enhancement: View and trace secondary variables for 1-D physical systems

The **1-D Physical System** node in the Statistics Viewer contains information on the number of variables in the system, with separate statistics for different categories of variables: continuous, discrete, differential, algebraic, and so on. During the compilation process, a model undergoes multiple transformations, with some variables being eliminated and other, secondary, variables being added to the system to make it solvable. The Statistics Viewer includes eliminated variables statistics as a separate subcategory. In this release, statistics on secondary variables have also been added.

Secondary variables are generated by the compiler. In previous releases, you could see them as Simulink states after running the simulation, but their names were cryptic. Now these secondary variables have meaningful names and descriptions that let you trace the variable origin and understand the transformation that introduced it. When you select a statistic for secondary variables, the names and descriptions of these variables appear in the **Sources** section of the Statistics Viewer:

- The **Source** column contains the variable path, including the top-level model and the name of the primary variable, with a link to the block containing the primary variable.
- The **Value** column contains the secondary variable description.

Improved Handling of Implicit Asserts: Avoid runtime errors without impacting performance

You can now use `isinf`, `isnan`, and `isfinite` without triggering asserts when developing Simscape library blocks. Instead, you can take advantage of the non-finite value protection that these functions offer.

For scalar values, expressions such as

```
x == if (isfinite(x1./x2)), x1./x2 else 0 end
```

no longer trigger asserts that generate a runtime error. This feature applies to all operations that output `nan` or `inf` values in Simulink, including

- dividing by zero.
- taking the root of negative numbers.
- taking the logarithm of nonpositive numbers.
- using `arcsin` or `arccos` with numbers above one.
- raising negative numbers to non-integer powers or raising zero to negative powers.

Simscape ensures that the results are finite. Note that when using array values, these functions return an array of logicals. You must convert these logicals to scalars when using conditional expressions.

Simscape enforces finite intermediates, but when you use intermediates within `isinf`, `isnan`, or `isfinite`, Simscape introduces duplicate intermediates to allow `isinf`, `isnan`, and `isfinite` function correctly. Therefore, these functions may signal a non-finite value while the results show the original intermediate as finite. Simscape does not support array values with intermediate expressions.

Functionality being removed or changed

Specifying complex numbers as parameter values generates an error
Behavior change

Simscape blocks and functions do not support complex numbers. In previous releases, if you specified a complex number as a block parameter value, the compiler used the real part of the complex number as the parameter value and ignored the imaginary part, which sometimes led to inconsistent behavior.

Starting in R2021b, if you assign complex numbers, such as a noninteger power of a negative number, as parameter values, the model generates an error upon simulation.

Compatibility Considerations

To preserve compatibility with previous releases, change the parameter value to the real part of the complex number previously assigned.

R2021a

Simscape Language

Variable Initialization Ranges: Specify acceptable minimum and maximum values for variable initialization targets

In previous releases, when declaring a variable, you could specify its target value, unit, and initialization priority. Now you can additionally specify the minimum and maximum acceptable values for variable initialization, for example:

```
variables
  x = {value={0,'deg'},priority=priority.high,imin={0,'deg'},imax={360,'deg'}};
end
```

When multiple initialization solutions exist, this syntax lets you guide the solver towards the preferred solution. For more information, see Block-Level Variable Initialization. If the specified range cannot be satisfied during initialization, the solver issues an error.

The solver tries to satisfy the initialization range for a variable regardless of whether its initialization priority is high, low, or none. It is recommended that you use the `priority` attribute sparingly. The default priority value, `priority.none` (which is equivalent to leaving out the `priority` attribute entirely), is suitable in most cases. The block user can modify the variable priority value, as needed, in the **Variables** tab of the block dialog box prior to simulation. However, the block user does not have control over the variable initialization range. Only the block author can specify the acceptable minimum and maximum values for variable initialization in the component file, both for continuous and for event variables.

When declaring custom domains, you can specify initialization ranges for domain Across variables, for example, to exclude negative values for pressure or temperature.

The default initialization range is `(-inf,inf)`. Therefore, you do not have to specify both values to define the range, it is sufficient to specify only `imin` or `imax`. For example, use this syntax to limit the temperature to positive values:

```
variables
  T = {value={293.15,'K'},imin={0,'K'}};
end
```

When you specify `imin` or `imax`, these values define an open range.

For model initialization using an operating point, the solver tries to satisfy initialization ranges only for variables that do not have a target in the operating point data tree. For more information, see Using Operating Point Data for Model Initialization.

Table Lookup for 1-D Array of Query Points: Specify table lookup query arguments as row or column vectors

The `tablelookup` function lets you perform grid-based interpolations in Simscape blocks. The `tablelookup` function supports one-dimensional, two-dimensional, three-dimensional, and four-dimensional lookup tables. The full syntax is:

```
tablelookup(x1d, x2d, x3d, x4d, fd, x1, x2, x3, x4, interpolation = linear|
smooth, extrapolation = linear|nearest|error)
```

`x1d`, `x2d`, `x3d`, `x4d`, and `fd` are the data sets that form the grid. `x1d`, `x2d`, `x3d`, and `x4d` are one-dimensional arrays representing the input data points along each of the axes. `fd` is the array of output function values corresponding to these input data points, with the array size matching the dimensions defined by the input data sets. For example, for two-dimensional table lookup, if `x1d` is a 1-by-*m* array, and `x2d` is a 1-by-*n* array, then `fd` must be an *m*-by-*n* matrix.

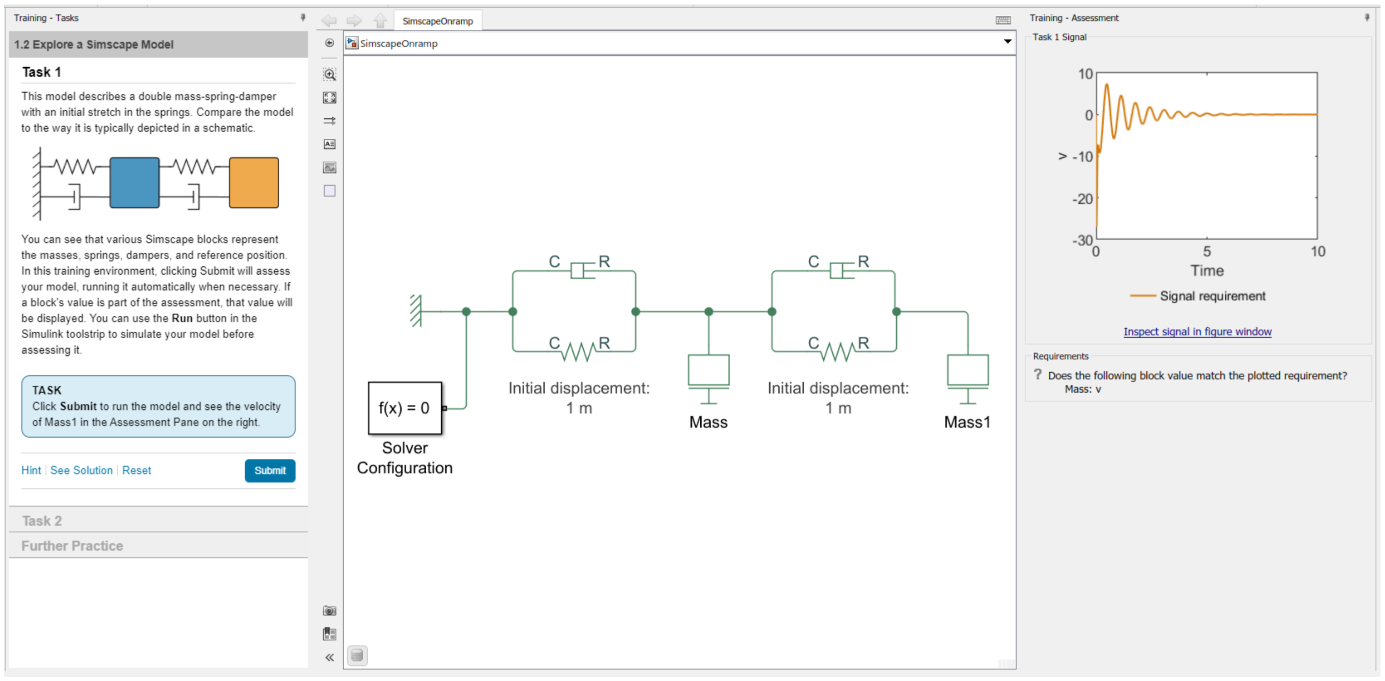
`x1`, `x2`, `x3`, and `x4` are input values that represent the coordinates of the query point along each direction. In previous releases, you could specify a single query point, which meant that each of these query input values had to be a scalar, and the function returned a scalar value `f` by interpolating the query point against the grid.

Now you can perform table lookup for a one-dimensional array of query points, by specifying `x1`, `x2`, `x3`, and `x4` as row or column vectors of the same size (1-by-*k* or *k*-by-1). The function performs the interpolation of each of these query points against the grid and returns a one-dimensional array of the same size as the input query vectors.

Foundation Library

Simscape Onramp: Self-paced, interactive tutorial for getting started with Simscape

Simscape Onramp is a self-paced, interactive tutorial that helps you get started with Simscape. To teach concepts incrementally, Simscape Onramp uses hands-on exercises. You receive automated assessments and feedback after submitting tasks.



Task 1

This model describes a double mass-spring-damper with an initial stretch in the springs. Compare the model to the way it is typically depicted in a schematic.

You can see that various Simscape blocks represent the masses, springs, dampers, and reference position. In this training environment, clicking Submit will assess your model, running it automatically when necessary. If a block's value is part of the assessment, that value will be displayed. You can use the **Run** button in the Simulink toolstrip to simulate your model before assessing it.

TASK
Click **Submit** to run the model and see the velocity of Mass1 in the Assessment Pane on the right.

? Does the following block value match the plotted requirement?
Mass: v

Your progress is saved if you exit the training, so you can complete the training in multiple sessions.

To open Simscape Onramp, use one of these options:

- On the Simulink Start Page, click **Simscape Onramp**.
- In the MATLAB Command Window, enter `learning.simulink.launchOnramp('simscape')`.

Simscape Onramp:

- Introduces Simscape models, including an RC circuit and rotational mass damper
- Explores simulation results for physical quantities using sensor blocks
- Models physical systems with external inputs or initial values
- Demonstrates interactions between multiple physical domains with examples of electromechanical conversion, fluids, and hydroelectric power
- Implements feedback control with Simscape and Simulink
- Helps you practice what you learn with an electronic valve project

Multibody Interface Blocks: Connect Simscape networks to Simscape Multibody joints

You can model 3-D mechanical systems by using Simscape Multibody™ blocks that represent bodies, joints, constraints, force elements, and sensors. These models can also include Simscape networks that represent hydraulic, electrical, pneumatic, and other physical systems. However, you cannot directly connect Simscape mechanical rotational and translational ports to Simscape Multibody blocks. You need to establish bidirectional connections between the Simscape portions of a block diagram and Simscape Multibody joints that involve both sensing and actuation, and in some cases must pass position information as well.

The Mechanical library now has a Multibody Interfaces sublibrary, which contains blocks that let you connect Simscape blocks with mechanical translational or rotational ports to Simscape Multibody joints:

- The Translational Multibody Interface block matches the force and relative velocity across the interface. You can connect it to any Simscape Multibody joint that has a prismatic primitive.
- The Rotational Multibody Interface block matches the torque and relative angular velocity across the interface. You can connect it to any Simscape Multibody joint that has a revolute primitive.

For more information, see Connecting Simscape Networks to Simscape Multibody Joints.

Additionally, the following actuator blocks now have an optional input port that lets you pass position information from a Multibody joint:

- Translational Mechanical Converter (IL)
- Rotational Mechanical Converter (IL)
- Translational Mechanical Converter (TL)
- Rotational Mechanical Converter (TL)
- Translational Mechanical Converter (2P)
- Rotational Mechanical Converter (2P)
- Translational Mechanical Converter (G)
- Rotational Mechanical Converter (G)
- Translational Mechanical Converter (MA)
- Rotational Mechanical Converter (MA)
- Translational Hydro-Mechanical Converter
- Rotational Hydro-Mechanical Converter

For more information on how to enable and use this input port, see How to Pass Position Information.

hydraulicToIsothermalLiquid Conversion Tool Enhancement: Easily upgrade models containing model references, subsystem references, and library links

To facilitate conversion of models containing model references, subsystem references, and library links, the `hydraulicToIsothermalLiquid` conversion tool now additionally accepts either a list of files or a path to the top-level folder as an input argument. The tool converts all the files that contain hydraulic blocks, while preserving the links between them, and returns the list of converted files.

Like in previous releases, the conversion tool does not overwrite the original files, but saves the new file under a different name, by appending the `_converted` suffix to the original file name. Unlike in previous releases, if a file does not contain hydraulic blocks, the tool leaves it unchanged and does not create a copy with the `_converted` suffix.

If you specify a list of files or a path to the top-level folder as an input argument, the conversion tool updates all the library links, model references, and subsystem references in these files to point to the `_converted` versions of these files.

Once you are happy with the conversion, you can use the new `hydraulicToIsothermalLiquidPostProcess` function to restore the original file names. This function accepts either a list of converted files or a path to the top-level folder as an input argument. The function warns you that it will overwrite the original files, and then removes the `_converted` suffixes and restores all the library links, model references, and subsystem references.

For more information, see Upgrading Hydraulic Models To Use Isothermal Liquid Blocks.

Compatibility Considerations

In previous releases, the conversion tool always created a new file by appending the `_converted` suffix to the original file name. If a file did not contain hydraulic blocks, the original file and the converted file were identical. Now, if a file does not contain hydraulic blocks, the tool leaves it unchanged and does not create a copy with the `_converted` suffix.

In previous releases, the tool returned the block diagram name of the converted model, subsystem, or library, as a character vector. Now, even if converting a single file, the tool always returns a cell array, where each element is a converted block diagram name. The block diagram name is the file name without the file path or extension.

Saturation Properties Sensor (2P) Block: Measure liquid and vapor saturation properties

The new Saturation Properties Sensor (2P) block in the Two-Phase Fluid > Sensors library lets you measure liquid and vapor saturation properties:

- Saturated temperature, specific volume, internal energy, or enthalpy, based on pressure measurement
- Saturated pressure, based on temperature measurement

Physical Signals Library Enhancements: Expand modeling capabilities using physical signal blocks

Several enhancements in the Physical Signals library provide additional modeling options within a Simscape network, using physical signal blocks rather than Simulink signals. These enhancements include:

- Three new blocks in the Physical Signals > Sources library, PS Ramp, PS Sine Wave, and PS Step, let you generate a physical signal of the desired shape, rather than use a Simulink signal and a Simulink-PS Converter block.
- The PS Integrator block is now resettable at events. You can specify that the block output resets to the initial condition when the physical signal at port **R** triggers a reset. The initial condition

source can now be either a physical signal or a block parameter. If you set the **Initial condition source** parameter to `External`, the physical signal at port **X0** supplies the initial condition value for integration and reset. In its default configuration, with **External reset** parameter set to `None` and **Initial condition source** set to `Internal`, the block functions exactly as in previous releases.

- The PS Math Function block has an additional option, `u.^v`, where **v** is a block parameter. When you use this option, the value of the output signal is the value of the input signal, *u*, to the power of *v*. The unit of the output signal is the result of the operation of the function on the input signal unit. Therefore, unless the input signal is unitless, changing the value of **v** changes the unit of the output signal.

PS Lookup Table Block Enhancement: Use row or column vectors of query points as input and output signals

The `tablelookup` function now lets you perform table lookup for a one-dimensional array of query points, by specifying `x1`, `x2`, `x3`, and `x4` as row or column vectors of the same size (1-by-*k* or *k*-by-1). The function performs the interpolation of each of these query points against the grid and returns a one-dimensional array of the same size as the input query vectors. For more information, see “Table Lookup for 1-D Array of Query Points: Specify table lookup query arguments as row or column vectors” on page 4-2.

As a result, the PS Lookup Table (1D), PS Lookup Table (2D), PS Lookup Table (3D), and PS Lookup Table (4D) blocks now also accept one-dimensional arrays (row or column vectors) of query points as their input signals. For multidirectional table lookup, all the input vectors must be of the same size. The block output signal in this case is also a row or column vector of the same size. See the block reference pages for details.

Expanded Support for Power Dissipation Data: View power_dissipated simulation logging data for several Foundation library blocks

Several blocks in the Foundation library now calculate dissipated power and output it in their simulation data log under the `power_dissipated` node. You can now analyze data for a `power_dissipated` logging node for these blocks:

- Capacitor
- Diode
- Inductor
- Resistor
- Switch
- Thermal Resistor
- Variable Resistor
- Rotational Damper
- Translational Damper
- Rotational Friction
- Translational Friction

For more information on logging, viewing, and analyzing simulation data, see `sscexplore`. Also, if you have a Simscape Electrical™ license, you can use functions like `ee_getEfficiency` (Simscape

Electrical), `ee_getPowerLossSummary` (Simscape Electrical), and `ee_getPowerLossTimeSeries` (Simscape Electrical) to collate and analyze power dissipation data for your model.

Variant Connector Block Enhancement: Limit scope of variant control variables of Variant Connector blocks and include Simscape Bus blocks in model variants

You can now use the Variant Connector block with mask and model workspace variables and Simscape Bus blocks.

- Previously, you could define the variant control variable of the Variant Connector block only in a base workspace or a data dictionary. From R2021a onward, you can also define the variant control variable in the mask and the model workspace. This functionality limits the scope of the variables, which helps you avoid name conflicts and establish clear ownership of the variable between the blocks. It also enables you to use the same names for variables in different scopes. For more information, see Mask Workspace Variable in Variant Connector Block.
- You can now place Simscape Bus blocks in a leaf region or a bounded region formed by a Variant Connector block. For more information, see Variant Connector Block with Simscape Bus Block.

Simulation

daessc Solver Enhancement: Use daessc solver as default variable-step solver for new models

When you first create a model, the default Simulink solver is `VariableStepAuto`. Auto solver chooses a suitable solver as described in Select Solver Using Auto Solver. For new models created in this release and beyond, if your model contains Simscape blocks and Differential Algebraic Equations (DAEs), auto solver defaults to the `daessc` solver.

The `daessc` variable-step Simulink solver, which was designed specifically for physical modeling, was introduced in R2018b. In R2021a, this solver has additional options that let you fine-tune the solver performance. In the **Solver** pane of the Configuration Parameters dialog box, if **Solver** is set to `daessc (DAE solver for Simscape)`, the new **Daessc mode** parameter under **Solver details** contains the following drop-down list:

- `auto` — Automatically selects the optimal `daessc` solver mode. This is the default setting. In R2021a, `auto` is equivalent to `Balanced`, but this may change in future releases.
- `Fast` — The most efficient mode in terms of computation cost, but less robust.
- `Balanced` — Provides a balance between computational costs and robustness.
- `Robust` — More robust, but also more costly. This option is equivalent to the way `daessc` solver worked in previous releases.
- `Quick debug` — Updates the solver Jacobian at every integration step, and is therefore even more costly than `Robust`. Recommended only for interactive model development, to quickly find issues with equations.
- `Full debug` — Updates the solver Jacobian at every integration step and every Newton iteration. This mode is the most expensive in terms of computational cost. Recommended only for interactive model development, to thoroughly check equations and find possible issues.

Compatibility Considerations

In previous releases, the default `VariableStepAuto` solver for models containing Simscape blocks and DAEs was `ode23t`. If you open an existing model saved with `VariableStepAuto`, the solver selection does not automatically change to `daessc` in R2021a. The model continues to use the same auto solver that it used in previous releases and your simulation results will not change.

To upgrade your existing models to use `daessc` as the default auto solver, use the **Check integration method used by 'auto' solver for Simscape DAEs** check in the Upgrade Advisor.

Run-Time Parameters Editable in Fast Restart: Change run-time parameter values in the Property Inspector while in fast restart mode

You can now change run-time parameter values in the block dialogs or the Property Inspector window while simulating in fast restart mode. Once you restart the simulation, the new values will take effect. You can still specify run-time parameters as workspace variables, if you choose.

Partitioning Solver Enhancement: Improve stability of Partitioning solver

When you use the `Robust simulation` mode of the Partitioning local solver, more equations are now solved using the Backward Euler scheme. This enhancement results in increased solver stability and extends the range of models that can be simulated using the Partitioning solver.

Simscape Hardware-in-the-Loop Workflow Enhancement: Use Partitioning solver in Simscape models

If you have an HDL Coder license, you can now generate an HDL implementation model from Simscape models that use a Partitioning solver. For more information, see *Release Notes for HDL Coder* (HDL Coder).

Data Logging Support for Accelerator Mode: Use accelerator mode to simulate models with data logging enabled

In previous releases, data logging was available only in normal simulation mode. If you wanted to simulate your model in accelerator or rapid accelerator mode, you had to turn data logging off. Now you can log simulation data when running the simulation in accelerator mode. All the data logging workflows available in the normal simulation mode are now available in accelerator mode as well.

simscape.logging.timestamp Function: Determine whether the simulation log is current or stale

You can now determine whether the `simlog` object in your workspace is current or stale by using the `simscape.logging.timestamp` function. The function accepts a `simscape.logging.Node` object as the input argument and returns the model timestamp associated with the node.

Numerical Solve Enhancement: Support wider range of high-differential-index problems for equations with vector variables

The Simscape solver now supports a wider range of high-differential-index problems for equations that include vector variables. This means that many models involving the three-phase electrical domain and using Simscape Electrical blocks no longer require you to add parasitic components or leakage paths in order to avoid numerical simulation issues. For example, wye-connected inductors (which are used as a building block in many of the three-phase machine models) no longer require a parasitic conductance to ground at the common connection.

For more information, see Differential Index and Avoiding Numerical Simulation Issues.

Improved Conservation Laws: Better conserve physical quantities, such as momentum and charge, at events

The new algorithm for deriving conservation laws from the DAE system results in improved simulation behavior at events. For example, you can see improvements in momentum conservation for mechanical systems and in charge conservation for electrical systems.

The algorithm handles a wide range of events, but has some limitations. For example, it does not handle high-differential-index problems with index greater than two, or differential equations that involve both differential and algebraic variables in the same nonlinear term.

Compatibility Considerations

Because of the new algorithm, your simulation results might be slightly different for some models:

- If, in the previous releases, there was an event where momentum or charge was not conserved properly during simulation, the results may change when the new algorithm ensures proper conservation of these quantities.
- If you are providing a wrong value of a derivative signal by using a Simulink-PS Converter block, the simulation results may change because the derivative information is now used differently. Therefore, if you notice a difference in results, check your model to ensure that the derivatives provided using the Simulink-PS Converter blocks are correct.

New examples

Examples introduced in this version include:

- Nonlinear Electromechanical Circuit with Partitioning Solver
- PEM Fuel Cell System
- Lithium Pack Thermal Runaway
- Variant Connector Block with Simscape Bus Block
- Mask Workspace Variable in Variant Connector Block
- Lead-Acid Battery with Dashboard Blocks
- Hydraulic Actuator with Analog Position Controller and Dashboard Blocks

R2020b

Simscape Language

State Reset: Reinitialize state variables at events

Event-based methods of state reinitialization and impulse handling let you model physical phenomena such as collisions and bouncing balls. Using state reset methods provides a significant boost in simulation speed for such models.

To implement a state reset, mode charts can now contain instantaneous modes and compound transitions. An instantaneous mode is a mode that is active only for one event iteration. You specify that a mode is instantaneous by using a compound transition:

```
A -> B -> C : t
```

The middle mode, `B`, is instantaneous. When predicate `t` becomes true, the system transitions from mode `A` to mode `B`, performs one event iteration, and then immediately transitions to mode `C`.

You declare instantaneous modes the same way as regular modes, using the `mode` section of a mode chart. To specify that a mode is instantaneous, list it as the middle mode in a compound transition. Only one instantaneous mode is allowed per transition, therefore, a compound transition cannot contain more than three modes.

In the majority of state reset use cases, the reset value is a function of the previous value of the variable. For example, when modeling a bouncing ball, the new velocity depends on the velocity before impact. The new `entry` section, declared within a `mode` section in a mode chart, lets you specify the actions to be performed upon entering the mode. These actions let you perform event variable updates based on the value of the respective continuous variable immediately before entering the mode. You can use entry actions both in instantaneous and regular modes.

When you connect multiple ideal components that use state reset, the solver automatically detects and propagates impulses in continuous states during variable reinitialization. Therefore, impulse detection can add computational cost during transient initialization. Two new options in the Solver Configuration block, **Compute impulses** and **Impulse iterations**, let you control the computational cost of impulse detection during transient initialization.

For more information, see State Reset Modeling.

Extended Functionality for Component Arrays: Observe array members during simulation

Arrays of components in Simscape language were introduced in R2020a. In R2020b, this functionality has been extended:

- You can now set the `ExternalAccess` attribute of array members to `observe`, which makes their variables visible in simulation logs, Variable Viewer, Statistics Viewer, and so on.
- You can use command-line interface to index into an array of components, for example, to plot logged simulation data for an array member, or to create operating point targets. For more information, see Indexing into Component Arrays.

Extended for-Loop Functionality: Include equations in for-loops

In R2020a, when arrays of components were introduced, `for`-loops could be used only in the declaration section, to declare an array of components, and the structure section, to declare the component connections. In R2020b, the `for`-loop functionality has been extended to also include equations.

Block Layout Annotation: Group and reorder parameters in the block interface

In previous releases, custom Simscape blocks displayed the underlying component parameters on a single tab of the block dialog box, titled **Parameters**. The parameters were listed on this tab in declaration order.

A new class-level annotation, `UILayout`, lets you make the custom block layout more flexible by defining titled groups of component parameters. When you deploy the component as a custom Simscape block, these groups translate into dialog box tabs (as well as Property Inspector tree nodes). This annotation also defines the order of the tabs and the order of parameters in each tab. The annotation does not affect component variables: they are still listed on a separate **Variables** tab in declaration order.

ssc_build Enhancement: Specify name and location for generated custom libraries

You can now use an optional name-value pair argument with `ssc_build` to specify the output library name and location. If you omit this argument, the custom library generated by `ssc_build` is automatically named *`package`*`_lib` and automatically placed in the package parent directory, the same as in previous releases.

Foundation Library

Variant Connector Block: Selectively enable branches of a physical network

The new Variant Connector block in the Utilities library lets you define bounded or leaf regions in a physical network and then enable or disable them for simulation. Use the Variant Connector block to define model variants, by including or excluding certain parts of the model.

Mass and Inertia Blocks with Two Graphical Ports: Enhance flexibility of block diagram layout

In some applications, it is customary to display mass or inertia in series with other components in the block diagram layout. To support this use case, you now have an option to display a second port on a Mass or Inertia block icon. The two-port variant is purely graphical: the two ports have the same velocity, so the block functions exactly the same whether it has one or two ports.

Isothermal Liquid Model Template: Create new isothermal liquid models by using ssc_new or model templates

New **Isothermal Liquid** template is now available in the Simscape section of the Simulink Start Page. The new template contains additional domain-specific blocks on the model canvas.

The `ssc_new` function has also been updated to provide an equivalent domain option. Typing

```
ssc_new('MyModel','isothermal_liquid')
```

creates a new model called `MyModel` by using the **Isothermal Liquid** template.

Sensor Blocks Enhancement: Eliminate unused variables

The source code for the following blocks has been streamlined to take advantage of the latest Simscape language capabilities:

- Current Sensor
- Voltage Sensor
- Hydraulic Flow Rate Sensor
- Hydraulic Pressure Sensor
- Flux Sensor
- MMF Sensor
- Ideal Force Sensor
- Ideal Torque Sensor
- Ideal Rotational Motion Sensor
- Ideal Translational Motion Sensor

As a result, some unnecessary variable declarations have been removed. The block equations have been rewritten to refer directly to the output, where possible. This enhancement results in increased compilation speed due to more efficient variable elimination. The block functionality and simulation results are unchanged, compared to the previous releases.

Compatibility Considerations

If you were logging a variable that has been eliminated, update your scripts by replacing this variable with the corresponding output node. For example, if you were logging variable `v1` in the Voltage Sensor block, replace it with the output `V`.

Functionality being removed or changed

Spectrum Analyzer block default averaging method changed

Behavior change

Starting in R2020b, by default, the Spectrum Analyzer block uses the exponential averaging as the averaging method. The default value of the **Averaging method** parameter is now `Exponential`.

Compatibility Considerations

For existing models with a Spectrum Analyzer, the averaging method is not changed. For new visualizations with the Spectrum Analyzer block, you may see some changes in the output because of the averaging method. If you want to use the previous defaults, set **Averaging method** to `Running`.

Simulation

Stiffness Impact Analysis Tool: Analyze effect of particular block variables on overall system stiffness of a Simscape network

When you use an explicit solver, the simulation may become unstable because the system is stiff. For more information, see Explicit Versus Implicit Continuous Solvers. In most models, the instability can be removed by changing the block parameter values to reduce system stiffness. The new Stiffness Impact Analysis tool lets you analyze the model and determine which of the variables and equations have the most impact on system stiffness. You can then modify the values of parameters involved in these equations to make the system less stiff.

To access the Stiffness Impact Analysis tool, select the **Simscape Stiffness** check box in the **Solver Profiler** toolstrip. The **Simscape Stiffness** tab in the bottom pane lists variables with the most impact on the system stiffness and lets you determine, for each of these variables, the corresponding stiffness coefficient, as well as the block and equation where the variable is used.

Stiffness analysis is performed at initialization time only.

The tool performs stiffness analysis of the Simscape networks only. If a model does not contain Simscape blocks, the tool does not show any results.

Variable Viewer Support for Component Arrays: View variable initialization data for array members

When your model contains blocks with underlying arrays of components, Variable Viewer now includes variables that belong to array members. For more information, see Component Array Representation in Variable Viewer.

Operating Point Support for Component Arrays: Set and get operating point targets for array members

When your model contains blocks with underlying arrays of components, operating point data now includes variables that belong to array members. You can use command-line interface to index into an array of components, for example, to get or set operating point targets for a particular array member. For more information, see Indexing into Component Arrays.

Statistics Viewer Support for Component Arrays: View model statistics for array members

When your model contains blocks with underlying arrays of components, Statistics Viewer now includes data on array members. For more information, see Model Statistics for Component Arrays.

Data Logging Support for Component Arrays: Log and plot simulation data for array members

When your model contains blocks with underlying arrays of components, you can now log and plot simulation data for array members. Simulation data for array members is logged independent of the logging method, whether you use logging to workspace or stream logged simulation data to disk.

You can use either Simscape Results Explorer or Simulation Data Inspector to view logged simulation data for individual array members. For more information, see Data Logging for Component Arrays.

You can also use command-line interface to index into an array of components, for example, to plot logged simulation data for a particular array member. For more information, see Indexing into Component Arrays.

Data Logging Enhancements: Improve simulation performance and usability

In this release, multiple enhancements have been implemented for data logging.

You can now access `simscape.logging.Node` objects using block names, instead of IDs, by using the new `get` object function. This object function supports tab completion when you type the path to node. The path must start with a block name, for example:

```
n = get(simlog_ssc_dcmotor,'DC Motor/Friction/C');
```

`simscape.logging.findNode` and `simscape.logging.findPath` have been optimized to perform much faster. As a result, simulation performance when streaming logged simulation data to disk has been significantly improved. The file format for streaming logged simulation data to disk has been changed from HDF5 to MLDATX. MLDATX files are binary files, commonly used by the Simulation Data Inspector.

Also, series data is now automatically reshaped to match the dimensions of the logged variable. In previous releases, if you logged an *n*-dimensional variable, the series data was logged as a one-dimensional array of *n***T* length, where *T* is the number of time steps. Now, the series data for such variable is automatically reshaped as an *n*-by-*T* array.

Compatibility Considerations

The file format when streaming logged simulation data to disk has changed from HDF5 (H5) to MLDATX. As a result:

- When you import an H5 file from a previous release using `simscape.logging.import`, the data is automatically converted to the new format, loaded into the `Node` object, and copied to a binary MLDATX file in the temporary directory. If there are two files with the same name and different extensions (H5 and MLDATX), then the MLDATX file is loaded.
- When you use `simscape.logging.export` without specifying the file extension, the data is stored in an MLDATX file in the temporary directory.
- Currently, you can still export data in H5 format, to share it with colleagues who are using older software releases. To do this, specify H5 file extension when using `simscape.logging.export`. The function stores the data in H5 format and issues a warning that this functionality will be removed in a future release. If you specify a file extension other that H5 or MLDATX, you get an error.

There is a slight change to the logging output from multidimensional variables because of the automatic data reshaping. If your scripts relied on the series data being a one-dimensional array, you have to update these scripts. If you applied manual reshaping to logged simulation data in previous releases, your results will not change in this release.

daessc Solver Enhancement: Improve simulation performance

Enhancements made to the `daessc` solver in this release result in improved simulation performance, with more than a 10% speed increase in the majority of test cases.

The `daessc` variable-step Simulink solver, which was designed specifically for physical modeling, was introduced in R2018b. To select this solver, open the **Solver** pane of the Configuration Parameters dialog box, set **Type** to `Variable-step`, and then from the **Solver** drop-down list, select `daessc (DAE solver for Simscape)`.

Functionality being removed or changed

Initial equations ignored when initializing from operating point
Behavior change

Initial equations provide an alternative way to specify high-priority variable targets for model initialization. However, when you initialize a model from an operating point, especially one that was generated from logged simulation data, the operating point is likely to contain all the necessary high-priority targets. In this case, applying initial equations results in an over-specified model. Moreover, if you initialize the model by applying an operating point extracted from logged simulation data at time t, variable targets supplied by initial equations for $t=0$ are no longer applicable.

In previous releases, the solver applied all the targets and then tried to resolve the conflict if the model became over-specified. Now, if you initialize a model from an operating point, the solver ignores all the initial equations that contain variables present in the operating point data. Initial equations for other variables are still applied: for example, if you add a block to the model after extracting the operating point data, initial equations for this block will be executed at initialization time. For more information, see Initial Equations and Using Operating Point Data for Model Initialization.

Compatibility Considerations

If your model uses both initialization from an operating point and blocks with initial equations, your initialization results may be slightly different than in previous releases.

R2020a

Simscape Language

Component Arrays: Define components using resizable arrays of elements

You can now use arrays of components and `for`-loop constructs in Simscape language. This feature lets you create composite components that contain an arbitrary number of homogeneous members, such as segmented pipelines, battery packs, or transmission lines.

Rules and restrictions:

- Arrays apply only to the `component` member class.
- Array members must have the `ExternalAccess` attribute set to `none`.
- Array members must all belong to the same class. However, they can have different parameter values.
- The array size can be a parameter with the `ExternalAccess` attribute set to `modify`, which means that the block users can modify this value.
- Use `for`-loops to declare component arrays and to connect members of the array to each other.
- Use nested `for`-loops to create multidimensional arrays of components.
- You cannot include conditional sections inside `for`-loops.

For more information, see Component Arrays.

Compatibility Considerations

Some models with `let` and `if` expressions that used to work in previous releases may generate errors in this release, as a result of improved semantics consistency of these expressions and tightening of the lookup algorithms. For example, if an identifier in the declaration clause of a `let` expression clashes with a member name in the same component, the old algorithm was able to resolve the name clash, but this was inconsistent with general name lookup rules. The new lookup algorithm, once it finds the identifier in the declaration clause, does not look outside the scope of the `let` expression. This behaviour is consistent with other instances of name resolution semantics in the language.

Local Simscape Functions: Define functions to be used within a specific component, domain, or function

Simscape functions, introduced in R2017b, are intended to reuse expressions in equations and member declarations of multiple components. Each of these functions must be in a separate Simscape file, with the file name matching the function name.

In contrast, local Simscape functions reside inside a Simscape file that defines a component, domain, or another function, and are accessible only by that component, domain, or main function. For example, when you need to use a function in a single component only, defining it as a local function:

- Reduces the overhead of creating and packaging separate files.
- Restricts access, to ensure that only that specific component can use this function.

For more information, see Local Simscape Functions.

Annotation for Port Location: Place ports on all four sides of the block

In previous releases, you customized both the name and location of a block port by supplying a comment immediately after the respective node, input, or output declaration. This method allows you to place ports only on two opposite sides of the block icon, either left and right or top and bottom, for example:

```
nodes
    H = foundation.thermal.thermal; % H:left
    p = foundation.electrical.electrical; % +:left
    n = foundation.electrical.electrical; % -:right
end
```

Now you can use the `annotations` section to specify port locations on the block icon. This method allows you to place ports on all four sides of the block. For example, you can place the thermal port at the top of the block icon, while keeping the two electrical ports on the left and right side, as before:

```
nodes
    H = foundation.thermal.thermal;
    p = foundation.electrical.electrical; % +
    n = foundation.electrical.electrical; % -
end
annotations
    H : Side = top;
    p : Side = left;
    n : Side = right;
end
```

You still customize the port label using the comment immediately after the declaration, similar to specifying meaningful names for block parameters and variables.

The old method of using comments to specify port locations continues to work. However, using the `annotations` section is the recommended method, because it provides more flexibility.

If you specify multiple locations for a port by using multiple annotations, you get a compilation error. If you specify multiple locations for a port by using both the new and the old method, the location specified in the `annotations` section takes precedence. For more information, see Control Port Locations Using Annotations.

Compatibility Considerations

The old method of using comments to specify port locations continues to work. There is no change to existing models saved in previous releases.

Going forward, if you use a combination of the new and old methods to specify port locations in the same component file, consider the following:

- If you specify multiple locations for the same port by using both the new and the old method, the location specified in the `annotations` section takes precedence.
- In the old method, if you specify ports on the top and bottom of the block, the block icon is rotated by 90 degrees during the library build. This behavior is preserved. However, if you add a `Side` annotation to such a component, the block icon is no longer rotated.

Foundation Library

Isothermal Liquid Domain and Block Library: Model isothermal hydraulic systems based on mass flow rate and pressure

The Foundation library now contains an isothermal liquid domain and the Isothermal Liquid block library. This library contains chambers, reservoirs, local restrictions, energy converters, sources, and sensors. The block equations assume that fluid temperature remains constant during the simulated time interval. The library also contains the Isothermal Liquid Properties (IL) block, which controls fluid properties for the attached circuit.

Use these blocks to model hydraulic power and control systems where temperature fluctuations are small, such as hydraulic actuators operating at nearly constant temperature.

For more information, see Modeling Isothermal Liquid Systems and the block reference pages. Also see Isothermal Liquid Domain for information about the isothermal liquid domain definition.

In the isothermal liquid domain, the Across variable is absolute pressure and the Through variable is mass flow rate. (In the hydraulic domain, the variables are gauge pressure and volumetric flow rate.) Note that the product of pressure and mass flow rate is not power, and therefore the result is a pseudo-bond graph. However, using mass flow rate, instead of volumetric flow rate, as the Through variable reduces the potential for small errors in mass conservation to accumulate over time (due to conversion between mass and volumetric quantities) and thus results in increased accuracy.

It is recommended that, going forward, you use the Isothermal Liquid library for modeling isothermal hydraulic systems. Unlike the Hydraulic library blocks, the pipe and actuator blocks in the Isothermal Liquid library account for fluid compressibility by default, which reduces the likelihood of dry nodes. Also, all blocks in the Isothermal Liquid library account for fluid density being a function of pressure, as opposed to only certain blocks in the Hydraulic library. The Isothermal Liquid library makes it easy to specify your working fluid by selecting the bulk modulus model and the desired option for modeling the amount of entrained air, and have these selections reflected in all the block equations. For more information, see Isothermal Liquid Modeling Options.

The Hydraulic library and the hydraulic domain definition are still included with the software, and all the hydraulic blocks in your models continue to work as before. At this point, there are no plans to remove them. However, using Isothermal Liquid library blocks to model isothermal hydraulic systems provides improved usability, increased accuracy, and enhanced simulation robustness. For more information, see Upgrading Hydraulic Models To Use Isothermal Liquid Blocks.

hydraulicToIsothermalLiquid Conversion Tool: Upgrade your hydraulic models to use isothermal liquid blocks

The new Isothermal Liquid block libraries are structured similar to other fluid domains, such as Thermal Liquid, to take advantage of the latest Simscape language features and to improve block usability. As a result, often there is no one-to-one correspondence between the Isothermal Liquid and Hydraulic library blocks and you cannot easily substitute one for another.

The `hydraulicToIsothermalLiquid` conversion tool facilitates the model upgrade process by automatically replacing hydraulic blocks from the Foundation library with equivalent isothermal liquid blocks, while preserving the parameter values and connections between the blocks, where possible. The tool saves the converted model under a new name and generates an HTML report that lists any issues encountered during the conversion process. For more information, see Upgrading Hydraulic Models To Use Isothermal Liquid Blocks.

Interface (H-IL) Block: Combine hydraulic and isothermal liquid networks in one model

The new Interface (H-IL) block in the Hydraulic Utilities library represents a flow connection between hydraulic and isothermal liquid networks:

- Pressure and mass flow rates are equal at the interface.
- Fluid properties are not shared across the interface. Each network has its own fluid properties.

Use this block in existing models with Hydraulic library blocks to implement parts of the model using Isothermal Liquid library blocks, without converting the whole model.

Probe Block: Output block variables as signals during simulation

The new Probe block in the Utilities library lets you select variables from another block in the model and output them as Simulink signals. The following rules apply:

- The selected block must be at the same level of the model hierarchy as the Probe block.
- After selecting the block, you can select only the variables exposed on its **Variables** tab (that is, the same variables that can be used for block-level variable initialization).
- Each of the selected variables is output as a separate signal.
- The Probe block outputs Simulink signals. Therefore, you can connect it directly to Simulink blocks, like scopes or buses.
- You can attach (bind) a Probe block to only one block at a time. In other words, you can bind a Probe block to a block in a model, simulate, and then bind the Probe block to another block. However, you can bind multiple Probe blocks to the same block in the model at the same time.

PS Transfer Function Block: Model low-pass and lead-lag filters

The new PS Transfer Function block in the Physical Signals/Linear Operators library lets you model first-order low-pass filters or lead-lag filters as a Simscape block. Modeling both the plant and controller in a Simscape network improves simulation efficiency.

Previously, if you wanted to model a low-pass and lead-lag filter as part of a Simscape network, you had to use a combination of PS Integrator, PS Gain, and PS Subtract blocks. Such a model could become complicated, especially for any-order lead-lag filters, and building it required knowledge of control systems.

The PS Transfer Function block lets you easily parameterize a filter in the desired configuration:

- `Lag` — Model a first-order low-pass filter. You have to provide the lag time constant and the initial output as block parameters.
- `Lead-lag` — Model a lead-lag filter of any order. You have to provide the lead time constant, the lag time constant, and the initial output as block parameters.

The block assumes unity low-frequency gain. To model non-unity gain, connect a PS Gain block in series with the PS Transfer Function block.

New Option for Ideal Rotational Motion Sensor Block: Keep angle measurement range between 0 and 360 degrees

The Ideal Rotational Motion Sensor block has a new parameter, **Wrap angle to [0, 2*pi]**. When set to `On`, it keeps the sensor angular displacement output within the range from 0 to 2π radians (360 degrees), regardless of the number of revolutions performed by the object and the direction of rotation. The default is `Off`.

Using this option simplifies development of models with complex relationships between model parameters and the rotation angle, such as pumps and motors.

Simulation

Run-Time Parameters Enhancement: Refer to structure fields in parameter expressions

Simscape run-time parameters have been enhanced to make them more consistent with the Simulink run-time parameter functionality. As a result, both structures and cell arrays are now supported. For example, run-time parameter expressions can now refer to structure fields, such as `spring.theta`.

Numerical Solve Enhancement: Support wider range of high-differential-index problems for scalar equations

Simscape solver now supports a wider range of high-differential-index problems for scalar equations. This means that, in many cases, your models no longer require adding parasitic components to avoid numerical simulation issues. For example:

- Nonlinear capacitors connected in parallel no longer require a parasitic series resistance.
- Delta-connected nonlinear capacitors (often used in transistor models) no longer require parasitic series resistances.
- Nonlinear inductors connected in series no longer require a parasitic parallel conductance.
- Wye-connected nonlinear inductors (often used in machine models) no longer require a parasitic conductance to ground at the common connection.

For more information, see Differential Index (MATLAB) and Avoiding Numerical Simulation Issues.

Frequency-and-Time Simulation Performance Improvement: Simulate with reduced compilation times

The frequency-and-time equation formulation now uses a more efficient method for variable elimination. This enhancement results in reduced compilation times, particularly for large models.

Partitioning Solver Performance Improvement: Simulate without recomputing linear solutions for each step

In previous releases, for linear partitions with a large number of states, the Partitioning solver computed the solution for linear equations, switched linear, and nonlinear equations at each time step in the simulation. Now the computational cost at each step is lower because the Partitioning solver computes the linear solution for the simulation only at compile time. The lower computational cost yields faster simulation times for the Partitioning solver.

Simscape Hardware-in-the-Loop Workflow Enhancement: Generate HDL implementation model from multiple Simscape networks

If you have an HDL Coder license, you can now generate an HDL implementation model from multiple Simscape networks. For more information, see *Release Notes for HDL Coder* (HDL Coder).

Simulink Toolstrip Changes: New location for Statistics Viewer and Variable Viewer

The Statistics Viewer and Variable Viewer, previously available on the **Apps** tab of the Simulink Toolstrip, are now located on the **Debug** tab, in the **Diagnostics** section.

Tool	R2019b	R2020a
Statistics Viewer	On the **Apps** tab, in the **Physical Modeling** gallery, click **Simscape Statistics Viewer**.	On the **Debug** tab, click **Diagnostics** > **Simscape** > **Statistics Viewer**.
Variable Viewer	On the **Apps** tab, in the **Physical Modeling** gallery, click **Simscape Variable Viewer**.	On the **Apps** tab, click **Diagnostics** > **Simscape** > **Variable Viewer**.

Improved Interoperability: Support UTF-8 encoding of Simscape files

When using **Save as**, you now have a choice of encodings available for Simscape files, including UTF-8. Automatic character set detection, also implemented in this release, ensures that localized Simscape files load and execute correctly across platforms and locales.

Table lookup algorithm unification

The Simscape table lookup functionality, which is used by `tablelookup` and blocks in the Lookup Tables library, has been streamlined and unified with the table lookup functionality in MATLAB and Simulink. This change increases computation efficiency.

Compatibility Considerations

Consistent Akima interpolation algorithms are now used for both Simscape and MATLAB computations. As a result, the numerical results of Akima (smooth) interpolation by `tablelookup` and blocks in the Lookup Tables library might be slightly different than in previous releases.

Private variables now included in operating points extracted from data logged using the Simulation Data Inspector

In previous releases, when you logged simulation data by recording it in the Simulation Data Inspector, the simulation log did not contain private Simscape language data. Therefore, if you extracted an operating point from data logged using the Simulation Data Inspector, private data was not included.

This limitation has now been removed. When you extract an operating point from data logged using the Simulation Data Inspector, private variables are now included by default in the operating point data. For more information, see hasPrivateData.

Functionality being removed or changed

simscape.logging.sli.findNode is being removed
Warns

`simscape.logging.sli.findNode` will be removed in a future release. Use `simscape.logging.findNode` instead. The syntax and arguments of the two functions are identical, but the new function is more efficient.

simscape.logging.sli.findPath is being removed
Warns

`simscape.logging.sli.findPath` will be removed in a future release. Use `simscape.logging.findPath` instead. The syntax and arguments of the two functions are identical, but the new function is more efficient.

R2019b

Simscape Language

sscnewfile function: Use Simscape file templates to create custom components, domains, or functions

The `sscnewfile` function lets you create a new Simscape file based on a library of predefined templates for components, domains, and functions, or based on an existing Simscape file. Conceptually, it is similar to the `ssc_new` function, which creates a new Simscape model based on a library of predefined templates. Just as a model created using `ssc_new` contains the required and commonly used blocks for each domain, a Simscape file created using `sscnewfile` contains the required keywords and constructs that help you get started.

The function takes the name of the new component, domain, or function as the first argument in single or double quotes. This name is also used as the file name. The second argument is optional, and can either indicate the domain type (similar to `ssc_new`) or provide the full path and name of an existing Simscape file to be used as a template for the new file.

Examples:

- `sscnewfile('MyComponent')` creates a component named `MyComponent` using the default component template and saves it as `MyComponent.ssc` in the current folder.
- `sscnewfile('MyResistor', 'electrical')` creates a component named `MyResistor` using the default component template for the electrical domain and saves it as `MyResistor.ssc` in the current folder.
- `sscnewfile('MyGasDomain', 'foundation.gas.gas')` creates a domain named `MyGasDomain` using the Foundation gas domain as a template and saves it as `MyGasDomain.ssc` in the current folder. You can then modify the domain parameters and properties to suit your application.

Initial Equations: Specify equations to be executed during model initialization only

A new attribute, `(Initial=true)`, lets you specify equations that should only be executed during model initialization.

Regular component equations alone are not sufficient to initialize a DAE system. Consider a system with *n* continuous differential variables and *m* continuous algebraic variables. During simulation, this system has *n+m* degrees of freedom and must provide *n+m* equations. The initialization problem has up to *n* additional unknowns corresponding to the derivative variables. Until now, these additional unknowns were satisfied by using initial targets for block variables. Initial equations provide another way to initialize a system.

In general, the maximum number of high-priority targets you can specify is equal to the number of additional unknowns in the initialization problem. Besides the unknowns coming from differential variables, the initialization problem also has one more unknown for each event variable. These additional unknowns determine the maximum combined number of initial equations and high-priority variable targets. If there are too many high-priority targets, they cannot all be met. For more information, see Block-Level Variable Initialization.

Because the default value of the `Initial` attribute for equations is `false`, you can omit this attribute when declaring regular equations:

```
equations (Initial = true)  % initial equations
  [...]
end

equations (Initial = false) % regular equations
  [...]
end

equations % regular equations
  [...]
end
```

Initial equations use the same syntax as regular equations, except:

- `der(x)` is treated as an unknown value.
- `delay` and `integ` operators are not allowed.

For more information, see Initial Equations.

Compatibility Considerations

In previous releases, event variables were not part of the initial solve and event variable targets always had high priority. In R2019b, event variables are treated the same as any other type of variable during the initial solve. Therefore, their default priority is none, and you might have to explicitly declare them as high-priority to achieve the previous model behavior during initialization.

All Simscape libraries have been updated to use the correct priority for event variables. A new Upgrade Advisor check, **Check usage of Simscape event variables with unspecified priority**, flags legacy models where a custom component has an event variable with an unspecified priority that is used outside of a `when` clause. This situation can lead to different initialization results. Review the model and the underlying source code and modify it, if necessary.

Foundation Library and Simulation

Simulink Toolstrip: Simscape apps and contextual tab

In R2019b, the Simulink Toolstrip replaces the Simulink Editor menu bar. See “Simulink Toolstrip: Access and discover Simulink capabilities when you need them” for more details. The location of several Simscape tools and features has changed. Some items are now located in contextual that appear only when you need them.

To access the **Simscape Block** tab, select a Simscape block in your model. This tab provides shortcuts to items that were previously available only by right-clicking the block, such as block variants, logging and viewing simulation data, and viewing source code.

Note Most of the blocks in the Utilities library do not open the **Simscape Block** tab because actions like viewing their source code, or logging and viewing simulation data, are not applicable in their context. Therefore, these blocks open the regular **Block** contextual tab when selected.

This table lists Simscape tools and features that were formerly accessed through the Simulink Editor menu bar and maps them to their new locations in the Simulink Toolstrip.

R2019a Simulink Editor Menu Bar Item	Simulink Toolstrip Equivalent
Display > Simscape > Domain Styles	On the **Debug** tab, select **Information Overlays > Simscape Domains**.
Display > Simscape > Legend	On the **Debug** tab, select **Information Overlays > Simscape Legend**.
Display > Simscape > Toggle Sparklines When Clicked	• On the **Debug** tab, in the **Tools** section, in the **Output Values** button group, expand . In the **Sparkline Plots (Simscape)** section, click **Enable Sparkline Plots**. • On the **Simscape Block** tab, in the **Review Results** section, click **Sparkline Plot > Enable Sparkline Plots**.
Display > Simscape > Remove All Sparklines	• On the **Debug** tab, in the **Tools** section, in the **Output Values** button group, expand . In the **Sparkline Plots (Simscape)** section, click **Remove Sparklines**. • On the **Simscape Block** tab, in the **Review Results** section, click **Sparkline Plot > Remove Sparklines**.
Analysis > Simscape > Statistics Viewer	On the **Apps** tab, in the **Physical Modeling** gallery, click **Simscape Statistics Viewer**.
Analysis > Simscape > Variable Viewer	On the **Apps** tab, in the **Physical Modeling** gallery, click **Simscape Variable Viewer**.

Connection Label Block: Reduce diagram clutter by using virtual connections between conserving ports

The new Connection Label block in the Utilities library lets you specify virtual connections between conserving ports, similar to the Goto and From blocks in Simulink diagrams. Physical connection lines are nondirectional, therefore, both sides of the virtual connection use the same type of block.

The virtual connection is established by the label name. If two or more Connection Label blocks in a model subsystem have the same label, then the conserving ports of other blocks connected to them behave as though they were physically connected. Virtual connections cannot cross subsystem boundaries.

Use the Connection Label block to reduce diagram clutter by breaking off tangled connection lines.

Run-Time Parameters for Gas, Moist Air, Thermal Liquid, and Two-Phase Fluid Blocks: Modify parameter values without regenerating C code

The underlying source code for the Gas, Moist Air, Thermal Liquid, and Two-Phase Fluid library blocks has been streamlined and enhanced in this release, and now takes advantage of the latest Simscape language features. As a result, all block and domain parameters are now run-time capable. For more information, see Manage Simscape Run-Time Parameters.

Conditional Port Visibility for Gas, Moist Air, and Thermal Liquid Blocks: Expose additional ports in block variants

The underlying source code for the Gas, Moist Air, and Thermal Liquid library blocks has been updated to take advantage of the conditional port visibility feature that was implemented in R2019a. As a result, these block libraries have been streamlined by combining similar blocks that differed primarily by the number of ports:

- Constant volume chambers in each domain can now have between one and four ports. The four-port option is new. If a chamber has four ports, you can use it as a junction in a cross connection. Separate two-port and three-port blocks are no longer available. This change has no compatibility impact. All two-port and three-port chamber blocks in existing models are automatically replaced by a new block with the appropriate number of conserving ports.
- Local restrictions can now have an optional input physical signal port, to model the variable restrictions of flow area in each domain. Separate variable restriction blocks are no longer available. This change has no compatibility impact. All variable restriction blocks in existing models are automatically replaced by local restriction blocks with an exposed control port.

Additionally, all Moist Air library blocks with a finite moist air volume now have a new parameter, **Moisture and trace gas source**, which controls the visibility of port **S** and provides these options for modeling moisture and trace gas levels inside the component:

- `None` — No moisture or trace gas is injected to or extracted from the block. Port **S** is hidden. This is the default.
- `Constant` — Moisture and trace gas are injected to or extracted from the block at a constant rate. The same parameters as in the Moisture Source (MA) and Trace Gas Source (MA) blocks become available in the **Moisture and Trace Gas** section of the block interface. Port **S** is hidden.

- `Controlled` — Moisture and trace gas are injected to or extracted from the block at a time-varying rate. Port **S** is exposed. Connect the Controlled Moisture Source (MA) and Controlled Trace Gas Source (MA) blocks to this port.

For more information, see Modeling Moisture and Trace Gas Levels.

Compatibility Considerations

The conditional visibility of port **S** for Moist Air library blocks makes the Moisture & Trace Gas Cap (MA) block obsolete. In previous releases, you needed this block to cap unused ports **S** in the model. Now, ports **S** should be exposed only when in use.

Currently, legacy models using this block work the same as in previous releases. However, this block will be removed in the future. To update your legacy models, in each block connected to a Moisture & Trace Gas Cap (MA) block, set the **Moisture and trace gas source** parameter to `None`. Then delete the Moisture & Trace Gas Cap (MA) blocks and unused connection lines.

Additional Measurement Option for Thermodynamic Properties Sensor (MA) Block: Measure thermodynamic properties based on a unit mass of dry air

In previous releases, the Thermodynamic Properties Sensor (MA) block measured each quantity, such as mixture specific enthalpy, as the total amount of that quantity in a volume of moist air mixture divided by the total mass of moist air mixture in that volume. This approach is applicable both for general multi-species gas modeling and HVAC modeling.

However, the American Society of Heating, Refrigerating and Air Conditioning Engineers (ASHRAE) standards contain property data normalized by the mass of dry air, because in HVAC modeling, where moisture gets added and removed, it is more convenient to normalize the mixture properties by a fixed mass.

Therefore, the Thermodynamic Properties Sensor (MA) block now has a parameter, **Measurements based on**, that lets you select the measurement convention:

- `Unit mass of moist air mixture` — For each thermodynamic property, the measured mass of that property in a moist air volume is divided my the mass of the most air mixture. This option is the same as in previous releases.
- `Unit mass of dry air and trace gas` — For each thermodynamic property, the measured mass of that property in a moist air volume is divided my the mass of dry air and trace gas (leaving out the mass of the water vapor). Use this option when you need to compare your measurements with ASHRAE figures and charts.

Two-Phase Fluid Properties Visualization: Plot the two-phase fluid domain data

You can plot domain data specified using the Two-Phase Fluid Properties (2P) block in your model. Plotting the properties lets you visualize the data before simulating the model.

To plot the data, right-click a Two-Phase Fluid Properties (2P) block in your model and, from the context menu, select **Foundation Library > Plot Fluid Properties (3D)** or **Foundation Library > Plot Fluid Properties (Contours)**.

Simulink-PS Converter and PS-Simulink Converter Block Enhancements: Input filtering and unit propagation

The following usability enhancements have been implemented in this release:

- For the Simulink-PS Converter block, the block icon now indicates input filtering. Block parameter options have not changed, but the icon appearance changes depending on the **Input filtering order** parameter option selected.
- For the PS-Simulink Converter block, the new **Input signal unit** parameter option, `inherit`, automatically sets the unit at the block output port to match the unit of the input physical signal coming into the block, based on unit propagation rules. This enhancement lets you easily connect a PS-Simulink Converter block to any signal, without worrying about setting the commensurate output unit.

  `inherit` is now the default value of the **Input signal unit** parameter for new PS-Simulink Converter blocks. For existing instances of PS-Simulink Converter blocks, in models created prior to this release, the **Input signal unit** parameter value stays unchanged.

See the block reference pages for details.

Unconnected Conserving Ports: Treat unconnected conserving ports as open circuit

The restriction that disallowed unconnected conserving ports in Simscape models has been lifted. Now, if you leave a conserving port unconnected, the physical network sets all the Through variables at this port to 0.

You no longer need to cap unconnected conserving ports with terminator blocks for each domain, such as Open Circuit, Perfect Insulator, and so on. However, you can still use these blocks to improve the diagram readability. You can also use these blocks to set the initial targets for the node variables, to assist with model initialization. See the block reference pages for details.

There are no plans to remove the terminator blocks. All the existing models using these blocks work the same as in previous releases.

Additional Plotting Option for Data Logging: Plot logged data for a node in the Simulation Data Inspector

The `simscape.logging.plot` function has a new name-value pair argument that lets you plot logged simulation data for a Node object in the Simulation Data Inspector. The name-value pair is `'viewer','datainspector'`. This argument is also valid for the `plot` function of a Node object. The argument is ignored if the object is a Series.

Memory Caching: Improve model compilation performance

Memory caching is now used for model compilation artifacts, which can lead to shorter compilation times for subsequent compilations of the same model during the same MATLAB session. This enhancement results in improved performance for repeated model simulations and block diagram updates.

Lookup Algorithm Enhancements: Improve lookup performance and consistency

Lookup algorithm enhancements, implemented in this release, provide fast and consistent results that can lead to improved compilation performance and avoid name clashes.

Compatibility Considerations

In the new algorithm, a local identifier always has precedence over classes, objects, and packages on the path. Therefore, models that relied on namespace resolution in previous releases might no longer compile in this release. For example, if you have a component that declares a member `m` and also calls a function located in a package `+m`, compilation will fail to find the function `m.foo` because the local identifier for member `m` has precedence over the package name.

R2019a

Simscape Language

annotations Section: Define conditional visibility of ports and parameters in block variants

The new `annotations` section in the component file lets you control visibility of component members, such as parameters and nodes, in block icons and dialog boxes.

When you declare a component member, the `ExternalAccess` attribute sets the visibility of the member in the user interface, that is, in block dialog boxes, simulation logs, variable viewer, and so on. The `annotations` section serves a similar purpose, but it is especially useful for block variants because it lets you define conditional visibility of component members, based on a predicate condition.

When you define component variants using conditional declarations, certain parameters or ports can be used in one block variant, but not in others. Suppose you have a component that models hydraulic pipelines with circular and noncircular cross sections. For a circular pipe, you need to specify the internal diameter. For a noncircular pipe, you need to specify the hydraulic diameter and pipe cross-sectional area. You can now use the `annotations` section to control the visibility of these parameters in the block dialog box:

```
component MyPipe
  parameters
    circular  = true;             % Circular pipe?
    d_in      = { 0.01, 'm' };    % Pipe internal diameter
    area      = { 1e-4, 'm^2' };  % Noncircular pipe cross-sectional area
    D_h       = { 1.12e-2, 'm' }; % Noncircular pipe hydraulic diameter
  end
  if circular
  % Hide inapplicable parameters
    annotations
       [area, D_h] : ExternalAccess=none;
    end
    equations
       % First set of equations, for circular pipe
    end
  else
  % Hide inapplicable parameter
    annotations
       d_in : ExternalAccess=none;
    end
    equations
       % Second set of equations, for noncircular pipe
    end
  end
  [...] % Other parameters, variables, branches, equations
end
```

Similar to other types of conditional declarations, the predicate of a conditional annotation must be a parametric expression that evaluates to true or false. However, there is an additional restriction that all the parameters used in the predicate of a conditional annotation must be either of type logical or enumerated. In this example, the `circular` parameter is of type logical.

The `annotations` section also lets you specify conditional custom icons. This is especially useful if the number of ports changes for different variants. For example:

```
component MyPipe
  parameters
    thermal_variant = false; % Model thermal effects?
  end
  if thermal_variant
  % Use icon with additional thermal port
    annotations
       Icon = 'pipe_thermal.jpg';
    end
  else
  % Use regular icon, with two fluid ports
    annotations
       Icon = 'pipe.jpg';
    end
  end
  [...] % Other parameters, variables, nodes, branches, equations
end
```

For more information on using custom block icons, see Customize the Block Icon.

Physical Signal Propagation: Specify units for physical signals and propagate them through the model

Simscape blocks can contain Physical Signal input and output ports, directional ports that carry signals with associated units. These ports are defined in the `inputs` and `outputs` declaration blocks of a component file. In previous releases, you could declare each input or output only as a value with unit.

To facilitate unit propagation, you can now also declare inputs and outputs as untyped identifiers and use these identifiers in the component equations. This way, the unit and size of the physical signal are propagated through the components that perform physical signal manipulation.

For example, this component implements a custom block that multiplies the input signal. The **Gain** parameter specifies the multiplication factor.

```
component MyGain
     parameters
         gain = {1, '1'}; % Gain
     end
     inputs
         I; % :left
     end
     outputs
         O; % :right
     end
     equations
          O == gain * I;
     end

     annotations
         gain : UnitDropdown = common
     end
 end
```

Input `I` and output `O` are declared as untyped identifiers. The signal type at port **I** is determined by the connection. For example, if port **I** is connected to the output port **F** of an Ideal Force Sensor block, which outputs the signal in N, then the input signal unit at port **I** is also N. The MyGain block propagates the physical signal type by multiplying the value and unit of the input signal by the value and unit specified for the **Gain** parameter. By default, the unit of the **Gain** parameter is `1` (unitless), which means that the output signal at port **O** has the same unit as the input signal, N. However, if the block user specifies the **Gain** parameter unit as `m`, the physical signal at port **O** has the unit of N*m.

The `annotations` entry specifies the units prepopulated in the drop-down list for the **Gain** parameter. `UnitDropdown = common` includes a list of common units, such as those available in the

Simulink-PS Converter and the PS-Simulink Converter block dialog boxes. The drop-down list also contains a field where the block user can type a unit name or expression. For more information, see How to Specify Units in Block Dialogs.

Compatibility Considerations

- Untyped inputs and outputs have been implemented for blocks in the Physical Signals library. For compatibility considerations, see “PS Block Library with Unit Propagation: Propagate physical signal units throughout the model” on page 8-7.
- The new library with unit propagation retains the name Physical Signals, but the package name is different. It is now `+foundation/+signal`. The old package (`+physical_signal`) is no longer part of the Foundation library. If your custom composite components use blocks from the old package, you have to update the package name and also ensure that the unit propagation works as expected.
- Arguments passed to rounding functions (`round`, `ceil`, `floor`, `fix`) must be dimensionless. Previously, these functions accepted a value with unit as an argument, but did not handle units. For example, if *x* is a length, in cm, `round(x)` would round *x* to the nearest whole number, but *x* may have been converted to a different length unit before rounding. With unit propagation, such behavior can lead to unexpected results.

  Now you have to pass a dimensionless argument to these functions, by converting to units of `1` before applying the function. In the example above, instead of `round(x)`, you now have to use `round(value(x,'cm'))`, or equivalently `round(x/{1,'cm'})`.

Parameterized Inputs and Outputs: Specify signal size by referencing a parameter

Besides declaring untyped inputs and outputs, you can also reference component parameters in input and output declarations. This enhancement lets you control the signal size by using a block parameter. For example:

```
component MyTransformer
     parameters
         N = 3; % Number of windings
     end
     inputs
         I = {zeros(N, 1), 'A'};
     end
     ....
 end
```

Run-Time Domain Parameters: Modify domain parameter values without regenerating C code

In previous releases, only component parameters were run-time capable. Domain parameters are now run-time capable as well. You can specify domain parameters as `Run-time` by using the drop-down in the respective source component that controls the domain parameter value. For Foundation domains, the source component is typically located in the Utilities sublibrary of the respective block library. For example, the Gas Properties block controls the values of the gas domain parameters. For more information on specifying block parameters as run-time configurable, see Manage Simscape Run-Time Parameters.

Unlike component parameters, domain parameters propagate to other components connected to the circuit. Therefore, when you set the parameter as `Run-time` in the source component, it is possible that another component connected to the same circuit is using this parameter in the context which prevents it from being run-time configurable. For example, if one of the components connected to the circuit uses a domain parameter in its `setup` function, you get an error when trying to simulate the model.

Compatibility Considerations

To avoid errors when using run-time domain parameters, it is recommended that you avoid using the `setup` function in your custom components. Other constructs available in Simscape language let you achieve the same results without compromising run-time capabilities.

Task	Recommended Technique
Validate parameters	Use an `assert` construct. For more information, see Programming Run-Time Errors and Warnings.
Compute derived parameters	Use declaration functions. For more information, see Declaration Functions.
Set initial conditions	Assign variable priority and target value. For more information, see Variable Priority for Model Initialization.
Designate source for domain parameters	Use direct assignment to a domain parameter in the component node declaration. For more information, see Source Components.

Parametric Assert Actions: Specify assert action based on parameter evaluation

The assert action determines whether triggering the assert results in a warning or an error during simulation. In previous releases, you used the optional `Warn` attribute to specify whether simulation errors out when the predicate condition is violated (Warn = false), or continues with a warning (Warn = true). This syntax requires you to write conditional statements if the assert action needs to be controlled parametrically, for example:

```
% Stepper motor action on slipping: warn, error, none
  if assert_if_slipping == 1
      assert(slipping<1,'Stepper motor slip',Warn=true)
  elseif assert_if_slipping == 2
      assert(slipping<1,'Stepper motor slip')
  else
      % No assertion
  end
```

The new `Action` attribute lets you specify the assert action based on an enumerated parameter value. A built-in enumeration `simscape.enum.assert.action` allows three possible actions when the assertion is triggered: `warn`, `error`, and `none`. You can provide an enumerated value directly to the `Action` attribute:

```
assert(u > 0, Action = simscape.enum.assert.action.warn)
```

or create an enumerated parameter and let the block user control the assert action:

```
parameters
  assert_action = simscape.enum.assert.action.warn % has 3 values - none, warn, error
end

equations
  assert(u > 0, Action = assert_action)
end
```

The stepper motor code above can now be rewritten as:

```
parameters
  assert_action = simscape.enum.assert.action.warn % Action on slipping
end

equations
  assert(slipping<1,'Stepper motor slip',Action = assert_action)
end
```

Compatibility Considerations

In previous releases, you specified the assert action by using the `Warn = true|false` attribute. This attribute still works. Internally, its values are automatically mapped to the appropriate values of the new `Action` attribute:

Old Syntax	New Syntax
`Warn = false`	`Action = simscape.enum.assert.action.error`
`Warn = true`	`Action = simscape.enum.assert.action.warn`

You cannot use the `Warn` and `Action` attributes together in a single `assert` construct. When authoring new components, use the `Action` attribute because it provides more flexibility.

Foundation Library and Simulation

PS Block Library with Unit Propagation: Propagate physical signal units throughout the model

All blocks in the Physical Signals library have been reimplemented with untyped inputs and outputs, to facilitate signal size and unit propagation. For more information, see “Physical Signal Propagation: Specify units for physical signals and propagate them through the model” on page 8-3. Most of the block names remain the same, but several block names have changed slightly, to improve consistency.

Old Name	New Name
Asynchronous Sample & Hold	PS Asynchronous Sample & Hold
Counter	PS Counter
Random Number	PS Random Number
Repeating Sequence	PS Repeating Sequence
Uniform Random Number	PS Uniform Random Number

A new block in the Physical Signals/Utilities library, PS Signal Specification, lets you explicitly specify the size and unit of a physical signal. Use this block when the signal size and unit cannot be determined implicitly, based on port connections in the model.

Compatibility Considerations

The new Physical Signals library consists of new blocks, with unit propagation. These blocks do not automatically replace the respective legacy blocks in your model. The former Physical Signals library is no longer part of the Foundation library, but the legacy blocks in your models continue to work as before. However, these blocks can be removed in a future release. To upgrade your blocks to the latest version, use the **Check and update outdated Simscape Physical Signal blocks** check in the Upgrade Advisor.

For better differentiation, the Model Advisor check **Check for outdated Simscape blocks**, introduced in R2013a, has now been renamed to **Check for outdated AC source blocks**. This check detects a pre-R2013a version of AC Voltage Source and AC Current Source blocks in your model.

Slider-Crank Block: Model piston engines and pumps

The new Slider-Crank block in the Mechanisms library represents the slider-crank mechanism as a converter between mechanical rotational and mechanical translational motions.

The mechanism has two connections:

- Port C corresponds to the crank and is a mechanical rotational conserving port.
- Port S corresponds to the slider and is a mechanical translational conserving port.

Volumetric Flow Rate Sources and Sensor for Gas Domain: Specify and measure volumetric flow rate of gas

Three new blocks are available for the gas domain:

- Volumetric Flow Rate Source (G) — Generate constant volumetric flow rate of gas.
- Controlled Volumetric Flow Rate Source (G) — Generate time-varying volumetric flow rate of gas.
- Volumetric Flow Rate Sensor (G) — Measure volumetric flow rate of gas.

See the block reference pages for details.

Partition Method Selection: Prioritize speed or robustness when using Partitioning local solver

The Solver Configuration block has a new parameter, **Partition method**, which lets you select between two options:

- `Robust simulation` — Increase simulation robustness. This is the default method for new models.
- `Fast simulation` — Improve simulation performance. This is the same method as in previous releases, and therefore it is the default for existing models.

Steady-State Initialization for Frequency-and-Time Compatible Models: Start a time simulation in steady state using frequency-time equations

For models compatible with frequency-and-time equation formulation, when you select the **Start simulation from steady state** check box in the Solver Configuration block dialog, the solver now attempts to perform sinusoidal steady-state initialization. In other words, initialization is performed using frequency-time equations, and then the simulation proceeds using the actual equation formulation and other options selected in the Solver Configuration block. For more information, see Frequency and Time Simulation Mode.

If the model is not frequency-and-time compatible, the initialization behavior with **Start simulation from steady state** is the same as in previous releases.

Compilation Status: Display model compilation and initialization progress in Simulink status bar

When you run a model simulation, the first phase of simulation invokes the model compiler. The model compiler converts the model to an executable form, a process called compilation. Compilation and initialization of a large Simscape model can take several minutes. The simulation status bar in the bottom-right corner of the model window now displays a series of messages that correspond to the various stages of physical network compilation and initialization, such as:

- Constructing equation systems for Simscape physical networks
- Analyzing equation systems for Simscape physical networks
- Setting up simulation for Simscape physical networks
- Initializing equation systems for Simscape physical networks

These messages help you monitor compilation progress.

Run-Time Capable Foundation Domains: Leverage run-time domain parameters to sweep more parameters faster

Run-time domain parameters, implemented in this release, allow several Foundation library blocks that set the domain parameters to have more parameters run-time capable. These include the Custom Hydraulic Fluid block, Gas Properties (G) block, Moist Air Properties (MA) block, and Two-Phase Fluid Properties (2P) block. For more information, see “Run-Time Domain Parameters: Modify domain parameter values without regenerating C code” on page 8-4.

Spectrum Analyzer improvements for exponential averaging, mixed-complexity inputs

The Spectrum Analyzer block now allows mixed-complexity inputs and exponential averaging.

Smooth data with exponential averaging

The Spectrum Analyzer block now has two averaging modes for smoothing input samples: running averages (existing) and exponential averages (new). To specify the smoothing options for your input data, use these properties:

- **Averaging method** (`AveragingMethod`) — Choose `Running` or `Exponential`.
- **Averages** (`SpectralAverages`) — For running averages, choose the number of spectrum estimates to include in the running average.
- **Forgetting factor** (`ForgettingFactor`) — For exponential averaging, weigh previous spectrum estimates with a forgetting factor in the range (0,1].

For more details about the averaging methods, see Spectrum Analyzer Algorithms.

Display block inputs with different complexity

In the Spectrum Analyzer block, you can now visualize frequencies of inputs with different complexities. The signals must have the same sample rate and frame size, but can have both real and complex values.

R2018b

Simscape Language

intermediates Section: Streamline code by declaring reusable equation terms

The new `intermediates` section in a component file lets you define named intermediate terms for use in equations. This functionality is similar to a `let` statement, except that a named intermediate term has an expanded scope. You can reuse it in any equations section within the same file or an enclosing composite component. When an intermediate is used in an equation, it is ultimately substituted with the expression that it refers to.

You can also include an `intermediates` section in a domain file and reuse these intermediate terms in any component that has nodes of that domain type.

You can include intermediate terms in simulation data logs by specifying the appropriate `ExternalAccess` attribute value. This is another advantage of using intermediate terms instead of `let` statements.

For more information, see Using Intermediate Terms in Equations.

subsystem2ssc Function: Convert a physical modeling subsystem to a Simscape file

The `subsystem2ssc` function lets you convert a subsystem consisting entirely of Simscape blocks into a single Simscape file. The function generates a composite component file based on the subsystem configuration. You can mark member block and subsystem parameters for promotion to the top level, and the function automatically generates the corresponding code, similar to composite components.

If the subsystem being converted contains nested subsystems, then the function generates several Simscape files, one for each subsystem.

For more information, see Converting Subsystems into Composite Components.

String Support: Simscape functions now accept MATLAB strings as input

You can now use MATLAB strings as input for Simscape functions. For information on MATLAB strings, see Create String Arrays (MATLAB).

Foundation Library and Simulation

HDL Code Generation from Simscape Models: Convert models to HDL code for simulation on FPGA devices

If you have an HDL Coder license, you can now deploy switched linear plant models developed using Simscape blocks to a target FPGA device.

To deploy the plant model to an FPGA device:

1. Create a Simscape model using switched linear blocks and configure it for HDL code generation compatibility. To find out whether blocks in your model are linear, switched linear, or nonlinear, use the `simscape.findNonlinearBlocks` function.
2. Run the Simscape to HDL code generation advisor to convert the Simscape design to an HDL native floating-point implementation model from which you can generate code.
3. Use the HDL Workflow Advisor to generate native floating-point HDL code from the implementation model.

Simscape Bus Block: Create buses from Simscape physical connection lines

The new Simscape Bus block in the Utilities library lets you bundle conserving connections into a Simscape Bus line. You can also use this block to access one or more connections from an existing Simscape Bus line.

daessc Solver for Simscape: Simulate models using a solver specifically developed for DAEs

A new variable-step Simulink solver, `daessc (DAE solver for Simscape)`, is designed specifically for physical modeling. The solver computes the model's state at the next time step by solving systems of differential-algebraic equations resulting from Simscape models. `daessc` provides robust algorithms specifically designed to simulate differential-algebraic equations arising from modeling physical systems.

The `daessc` solver is available with a Simscape license only. To select this solver, open the **Solver** pane of the Configuration Parameters dialog box, set **Type** to `Variable-step`, and then from the **Solver** drop-down list, select `daessc (DAE solver for Simscape)`.

Automatic Matrix Handling Option: Simulate your model more quickly using optimized linear algebra settings

The **Linear Algebra** parameter of the Solver Configuration block has a new option, `auto`, which is now the default. With this setting, the solver automatically selects the appropriate option, either sparse or full, for treating the matrices.

Supercritical Fluid Systems: Model fluids above the critical point using the Two-Phase Fluid domain and block library

The Two-Phase Fluid library and domain have been extended to allow modeling supercritical fluids:

- The `twoPhaseFluidTables` function can now return data in the supercritical region.
- Library blocks can handle pressure going above critical.
- Chambers, mechanical converters, pipe, and reservoir blocks have new options for specifying initial conditions. No need to specify the phase.
- New Vapor Quality Sensor (2P) block dedicated to measuring vapor quality. Thermodynamic Properties Sensor (2P) block now outputs entropy instead of vapor quality.

Compatibility Considerations

- The Pipe (2P) block heat transfer coefficient has been updated to use a different correlation in the two-phase mixture region. Therefore, your simulation results might be slightly different compared to previous releases.
- Existing models, if they contain a Thermodynamic Properties Sensor (2P) block with connections at port **x**, will now have the new Vapor Quality Sensor (2P) block automatically added to the model and these connections moved to port **x** of the Vapor Quality Sensor (2P) block.

Gas Properties Visualization: Plot the data for gas domain

You can plot Gas domain data specified using the Gas Properties (G) block in your model. Plotting the properties lets you visualize the data before simulating the model.

To plot the data, right-click the Gas Properties (G) block in your model and, from the context menu, select **Foundation Library > Plot Gas Properties**.

Frequency and Time Simulation Enhancements: Log amplitude, phase, and offset of frequency variables and specify variable initialization priority

Frequency-and-time simulation mode was introduced in R2018a (see “Frequency and Time Formulation: Increase simulation speed for systems with a single base frequency” on page 10-3). This functionality has been extended:

- Data logging now preserves the sinusoidal shape of frequency variables. You no longer have to use scopes to view the simulation results.
- The logged data for frequency variables now contains subnodes that let you examine the amplitude, phase, and offset data separately.
- Initialization priority and targets for frequency variables are no longer ignored.
- The **Representation** column in the Variable Viewer shows **Frequency** or **Time** designation for all variables, including the eliminated ones.

Partitioning Local Solver Enhancements: Optimize solver, view statistics, and log simulation data

Partitioning local solver was introduced in R2018a (see “Partitioning Local Solver: Increase real-time simulation speed” on page 10-3). This functionality has been extended:

- New Solver Configuration block options improve performance.
- Limitations on data logging, variable initialization, and model statistics have been removed.

Converter Block Icons: Convey signal conversion in minimal canvas space

Block icons for the Simulink-PS Converter and PS-Simulink Converter blocks have been updated. They are now smaller and unobtrusive to signify the signal type conversion using minimal space.

New examples

Examples introduced in this version are:

- Oxygen Concentrator
- Transcritical CO2 (R744) Refrigeration Cycle

R2018a

Simscape Language

tablelookup Usability Improvements: Use enumerations for interpolation and extrapolation options

The `tablelookup` function has several usability improvements in this release:

- You can now use built-in enumerations, included in Foundation library, for interpolation and extrapolation options. For more information, see Using Enumeration in Function Arguments. All Foundation library blocks now use these enumerations for table lookup.
- The `error` extrapolation option, previously implemented at the block level for some Foundation library blocks, is now available at the function level. If you choose `error` for `extrapolation`, the function generates an error when the query value is greater than the upper boundary or less than the lower boundary of the provided grid values.
- The function introduces no zero crossings.

These enhancements let you reduce the number of function calls from inside the block, increase code efficiency, and improve simulation performance.

Compatibility Considerations

The `tablelookup` function algorithm has changes, to ensure that the function introduces no zero crossings. Therefore, your simulation results might be slightly different compared to previous releases.

Improved Error Reporting: Report stack trace for assertions

Improved error handling now reports the correct assert location if an assert comes from an member component inside a composite component, or from a Simscape function that is called from another Simscape function.

Foundation Library and Simulation

Moist Air Domain and Block Library: Model HVAC and environmental control systems

The Foundation library now contains the Moist Air block library and two new domains: moist air and moist air source. The library contains chambers, reservoirs, local restrictions, energy converters, sources and sensors. It also contains the Moist Air Properties (MA) block, which controls the air mixture properties for the attached circuit and gives you several options for modeling trace gas properties. You increase and decrease levels of moisture and trace gas in the air mixture by using the blocks in the Moisture & Trace Gas Sources library.

Use the Moist Air library to perform the following tasks:

- Develop requirements of an HVAC system for an environment, such as a building, automobile, aircraft
- Ensure acceptable temperature, pressure, humidity, condensation within the environment
- Determine capacity of an HVAC system to match heating, cooling, and dehumidification requirements
- Analyze HVAC system performance, efficiency, cost
- Validate HVAC system model against test data
- Design and simulate HVAC components, tune component models to test rig data
- Simulate models including an HVAC system, environment model, and controller
- Design controllers for valves, fans, compressors to ensure safe and optimal operation
- Perform HIL testing

For more information, see Modeling Moist Air Systems, Modeling Moisture and Trace Gas Levels, and the block reference pages. See also Moist Air Domain and Moist Air Source Domain for information on the domain definitions.

Frequency and Time Formulation: Increase simulation speed for systems with a single base frequency

Frequency and time simulation mode speeds up simulation of systems with a single nominal frequency by letting you increase the maximum step size for variable solvers. This mode also lets you perform phasor analysis of such systems by using the blocks in the Periodic Operators sublibrary of the Physical Signals library.

You can switch between time and frequency-and-time simulation modes, depending on your task, without modifying the model. For example, use the time simulation mode to study transient effects, and then switch to the frequency-and-time mode to perform the phasor analysis of a model.

For more information, see Frequency and Time Simulation Mode.

Partitioning Local Solver: Increase real-time simulation speed

A new local solver option lets you increase real-time simulation speed of certain Simscape models by partitioning the entire system of equations corresponding to a Simscape network into a cascade of smaller equation systems. Not all networks can be partitioned. However, when a system can be partitioned, then it is more efficient to solve several smaller equation systems than a very large one. Therefore, real-time simulation is approximately 5 times faster with the new local solver, compared to existing local solvers.

To switch to the new local solver, open the Solver Configuration block dialog box, select the **Use local solver** check box, and then set the **Solver type** parameter to `Partitioning`.

For more information, see Increase Simulation Speed Using the Partitioning Solver.

twoPhaseFluidTables Function with CoolProp Support: Generate fluid property tables from CoolProp software

In previous releases, the `twoPhaseFluidTables` function required the use of the National Institute of Standards and Technology (NIST) REFPROP software. CoolProp is open-source software that has similar capabilities. You can now use the `twoPhaseFluidTables` function with either CoolProp or REFPROP software.

Live-Stream Data Logging: Display logged data from Simscape network in Simulation Data Inspector while simulating

New model configuration option, **Record data in Simulation Data Inspector**, lets you stream the time-series data, while it is being logged, and view the data in Simulation Data Inspector during simulation. After the simulation, you can view the logged data either in Simulation Data Inspector or in the Simscape Results Explorer.

To stream simulation data:

1. Set up your model to log simulation data, either for the whole model or on a block-by-block basis.
2. Enable data streaming by selecting the **Record data in Simulation Data Inspector** check box on the **Simscape** pane of the Configuration Parameters dialog box.
3. Start simulating the model. As soon as the streamed data becomes available, the Simulation Data Inspector button in the model toolbar highlights.
4. Open the Simulation Data Inspector to view the data during simulation and to compare data for different simulation runs. For detailed information on how to configure and use the Simulation Data Inspector, see Inspect and Analyze Simulation Results.

If you select the **Record data in Simulation Data Inspector** check box, then selecting **Open viewer after simulation** results in the Simulation Data Inspector opening after simulation, instead of the Simscape Results Explorer. To view the data in the Simscape Results Explorer, you must open it by right-clicking a block or by using `sscexplore`.

The **Record logged workspace data in Simulation Data Inspector** option on the **Data Import/Export** pane of the Configuration Parameters dialog box works the same way as before, that is, it lets you use the Simulation Data Inspector to view the logged Simscape data after simulation. The advantage of the new **Record data in Simulation Data Inspector** check box on the **Simscape** pane is that it lets you view the data both during and after the simulation.

Compatibility Considerations

The `loggingMode` property of the `simscape.logging.Node` object has been replaced with two properties, `savable` and `exportable`. When you retrieve a `simlog` object from a previous release, these properties are set automatically:

- If `loggingMode` was `memory`, then `savable` is `1` and `exportable` is `0`.
- If `loggingMode` was `disk`, then `savable` is `0` and `exportable` is `1`.

If you select the **Record data in Simulation Data Inspector** check box on the **Simscape** pane, then the `simlog` object has both `savable` and `exportable` properties set to `0`. You must use the Simulation Data Inspector data management functions to save and retrieve this data. For more information, see Inspect and Analyze Simulation Results.

Improved Operating Point Indexing and Robustness: Modify in place and restore private variables

Initializing models from saved operating points was introduced in R2017b (see “Operating Point Management: Initialize models from saved operating points” on page 11-4). This functionality has been extended:

- `OperatingPoint` and `Target` objects now use the `subsref` and `subsasgn` methods. This enhancement lets you use in-place editing and tab completion for traversing the operating point hierarchy, for example:

  ```
  op('DC Motor/Inductor 1/i_L') = simscape.op.Target(14,'mA');
  ```
- Private variables are now included by default in the operating point data. These variables are not observable by definition, therefore you cannot see them in the operating point data, but their inclusion helps restore the simulation state of the model during initialization. Two new methods, `hasPrivateData` and `removePrivateData`, let you find out whether an operating point has hidden data and, if necessary, remove it.

Note When you log simulation data by recording it in the Simulation Data Inspector, the simulation log does not contain private Simscape language data. Therefore, if you extract an operating point from data logged using the Simulation Data Inspector, private data is not included.

Code Generation Improvements for Lookup Tables: Generate faster and smaller code

In this release, there have been several enhancements to the `tablelookup` function (see “tablelookup Usability Improvements: Use enumerations for interpolation and extrapolation options” on page 10-2). These enhancements reduce the number of function calls from inside the block, increase code efficiency, and improve simulation performance. As a result, code generated for models that use lookup tables has become smaller and faster.

New examples

Examples introduced in this version are:

- Operating Point RLC Transient Response

- Aircraft Environmental Control System
- Medical Ventilator with Lung Model
- Pneumatic Actuator with Humidity

Also, the Vehicle HVAC System example has been updated to use Moist Air library blocks.

R2017b

Simscape Language

Simscape Functions: Reuse expressions in equations and member declarations of multiple components

Simscape functions model a class of pure first-order mathematical functions with explicit input-output relationship. These functions explicitly map the inputs of numerical values into outputs of numerical values by using declarative expressions. When a component calls a Simscape function, numerical input values are passed to the function, which then evaluates these declarative expressions to compute the output values.

Each function must be in a separate Simscape file. The file name must match the function name. For example, function `foo` must be in a file called `foo.ssc`.

initialevent Operator: Initialize event variables

`initialevent` is a new Simscape language operator that lets you specify initial values of event variables at the start of simulation. It returns true, and triggers an event, only once during simulation, right after initialization of continuous variables is finished.

The `initialevent` keyword is valid only inside a `when` clause predicate. For more information, see `initialevent`.

Enhanced import Statement: Apply specified context to resolving functions and enumerations

The `import` statement syntax remains the same as in previous releases. However, the functionality has been extended to support Simscape functions, MATLAB functions, and enumerations. Enumerations can now be located inside packages.

Compatibility Considerations

Simscape import now uses a consistent precedence order when loading files and calling functions. Previously, when both a Simscape and a MATLAB file on the path had the same name, sometimes the Simscape file was preferred. Now, the precedence order is always the same as that of the `which` function.

Foundation Library and Simulation

Nominal Value Specification: Improve simulation robustness by providing scale of variables to the solver

Nominal values provide a way to specify the expected magnitude of a variable in a model, similar to specifying a transformer rating, or setting a range on a voltmeter. Using system scaling based on nominal values increases the simulation robustness. This functionality provides a new way to fine-tune scaling of individual variables in a model. It is especially helpful for initial conditions convergence and maintaining a minimum step size.

To enable system scaling by nominal values:

1. Open the Configuration Parameters dialog box.
2. On the **Simscape** pane, select the **Normalize using nominal values** check box.

Note The **Normalize using nominal values** configuration option is selected by default for new models, but disabled for the existing models, to preserve the simulation results.

If scaling by nominal values is enabled, then the model provides the scaling information to the solver based on the specified nominal values. The scaling of each variable is determined by its nominal value and physical units. Nominal values can come from different sources:

- **Block** — You can specify nominal value and unit as variable declaration attributes in a Simscape component file underlying the block. These attributes translate into block parameters `x_nominal_value` and `x_nominal_unit` (where *x* is the variable name). You can also override these values on individual blocks in the model by setting the corresponding block parameter `x_nominal_specify` to `'on'` and supplying different values for `x_nominal_value` and `x_nominal_unit`. These parameters are not visible in the block dialog box, but you can use either the Property Inspector or `set_param` and `get_param` functions to view and change their values. For more information, see Modify Nominal Values for a Block Variable.
- **Model** — In absence of a nominal value specified for the block, a variable uses the nominal value for the commensurate physical unit specified in the model table. All models have a default table of nominal values and units (factory default). To view, add, and edit the value-unit pairs for the model, click the **Specify nominal values** button next to the **Normalize using nominal values** check box. For more information, see Specify Nominal Value-Unit Pairs for a Model.
- **Derived** — If the model table of nominal values does not contain a row for a unit commensurate with the physical unit of a variable, then the nominal value for this variable is derived from fundamental dimensions. For example, if the variable's initial value is in `lbf`, and there is no entry in the table for force, but the table contains `{10,'lbm'}`, `{12,'ft'}`, and `{2,'min'}`, then the nominal value for that variable is `{10*12/2^2,'lbm*ft/min^2'}`.
- **Fixed** — Event variables, top-level model inputs, and Simscape Multibody variables cannot be scaled according to nominal values.

The Variable Viewer in advanced configuration shows the nominal value and unit for each variable, along with the source.

Compatibility Considerations

For compatibility reasons, scaling by nominal values is disabled for models created in previous releases, and their simulation results are the same. However, if you open a model created in or before R2007b and not saved in a later release, it gets a default configuration set, with nominal scaling enabled, and the simulation results might change. To preserve the old simulation results, open the Configuration Parameters dialog box, go to the **Simscape** pane, clear the **Normalize using nominal values** check box, and save the model.

Operating Point Management: Initialize models from saved operating points

A new set of functions lets you save sets of data necessary to initialize a model, manipulate this data, and then use it to initialize another model, or the same model before another simulation run. These sets of data contain a hierarchy of operating point targets, each target consisting of a variable value, unit, and initialization priority.

For example, this command creates an empty `OperatingPoint` object named `op`:

```
op = simscape.op.OperatingPoint;
```

This command creates a new operating point target `t`:

```
t = simscape.op.Target(1.5, 'V', 'High')
```

And this command adds target `t` to the operating point `op` by assigning this target to the variable named `v0`:

```
op = set(op, 'v0', t)
```

Instead of adding targets one-by-one, you can create an `OperatingPoint` object by extracting data from an existing model or from logged simulation data.

For more information, see Initialize Model Using Operating Point from Logged Simulation Data.

Lookup Table Visualization: Plot the data based on specified interpolation and extrapolation methods

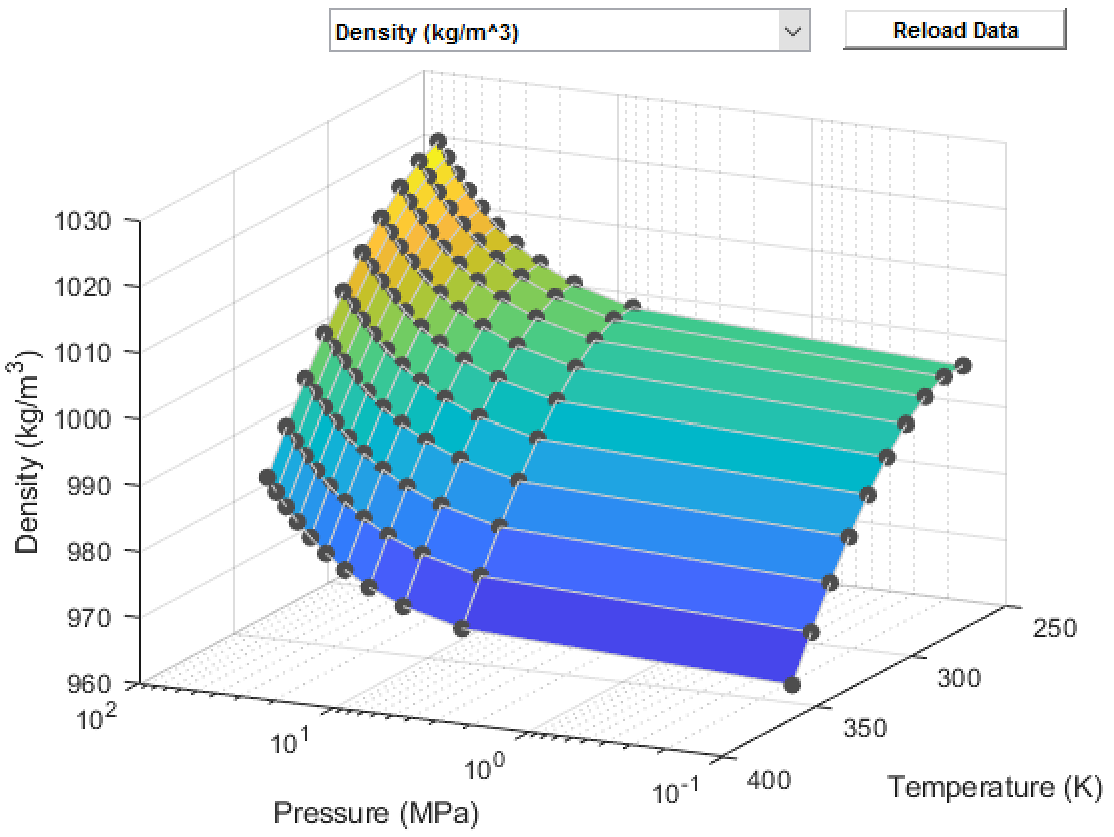
You can plot lookup table data specified for the PS Lookup Table (1D) and PS Lookup Table (2D) blocks in your model. Plotting the tables lets you visualize the data before simulating the model, to make sure that the table is correct. The plots reflect tabulated data specified for the block, as well as the selected interpolation and extrapolation options.

To plot the data, right-click the block in your model and, from the context menu, select **Foundation Library** > **Plot Table**. For more information, see Plot Lookup Tables.

Thermal Liquid Properties Visualization: Plot the data over pressure and temperature domains

You can now plot the properties of thermal liquids over their pressure and temperature domains. The plots are available through the context-sensitive menu of the Thermal Liquid Settings (TL) block. To generate them, right-click the block and select **Foundation Library** > **Plot Fluid Properties**.

Use the plots to visualize the dependence of the properties on pressure and temperature—for example, to catch anomalies in the specified data. You can view the plots one at a time. Use the drop-down list provided in the plot window to select the property to plot. Click the **Reload** button to regenerate the plots after you update the block parameters.



Thermal Liquid Properties Plot

Each plot depends on the block parameterization used. If you use a 2-D tabular parameterization, the properties are shown as functions of both pressure and temperature. If you use a 1-D tabular parameterization, the properties are shown as constant over the pressure axis—with the exception of density and any calculated properties that depend on it.

The dependence of density on pressure is important in a model and it is captured, in the 1-D tabular parameterization, by the **Constant isothermal bulk modulus** parameter. If the block is configured to not accept either the specific internal energy or the specific heat as an input, that quantity is calculated from an analytical expression that depends partly on density. Its plot then shows a dependence on pressure even if it is ultimately derived from 1-D tabular data.

Thermal Liquid Properties Specification: Provide partial data for selected properties

The Thermal Liquid Settings (TL) block adds simpler parameterizations for the various fluid properties. You can specify most properties as 2-D tables over pressure and temperature domains or as 1-D tables over the temperature domain alone. Properties such as density have an analytical option that requires only scalar coefficients and parameters at a known temperature and pressure.

The properties that you must specify depend on the parameterization that you select. For example, specific heat is no longer required if the **Table dimensions** parameter is set to `1D vectors based on temperature (T)` and the **Internal energy parameterization** parameter is set to `Specific internal energy vector - u(T)`. In this case, the specific heat is calculated from the specific internal energy that you specify.

Multiport Constant Volume Chambers: Model fluid tanks with multiple inlets and outlets

Constant volume chamber blocks with two and three conserving ports of the appropriate type have been added to the Gas, Thermal Liquid, and Two-Phase Fluid libraries. These blocks model mass and energy storage, similarly to the existing single-port constant volume chamber blocks in respective domains. Use the multiport constant volume chamber blocks to model fluid tanks with multiple inlets and outlets, or to add a fluid volume to a multiway connection in the network, to improve simulation robustness.

Fluid Flow Resistances: Model generic pressure loss in thermal liquid, gas, or two-phase fluid domains

Flow resistance blocks have been added to the Gas, Thermal Liquid, and Two-Phase Fluid libraries to provide a simple way to model pressure losses in the respective domain.

Other blocks that let you model pressure losses are pipes and local restrictions. However, these specialized blocks model other effects as well. Not all pressure loss processes fit into a pipe or a local restriction model.

The new flow resistance blocks model pressure loss in generalized, abstract terms, similarly to a resistor in electrical networks. You can use these blocks, for example, when you have pressure drop and flow rate information but do not have geometry information needed to use the local restriction block.

Thermal Resistances: Model heat transfer in generalized terms

Two new blocks in the Thermal Elements library let you model a heat transfer process in generalized terms, independently of whether it is by conduction, convection, radiation, or a combination thereof:

- Thermal Resistance — This block lets you model heat transfer in generalized terms by specifying the thermal resistance value as a block parameter. Thermal resistance is an abstract quantity that relates the thermal conductivity, the heat transfer coefficient, and the radiation coefficient.
- Variable Thermal Resistance — This block lets you model a variable heat transfer process in generalized terms. Thermal resistance of this block can vary with time, according to the input physical signal at port R.

Heat Flow Rate and Temperature Sources: Provide constant heat flow rate or temperature in thermal networks

Thermal source blocks generate thermal energy, characterized either by heat flow rate or by temperature. Existing blocks have an input physical signal port, and the magnitude of the heat flow rate or temperature controlling the block can vary with time. These blocks are not optimal for modeling constant sources. To set a constant heat flow rate or temperature at the source outlet, you would have to use a PS Constant block at the control signal port and connect a Thermal Reference block to the source inlet.

Two new blocks in the Thermal Sources library make it easier to model constant sources of thermal energy:

- Heat Flow Rate Source — An ideal energy source in a thermal network that maintains a constant heat flow rate, specified by the block parameter.
- Temperature Source — An ideal energy source in a thermal network that maintains a constant temperature, specified by the block parameter.

The existing Ideal Heat Flow Source and Ideal Temperature Source blocks have been renamed to Controlled Heat Flow Rate Source and Controlled Temperature Source, respectively. For consistency, the existing Ideal Heat Flow Sensor and Ideal Temperature Sensor blocks in the Thermal Sensors library have also been renamed, to Heat Flow Rate Sensor and Temperature Sensor, respectively. There is no compatibility impact because of these name changes.

Compatibility Considerations

The underlying component files for the Thermal Sources blocks have also been renamed, to avoid confusion.

Old Block Name	Old File Name	New File Name	New Block Name
Ideal Heat Flow Source	`heat_flow.ssc`	`controlled_heat_flow.ssc`	Controlled Heat Flow Rate Source
- (new block for R2017b)	-	`heat_flow.ssc`	Heat Flow Rate Source
Ideal Temperature Source	`temperature.ssc`	`controlled_temperature.ssc`	Controlled Temperature Source
- (new block for R2017b)	-	`temperature.ssc`	Temperature Source

If you used these files in your custom components, update the file names accordingly.

Two-Phase Fluid Sources That Perform No Thermodynamic Work: Configure flow conditions without affecting temperature

All blocks in the Two-Phase Fluid/Sources library now have a new **Power added** parameter that lets you select whether the source performs work on the fluid flow:

- `Isentropic power` — The source performs isentropic work on the fluid to maintain the specified pressure differential, mass flow rate, or volumetric flow rate depending on the source type. This is the default option, which is equivalent to the block behavior in the previous release. Use this option to represent an idealized pump or compressor and properly account for the energy input and output, especially in closed-loop systems.
- `None` — The source performs no work on the flow, neither adding nor removing power, regardless of the pressure differential or flow rate produced by the source. Use this option to set up the desired flow condition upstream of the system, without affecting the temperature of the flow.

Rotational Hard Stop and Translational Hard Stop Blocks with Configurable Force Law: Select numerically smooth options for faster simulation

Rotational Hard Stop and Translational Hard Stop blocks now have an additional parameter, **Hard stop model**, that lets you choose between three hard stop models:

- `Stiffness and damping applied smoothly through transition region, damped rebound` — This is the default option for new models. It has an additional parameter, **Transition region**. While the slider moves through the transition region, in which the force is scaled from zero. At the end of the transition region, the full stiffness and damping are applied. All equations are smooth and produce no zero crossings, resulting in faster and more robust simulation.
- `Full stiffness and damping applied at bounds, undamped rebound` — This model has full stiffness and damping applied with impact at upper and lower bounds, with no damping on the rebound. Equations produce no zero crossings when velocity changes sign, but there is a position-based zero crossing at the bounds. Having no damping on rebound helps to push the slider past this position quickly. This model has nonlinear equations.
- `Full stiffness and damping applied at bounds, damped rebound` — This hard stop model is the same as in previous releases. It has full stiffness and damping applied with impact at upper and lower bounds, with damping applied on the rebound as well. Equations are switched linear, but produce position-based zero crossings. Use this hard stop model if `simscape.findNonlinearBlocks` indicates that this is the block that prevents the whole network from being switched linear.

To preserve simulation results, legacy models have the **Hard stop model** parameter set to `Full stiffness and damping applied at bounds, damped rebound`, which is equivalent to the hard stop model used in previous releases.

Thermal Unit Conversion in Data Logging: Simscape Results Explorer now displays data according to unit conversion attribute

Simscape Results Explorer now supports affine unit conversion. Objects of the `simscape.logging.Series` class have a new property, `conversion`, which reflects the thermal unit conversion type (absolute or relative). For more information, see Thermal Unit Conversions and Member Attributes.

If a variable is declared with the `(Conversion=relative)` attribute, then Simscape plots the data correctly by applying affine conversion. If you select a node representing a variable, the status panel in the bottom-left corner displays the unit conversion attribute value for that variable.

Increased robustness for the ode15s solver

The `ode15s` solver now has three performance levels (`0`, `1`, and `2`), to help you find the right balance between the simulation speed and robustness. The default level is `2`, which provides the maximum robustness but can slow down the simulation.

Compatibility Considerations

Existing models can exhibit slower simulation times because of the increased solver robustness. To revert to the way the `ode15s` solver operated in previous releases, you can set its performance level to `0` by using this command:

```
slfeature('RobustOde15s',0)
```

ssc_new now uses model templates

Calling the `ssc_new` function to create a new model is now equivalent to opening the corresponding Simscape template from the Simulink Start Page. For example,

```
ssc_new('DCMotor','electrical')
```

creates a new model called `DCMotor` by using the **Electrical** template.

Typing `ssc_new` creates a new untitled model using the **Generic Simscape** template.

`ssc_new` no longer opens the main **Simscape** library when creating a new model. Each template contains special blocks that you can double-click to open the respective block libraries.

Compatibility Considerations

- `ssc_new` now always uses `VariableStepAuto` as the default solver. You can no longer specify another solver when creating a model. Once the model is created, use the Configuration Parameters dialog or `set_param` to change the solver, if needed.
- When calling `ssc_new`, you can no longer use a cell array of domain types to add more than one type of reference block to the new model.

Simscape now localized into Japanese

Simscape software has been localized into Japanese. If you work on a machine with Japanese locale, user interface and messages are now translated. Also, Simscape block names can now contain cross-locale non-ASCII characters.

New examples

Examples introduced in this version are:

- Vehicle HVAC System
- Simscape Functions

R2017a

Simscape Language

Mode Charts: Define distinct modes for component behavior

Mode charts provide an intuitive way to model components characterized by a discrete set of distinct operating modes. A car clutch is a good example of such a component. It has several operating modes, with each mode being defined by a different set of equations. It also has a transition logic, with a set of predicate conditions defining when the clutch transitions from one mode to another. It is possible to model this component using primitive constructs, such as event variables and `edge` operators, but this way of modeling lacks readability. For more complex components, the file becomes cumbersome and unwieldy. Every time you model a component with multiple operating modes and transitions, this component is a good candidate for a mode chart implementation.

For more information, see Mode Chart Modeling.

Enumerations: Specify a discrete set of acceptable parameter values

Simscape language now supports MATLAB enumerations in:

- Component parameters
- Event variables and `when` clause predicates
- Equation predicates
- Conditional declaration predicates
- Function arguments (such as an interpolation method in `tablelookup`)
- Mode charts

You define enumerations using a MATLAB enumeration class. For more information, see Enumerations.

Run-Time Parameter Support for Declaration Functions: Tune input values for MATLAB functions that declare block parameters without regenerating code

Member declarations for parameters and variables can now include calls to MATLAB functions that generate code. The ability to use declaration functions to compute derived parameter values or initialize variables was introduced in R2016b (see “Declaration Functions: Call MATLAB from member declarations” on page 13-2).

If a member declaration contains a declaration function, you can use a new member attribute, `MATLABEvaluation`:

- If you specify `MATLABEvaluation = compiletime`, the declaration function will be evaluated only at compile time, and all the function input parameters will be marked as compile-time only. Use this attribute if the MATLAB function does not support code generation, to prevent the block user from accidentally designating any of the function input parameters as `Run-time` in the block dialog.
- If you omit this attribute, or specify `MATLABEvaluation = default`, the declaration function will be evaluated at run time if a run-time parameter appears in its input parameters. Otherwise, it will be evaluated at compile time.

Block-Level Diagnostics for Numbers of Equations and Unknowns: Identify overdefined or underdefined components

System diagnostics, run automatically at compile time, compare the number of equations to the number of variables in the top-level model, as well as analyze the number of equations, variables, and nodes for each block.

The diagnostics also check that the number of Through and Across variables in a custom domain is the same. Although permitted by the language semantics, having a different number of Through and Across variables significantly restricts component connections.

For a domain with an equal number of Through and Across variables, well-defined components generally comply with the following equation:

$Nvar - Neq - Nports \cdot NThrVars = 0$,

where:

- *Nvar* is the number of variables.
- *Neq* is the number of equations.
- *Nports* is the number of nodes.
- *NThrVars* is the number of domain Through variables.

Error reporting includes:

- The name of the block or blocks that contribute to the problem with the top-level model.
- The number of equations and variables for each Simscape file where the counts are different. Separate listings for the number of equations that explicitly appear in the file and the number of equations added automatically by `branch` statements.
- Suggestions about possibly missing a branch statement in a component, or a reference node in a composite component.

Foundation Library and Simulation

simscape.findNonlinearBlocks Function: Identify components with nonlinear equations

The new `simscape.findNonlinearBlocks` function checks a model and reports which blocks, if any, contain nonlinear equations that keep the Simscape networks in the model from being linear or switched linear. The function takes the name of the model (in single quotes) as an argument and returns a cell array of the block names. If all the networks in the model are linear or switched linear, the cell array is empty.

On-Demand Update for Variable Viewer and Statistics Viewer: Streamline review of modeling statistics

In previous releases, every time you opened the Variable Viewer, it updated the diagram and ran the simulation for 0 seconds to calculate the actual initial values. Opening the Statistics Viewer also triggered the block diagram update. For complex models, a diagram update can last several minutes, and unnecessary diagram updates could lead to loss of productivity.

In this release, opening the tool does not trigger an automatic update. When you open the Variable Viewer or Statistics Viewer, it gets populated with the data from the last simulation or diagram update. You have to update the data explicitly by clicking the **Refresh** button (Update). If both tools are open, clicking **Refresh** in one of them updates both.

The status at the bottom of each tool displays the timestamp of its last update. If you have modified the model since the viewer has last been updated, the **Refresh** button displays a warning symbol (yellow triangle), and the timestamp at the bottom of the viewer window turns red to indicate that the data in the viewer might not reflect the latest model changes.

If you open a model, and then open the Variable Viewer or Statistics Viewer before simulating the model, then the viewer does not contain any data. The **Refresh** button displays a warning symbol (yellow triangle), and a message at the top of the viewer window tells you to click the **Refresh** button to populate the viewer with data.

Gas Pressure and Flow Rate Sources That Perform No Thermodynamic Work: Configure flow conditions without affecting temperature

All blocks in the Gas/Sources library now have a new **Power added** parameter that lets you select whether the source performs work on the gas flow:

- `Isentropic power` — The source performs isentropic work on the gas to maintain the specified pressure differential or mass flow rate, depending on the source type. This is the default option, which is equivalent to the block behavior in the previous release. Use this option to represent an idealized pump or compressor and properly account for the energy input and output, especially in closed-loop systems.
- `None` — The source performs no work on the flow, neither adding nor removing power, regardless of the pressure differential or mass flow rate produced by the source. Use this option to set up the desired flow condition upstream of the system, without affecting the temperature of the flow.

Thermodynamic Properties Sensor Blocks: Measure thermodynamic quantities in gas and thermal liquid domains

Similar to the existing Thermodynamic Properties Sensor (2P) block in the Two-Phase library, thermodynamic properties sensor blocks have been added to the Gas and Thermal Liquid libraries:

- Thermodynamic Properties Sensor (G) — Represents an ideal sensor that measures specific enthalpy, density, specific heat at constant pressure, and specific entropy in a gas network.
- Thermodynamic Properties Sensor (TL) — Represents an ideal sensor that measures specific internal energy, density, and specific heat at constant pressure in a thermal liquid network.

See the block reference pages for details.

Laminar-Turbulent Regime Definition by Pressure Ratio: Improve simulation robustness with smoother regime transitions

For improved simulation robustness, especially in the presence of dry nodes, several hydraulic blocks now contain a new **Laminar transition specification** parameter, with these options:

- `Reynolds number` — The transition from laminar to turbulent regime is assumed to take place when the Reynolds number reaches the value specified by the **Critical Reynolds number** parameter. This is the way these blocks worked before, and all the equations are the same as in the previous releases.
- `Pressure ratio` — The transition from laminar to turbulent regime is defined by the following equations:

  $$p_{cr} = (p_{avg} + p_{atm})(1 - B_{lam})$$

  $$p_{avg} = (p_A + p_B)/2$$

  where:

  - p_{cr} is the minimum pressure for turbulent flow.
  - p_A and p_B are gauge pressures at ports A and B, respectively.
  - p_{atm} is the atmospheric pressure, 101325 Pa.
  - B_{lam} is the value of the new block parameter, **Laminar flow pressure ratio**. The default value of this parameter is `0.999`.

The change affects the Constant Area Hydraulic Orifice and Variable Area Hydraulic Orifice blocks in the Foundation library, as well as several Simscape Fluids isothermal blocks.

New blocks use `Pressure ratio` as the default **Laminar transition specification** option. However, when you open an existing model, the **Laminar transition specification** parameter value is `Reynolds number` and the simulation results are the same as in previous releases.

Bulk Modulus Logging for Hydraulic Chamber Block: Record bulk modulus during simulation

Bulk modulus variable has been added to the logged simulation data for the Constant Volume Hydraulic Chamber block. To log and plot data for fluid bulk modulus:

1 Enable simulation data logging.
2 Run the simulation.
3 Open the Simscape Results Explorer.
4 In the left pane, expand the node representing a Constant Volume Hydraulic Chamber block and select the `bulk_inst` node. The right pane displays the plot of the instantaneous bulk modulus during simulation.

Rotational Friction and Translational Friction Blocks: Improve simulation speed and robustness

Rotational Friction and Translational Friction blocks now have a new **Breakaway friction velocity** parameter, which replaces the **Transition approximation coefficient** and **Linear region velocity threshold** parameters. The block equations are now smoother and introduce no zero crossings, which results in faster and more robust simulation.

Compatibility Considerations

Simulation results for existing models using the Rotational Friction and Translational Friction blocks can change slightly.

Input Derivative Setting for Piecewise Constant Signals: Smoother handling of discrete inputs to Simscape network

Simscape solver may require that you provide time derivatives of some of the input signals, especially if you use an explicit solver. You can control the way you provide time derivatives for each input signal by configuring the Simulink-PS Converter block connected to that input signal. One way of providing the necessary input derivatives is by filtering the input through a low-pass filter. You can also provide first and second derivatives separately, through additional ports on the Simulink-PS Converter block.

In previous releases, if you selected the `Use input as is` option and the solver required derivatives:

- For continuous signals, the solver issued an error.
- For discrete signals (that is, for signals with sample time other than continuous), the solver explicitly set the input derivatives to zero. This behavior could lead to unexpected results for sampled continuous (piecewise constant) signals.

To make the solver behavior more transparent, the options in the **Input Handling** section of the Simulink-PS Converter block dialog have changed. First, you make the decision whether to filter input or provide the derivatives separately, by using the **Filtering and derivatives** parameter:

- If you set this parameter to `Filter input, derivatives calculated`, the input filtering functionality works the same as before. Select the first-order or second-order filter, by using the **Input filtering order** parameter, and set the appropriate **Input filtering time constant (in seconds)** parameter value for your model.
- If you do not want to filter the input, keep the **Filtering and derivatives** parameter as `Provide signals`. Then, select the **Provided signals** parameter value: `Input only`, `Input and first derivative`, or `Input and first two derivatives`. Additional Simulink input ports appear

on the Simulink-PS Converter block, as needed, to let you connect the signals providing input derivatives.

- If your signal is piecewise constant, you can explicitly set the input derivatives to zero by selecting the `Zero derivatives (piecewise constant)` value for the **Filtering and derivatives** parameter.

Note When you add a new Simulink-PS Converter block to your model, the default input handling options are `Provide signals` and `Input only`. However, in legacy models, the Simulink-PS Converter blocks that had the `Use input as is` option now have the **Filtering and derivatives** parameter set to `Zero derivatives (piecewise constant)`, to avoid compatibility issues and preserve the old simulation results.

---

tanh Function in PS Math Function Block: Calculate hyperbolic tangent of a Simscape physical signal

The hyperbolic tangent function `tanh(u)` is now one of the choices available in the PS Math Function block. This function is useful for providing smooth transitions around zero. See the block reference page for details.

Improved Spectrum Analyzer: Analyze signals in the frequency domain using additional units and harmonics

Signal analysis with the Spectrum Analyzer block has been improved:

- Visualize your signal spectrum as the root-mean squares (RMS) using **Type** > **RMS**. When you select RMS as your spectrum type, you can choose from two spectrum units (previously called power units), **Vrms** and **dBV**.
- On the Distortion Measurement panel, measure up to 99 harmonics using the **Num. Harmonics** option.

Increased robustness for the ode23t solver

The `ode23t` solver now has three performance levels (`0`, `1`, and `2`), to help you find the right balance between the simulation speed and robustness. The default level is `2`, which provides the maximum robustness but can slow down the simulation.

Compatibility Considerations

Existing models can exhibit slower simulation times because of the increased solver robustness. To revert to the way the `ode23t` solver operated in previous releases, you can set its performance level to `0` by using this command:

```
slfeature('RobustOde23t',0)
```

Two-Phase Fluid block updates

The Two-Phase Fluid blocks have been updated for accuracy, versatility of use, and numerical robustness during flow reversals. Block changes include the following:

- The Two-Phase Fluid Properties (2P) block contains two new parameters—**Atmospheric pressure** and **Dynamic pressure threshold for flow reversal**. Use the **Atmospheric pressure** parameter to change the pressure of the external environment in the attached two-phase fluid network. Use the **Dynamic pressure threshold for flow reversal** parameter to control the smoothing applied to reversals in flow direction. For more information on the nature of the smoothing, see the parameter description in the block reference page.
- The Reservoir (2P) block contains a new parameter named **Reservoir pressure specification**. The Rotational Mechanical Converter (2P) and Translational Mechanical Converter (2P) blocks each contain a similar parameter named **Environment pressure specification**. Use the new parameter to set the component environment pressure to the **Atmospheric pressure** specified in the Two-Phase Fluid Properties (2P) block or to a different value. The environment pressures are by default set to the global **Atmospheric pressure** parameter.
- The inlet area in the Translational Mechanical Converter (2P) block is no longer assumed to be the same as the moving interface area. To reflect this change, the block now includes two area parameters—one for the interface, named **Interface cross-sectional area**, and one for the inlet, named **Cross-sectional area at port A**. The two parameters are by default identical in value.
- Source blocks no longer assume the port areas to be identical. To reflect this change, the **Cross-sectional area at ports A and B** parameter has been split into two new parameters—**Cross-sectional area at port A** and **Cross-sectional area at port B**. You can use the updated blocks to model more realistic compressors in which the outlet is smaller than the inlet.
- Flow regime transitions in the Local Restriction (2P) and Variable Local Restriction (2P) blocks are now based on the ratio of the outlet to the inlet pressure. To reflect this change, a new parameter named **Laminar flow pressure ratio** replaces the previous **Critical Reynolds number** parameter. The new parameter provides smoother and more robust flow rate transitions at near-zero mass flow rates. There is no direct mapping between the old and new parameters.
- Two sensor blocks have been renamed to more accurately reflect the quantities measured. The Mass & Energy Flow Sensor (2P) block is now the Mass & Energy Flow Rate Sensor (2P) block. Similarly, the Volumetric Flow Sensor (2P) block is now Volumetric Flow Rate Sensor (2P) block. The block behavior remains the same.
- Block equations have been updated to provide more robust simulation performance during flow reversals. These updates affect all Sources blocks as well as the Pipe (2P), Local Restriction (2P), and Variable Local Restriction (2P) blocks. For more information on the updated equations, see the block reference pages.

New examples

Examples introduced in this version are:

- Brayton Cycle (Gas Turbine)
- Building Ventilation
- Rankine Cycle (Steam Turbine)

Functionality being removed or changed

Feature Name	What Happens When You Use the Keyword?	Use This Instead	Compatibility Considerations
`ssc_reserved`	Does not run	Introduction of the new compiler in R2016a made this function obsolete. You still cannot use any of the Simscape language keywords as model or member names. This function used to list additional words that you also could not use. It returned an empty list in R2016b and has now been removed. If you use a word that is not allowed as a model or member name (such as a Simscape language keyword), you get a compile-time error.	None

R2016b

Simscape Language

Conditional Sections: Define variants within component files

A conditional section is a top-level section guarded by an `if` clause. Conditional sections are parallel to other top-level sections of a component file, such as declaration or equations sections.

A conditional section starts with an `if` keyword and ends with an `end` keyword. It can have optional `elseif` and `else` branches. The body of each branch of a conditional section can contain declaration blocks, equations, structure sections, and so on, but cannot contain the `setup` function. Nested conditional sections are allowed.

The `if` and `elseif` branches start with a predicate expression. The predicates must be parametric expressions, because the structure of a model must be fixed at compile time and cannot change once the model is compiled. If a predicate is true, the branch gets activated. When all predicates are false, the `else` branch (if present) gets activated. Elements (such as declarations, equations, and so on) from active branches only are included in the compiled model.

The following restrictions apply:

- Predicates may depend on parameters of the parent (enclosing) component. They may not depend, directly or indirectly, on parameters of member (embedded) components or on domain parameters.
- Accessibility of class members declared inside conditional sections is equivalent to private class members. They are not accessible from outside the component class, even if their branch is active.
- The scope of the class members declared inside a conditional section is the entire component class. However, using a conditional member outside the conditional section when the section is not active results in a compilation error.

For more information, see Defining Component Variants.

Declaration Functions: Call MATLAB from member declarations

You can now use declaration functions to compute derived parameter values or initialize variables, instead of doing this inside the `setup` function.

Declaration function is a MATLAB function used inside a member declaration section in a Simscape file. A declaration function can be any MATLAB function (even if it is not supported in the Simscape language `equations` section), including user-defined functions on the MATLAB path. For example:

```
component A
  parameters
    p1 = 1;
    p2 = 0;
  end
  parameters(Access=private)
    pDerived = gamma(p1) + p2;
  end
  variables(Access=private)
    vDerived = {value = {my_fcn(p1,p2) + 1, 'm'}, priority = priority.high };
  end
  equations
    ...
  end
end
```

Use the `Access=private` attribute for member declaration unless all the arguments of the declaration function are constants.

Exercise caution when using persistent variables inside a declaration function, because this may lead to inconsistent results for multiple simulation runs.

Declaration functions can return multiple values. They follow the general MATLAB function conventions for multiple return values. For example, if `my_fcn()` is a declaration function that returns three values:

```
[id1, ~, id3] = my_fcn();  % omit the second return value
```

```
[id1] = my_fcn();  % rules of single assignment apply, nonrequested return values ignored
```

The following restrictions apply:

- You can use multiple value assignments on the left-hand side only for parameters and variables with the `Access=private` attribute.
- When omitting return values using the placeholder attribute (~), at least one value must be assigned. Empty declarations produce an error in Simscape language.

Implicit Reference Connection Syntax: Simplify connections to reference node

The * symbol, previously used only in `branch` statements to indicate connection to a reference node, can now also serve the same purpose in `connect` statements.

Previously, to connect a node to an absolute reference (ground), you needed to write the corresponding `branch` statements and the grounding equations, setting the Across variables equal to zero.

To ground a node in a composite component, you had to add a hidden grounding component (domain reference block) as a member and explicitly connect the node to this grounding component.

Now, you can indicate connections to an implicit reference node within the structure section:

```
connections
    connect(A, *);
end
```

The * symbol is not domain-specific, and the same structure section can contain connections to implicit reference in different domains:

```
component abc
    nodes
        M = foundation.hydraulic.hydraulic;
        N = foundation.electrical.electrical;
    end
    connections
        connect(M,*);
        connect(N,*);
    end
end
```

However, multiple ports connected to an implicit reference within the same `connect` statement must all belong to the same domain:

```
connections
    connect(a, b, *);
end
```

The order of ports does not matter. This behavior is consistent with general connection rules for multiple conserving ports. For more information, see Connections to Implicit Reference Node.

Compatibility Considerations

All the domain reference blocks in Foundation library (such as Electrical Reference, Mechanical Rotational Reference, and so on) have been updated to use the implicit reference syntax. Therefore, the Through variables for these blocks are no longer visible in logged simulation data. If your legacy models rely on using these variables in simulation data analysis, add a corresponding Through variable sensor to your model (such as the Current Sensor before the Electrical Reference block).

4D Table Lookup: Define lookup tables with four independent variables

The `tablelookup` function has been extended to support four-dimensional table lookup. That is, the function can now compute an output value by interpolating the input value against a set of data points in a four-dimensional table.

Additional MATLAB operators allowed in Simscape language

The following MATLAB operators are now allowed in a Simscape file:

Name	Notes
`repmat`	
`reshape`	Expanded empty dimension is not supported.
`dot`	Resulting unit is the product of the units of the first two arguments.
`cross`	Resulting unit is the product of the units of the first two arguments.
`diff`	Resulting unit is the same as the unit of the first argument. In the two argument overload, the upper bound on the second argument is 4, due to a Simscape limitation.

All arguments that specify size or dimension must be unitless constants or unitless compile-time parameters.

Two-Phase Fluid Base Classes: Energy calculations updated for simulation robustness

The base classes of the two-phase fluid domain have been updated to allow for smoother and more robust energy flow transitions during fluid flow reversals. This update affects which variables are declared, how they are computed, and whether they are logged during simulation:

- `one_port_horizontal.ssc` and `one_port_vertical.ssc` — The port-node specific internal energy (`u_A`), normalized specific internal energy (`unorm_A`), and specific volume (`v_A`) are no longer declared or logged. Derived components dependent on the eliminated variables must now explicitly declare those variables and provide their equations.
- `two_port_dynamic.ssc` — The port-node specific internal energies (`u_A`, `u_B`), normalized specific internal energies (`unorm_A`, `unorm_B`), and specific volumes (`v_A`, `v_B`) are now computed from adiabatic expressions relating the port and internal nodes. The updated properties no longer always match those carried by incoming flows and, to prevent confusion in interpreting simulation results, are no longer logged.
- `two_port_steady.ssc` — All variables are declared and logged as before. However, derived components must include an additional equation relating the fluid properties at port A to those at port B. For components that neither add nor subtract energy from the flow, this equation reduces to an equality statement between the specific total enthalpies,

  ```
  ht_A == ht_B
  ```

  where `ht_A` is the specific total enthalpy at port A and `ht_B` is that at port B. See the `+elements/local_restriction.ssc` file for an example of such a component. See the `+sources/source_base.ssc` file for an example of a component that adds or subtracts energy, in this case, via isentropic processes.

Compatibility Considerations

The updates to the Two-Phase Fluid base classes affect all custom components derived from them. To ensure that your custom components continue to work as expected, you must modify the component files as described for the different base classes. As a temporary workaround, you can point your custom components to the legacy base classes. These classes are still defined but now have the string `legacy_` prepended to their names:

- `legacy_two_port_dynamic.ssc`
- `legacy_two_port_steady.ssc`
- `legacy_one_port_horizontal.ssc`
- `legacy_one_port_vertical.ssc`

To access the port-node specific internal energies or specific volumes in custom components derived from the `two_port_dynamic.ssc` base class, you must now use the appropriate sensor blocks. Use the Pressure & Internal Energy Sensor (2P) block to measure the specific internal energy. Use the Thermodynamic Properties Sensor (2P) block to measure the specific volume. You can also use simulation data logging and Simscape Results Explorer to access the fluid properties at internal nodes.

Foundation Library and Simulation

Gas Domain and Block Library: Model gas systems with various levels of idealization

The Foundation library now contains a gas domain and the Gas block library. This library contains chambers, reservoirs, local restrictions, energy converters, sources and sensors. It also contains the Gas Properties (G) block, which controls gas properties for the attached circuit and lets you select the level of idealization: perfect gas, semiperfect gas, or real gas.

Use these blocks for applications such as pneumatic actuation of mechanical systems, natural gas transport through pipe networks, modeling gas turbines for power generation, air cooling of thermal components, and so on.

For more information, see Modeling Gas Systems and the block reference pages. See also Gas Domain for information on the gas domain definition.

Compatibility Considerations

The Gas block library replaces the Pneumatic library as the recommended way of modeling pneumatic systems. The former Pneumatic library is now included in the product installation as an example custom library. The pneumatic domain definition is still provided with the software, and all the pneumatic blocks in your legacy models continue to work as before. However, these blocks no longer receive full production support and can be removed in a future release.

It is recommended that, going forward, you use the Gas library for modeling pneumatic systems. Unlike the Pneumatic library blocks, the blocks in the Gas library do not have the requirement of adding fluid volumes to every node. Also, the blocks in the Gas library provide enhanced simulation robustness during flow reversal. For more information, see Flow Reversal.

Dry Node Detection: Check for dry nodes in hydraulic systems

A new Model Advisor check, **Check for dry hydraulic nodes**, lets you detect hydraulic nodes that are considered dry due to a lack of compliance. The check is not triggered by default when you run Model Advisor on your model. You can run this check by selecting it under **By Product | Simscape** or **By Task | Modeling Physical Systems**.

After you run the check, a table of results appears in the right pane of the Model Advisor window. The first column lists the dry nodes found, with the middle column listing the blocks connected to each dry node. Each cell in the middle column of the table contains a link to the block in question, and the corresponding cell in the third column contains the name of port that connects to the dry node.

Clicking a link highlights the corresponding block in the model.

Consider adding one Constant Volume Hydraulic Chamber block to each dry node in the list. The presence of dry hydraulic nodes can reduce the solver robustness in complex Simscape models. By adding a hydraulic chamber to a node, you can considerably improve the convergence and computational efficiency of a model. Adding a chamber adds a degree of freedom. By adding a chamber, you replace a complex algebraic constraint (the dry node) with a dynamic constraint. For more information, see Troubleshooting Hydraulic Models.

PS Lookup Table (4D) Block: Graphically define implicit equations that require lookup tables with four independent variables

The new PS Lookup Table (4D) block computes an approximation to some function `f=f(x1,x2,x3,x4)` given the `x1`, `x2`, `x3`, `x4`, `f` data points. The four inputs and the output are physical signals. See the block reference page for details.

Improved Simulation Logging Speed: Stream data to disk 5 to 30% faster

In this release, there are significant speed improvements for streaming logged data to disk. To stream data to disk, open the MATLAB Preferences dialog box, go to the **Simscape** pane, and select the **Stream data to temporary disk directory** check box. For more information, see Stream Logging Data to Disk.

Spectrum Analyzer Block: Display frequency spectrum of time-domain signals

The Spectrum Analyzer block is now available in the Simscape > Utilities library. This block contains a subset of functionality of the DSP System Toolbox™ block with the same name.

The Spectrum Analyzer block accepts input signals with discrete sample times and displays frequency spectra of these signals. It allows you to perform measurements and analyze the signals using a variety of methods. For more information, see the Spectrum Analyzer block reference page.

If you have both a Simscape license and a DSP System Toolbox license, then the Spectrum Analyzer block in the Simscape > Utilities library is identical to the block in the DSP System Toolbox > Sinks library. For more information, see Spectrum Analyzer in the DSP System Toolbox documentation.

Memristor block that lets you model resistance as function of current

The new Memristor block in the Electrical Elements library models an ideal memristor with nonlinear dopant drift approach. The behavior of memristor is similar to a resistor, except that its resistance (also called memristance) is a function of the current that has passed through the device. The memristance is defined by two states, A and B, with some fraction of the device in one of those states at a given time.

Hydraulic mass flow rate source and sensor blocks

Two new blocks in the Hydraulic > Hydraulic Sources library let you specify mass flow rate through the source (as opposed to volumetric flow rate, like the existing Hydraulic Flow Rate Source and Hydraulic Constant Flow Rate Source blocks):

- Hydraulic Constant Mass Flow Rate Source — Maintains a specified constant mass flow rate regardless of the pressure differential across the source. You specify the mass flow rate through the source by using a block parameter.
- Hydraulic Mass Flow Rate Source — Maintains a specified mass flow rate regardless of the pressure differential across the source. The mass flow rate through the source is directly proportional to the signal at the control port M.

The Hydraulic Flow Rate Sensor block has an additional output port M that lets you measure mass flow rate.

Variable Viewer configuration preferences

The new **Save Viewer Configuration** button in the Variable Viewer toolbar lets you save the following configuration preferences:

- Variable Viewer view type (tree or flat)
- Visible columns
- Ordering of columns
- Filters applied for all columns (both visible and hidden)
- Sorting on a specific column

If you save viewer configuration, then the next time you open Variable Viewer, for this or another model, it will open with the same configuration. This behavior is consistent with saving other MATLAB preferences.

Descriptive variable names in Simscape Results Explorer

Simscape Results Explorer now makes it easy to establish connection between the logged data and the corresponding block variable. When you select a variable (for example, `w`) in the Simscape Results Explorer tree, the status panel at the bottom of the window displays the descriptive name (for example, `Rotor angle`), which is consistent with the name displayed on the **Variables** tab of the block dialog. Clicking on this descriptive name of a variable opens the corresponding block dialog box.

If the descriptive name of a variable is too long to fit into the status panel, it is truncated with an ellipsis (…). If you hover over the truncated name, the tooltip for the status panel displays the entire descriptive name. Similarly, if you hover over a parameter name in the block dialog box, the tooltip displays the parameter description and id. These enhancements improve the product usability.

Improved scalability of generated code

Code generated from Simscape models now has better modularity and scalability. Therefore, it is more compiler-friendly, especially for Visual Studio® C compiler.

Highlighting available for Simscape connection lines

You can now highlight physical signal and conserving connections in Simscape block diagrams. Highlighting crosses subsystem boundaries, allowing you to trace a connection across multiple subsystem levels. To highlight connections, right-click a connection line and, from the context menu, select **Highlight Connections**.

To remove all highlighting, select **Remove Highlighting** from the model's context menu, or select **Display > Remove Highlighting**.

This functionality is similar to highlighting Simulink signals to source and destination. For more information, see Display Signal Sources and Destinations in the Simulink documentation.

Property Inspector available for Simscape blocks

Property Inspector view is now available for Foundation library blocks and other blocks authored using Simscape language (including blocks in Simscape add-on products). For more information on Property Inspector, see Property Inspector: Edit parameters and properties of model elements using a single interface in the *Simulink Release Notes*.

Two-Phase Fluid Blocks: Energy calculations updated for simulation robustness

The smoothed upwind scheme used in the energy flow calculations now allows for smoother transitions during reversals in flow direction. The smoother transitions are in part due to an updated smoothing factor that improves numerical robustness when energy variations are large and flow reversals frequent.

The **Characteristic longitudinal length** block parameter, used in the previous energy flow rate calculations, is now obsolete. This parameter has been removed from all Two-Phase Fluid blocks. The affected blocks include Local Restriction (2P) and Variable Local Restriction (2P) from the Elements library and all blocks from the Sources library.

The port-node fluid properties of the Pipe (2P) block are now computed from adiabatic expressions relating the port nodes to the internal nodes. These properties no longer always match those carried by incoming flows and, to prevent confusion in interpreting simulation results, are no longer logged. The affected fluid properties include:

- Specific internal energy at port A (`u_A` in the simulation data log) and port B (`u_B`)
- Normalized specific internal energy at port A (`unorm_A`) and port B (`unorm_B`)
- Specific volume at port A (`v_A`) and port B (`v_B`)

These fluid properties are also no longer logged for one-port blocks that make no use of them, including Reservoir (2P), Constant-Volume Chamber (2P), Translational Mechanical Converter (2P), and Rotational Mechanical Converter (2P).

Compatibility Considerations

To access the port-node specific internal energies or specific volumes, you must use the appropriate sensor blocks. Use the Pressure & Internal Energy Sensor (2P) block to measure the specific internal energy. Use the Thermodynamic Properties Sensor (2P) block to measure the specific volume. You can still use simulation data logging and Simscape Results Explorer to access the fluid properties at internal nodes.

Functionality being removed or changed

Feature Name	What Happens When You Use the Keyword?	Use This Instead	Compatibility Considerations
Pneumatic domain and block library	Still runs	Gas domain and block library	See Compatibility Considerations under "Gas Domain and Block Library: Model gas systems with various levels of idealization" on page 13-6.

R2016a

Simscape Language

Lookup Table Improvements: Define 3D tables, interpolate using Akima splines

The `tablelookup` function has been extended to support three-dimensional table lookup. That is, the function can now compute an output value by interpolating the input value against a set of data points in a three-dimensional table.

The interpolation and extrapolation options have been improved:

- There are two interpolation options, `linear` and `smooth`. The `smooth` option replaces the previously available `cubic` and `spline` options. It uses a modified Akima algorithm, as described in Smooth Interpolation Algorithm, to produce a continuous curve or surface with continuous first-order derivatives. It also tries to preserve the slope and avoid undulations where the data suggests a flat region. The default is `linear`, which provides the best performance.
- There are two extrapolation options, `linear` and `nearest`. The names of the options are the same as in previous releases, but the underlying algorithms have changed. The `linear` method now ensures a smooth connection between the interpolation region and the extrapolation region. The `nearest` method produces a curve or surface with continuous value at the boundary between the interpolation region and the extrapolation region that does not go above the highest point in the data or below the lowest point in the data.

Compatibility Considerations

The `smooth` interpolation option replaces the `cubic` and `spline` options, available in previous releases. If your existing models use `cubic` or `spline` interpolation, these options will be automatically mapped to `smooth`. The new method is superior in the way it preserves the shape of the data and produces a continuous curve or surface with continuous first-order derivatives. However, this enhancement can produce a change in results for your legacy models.

The new interpolation algorithm tries to preserve the slope and avoid undulations where the data suggests a flat region. It interprets the data as a flat region whenever there are three or more consecutive colinear points in the table data. It then connects those three or more points with a straight line. The colinear points do not have to be horizontal. To ensure that the region between two data points is flat, insert an additional data point between those two points.

The `linear` extrapolation method now ensures a smooth connection between the interpolation region and the extrapolation region. The `nearest` method extends as a constant that is now equal to the last interpolated value, rather than the last point in the output table data. These enhancements can also produce a change in results for your legacy models.

Event Variables and edge Operator: Capture values at events

Event is a conceptual notation that denotes a change of state in a system. Event modeling lets you perform discrete changes on continuous variables. The two most common applications of event modeling are:

- Trigger-and-hold mechanism, such as a triggered delay. For example, a component has two inputs: `u` and `x` (triggered signal), and one output `y`. When and only when the triggered signal `x` changes value from false to true, output `y` is reset to the value of `u` at current time. `y` remains unchanged all other times.

- Enabled component, acting on a principle similar to Simulink enabled subsystem. That is, the component has a control signal as input. If the control signal has a positive value, then the component holds certain states to the most recent value, or resets them. When the control signal is negative, the states change according to component equations.

The following new constructs in Simscape language let you perform event modeling: event variables, `edge` operator, `when` clause, and `events` section.

Event variables are piecewise constant, that is, they change values only at event instants, and keep their values constant between events. You can declare internal component variables of type integer or real as event variables by setting the `Event=true` attribute.

The `edge` operator takes a scalar boolean expression as input. It returns true, and triggers an event, when and only when the input argument changes value from false to true. The return type of `edge` is event type. Event type is a special category of boolean type, which returns true only instantaneously, and returns false otherwise.

The `when` clause serves to update the values of the event variables. The syntax is

```
when EventPredicate
  var1 = expr1;
  var2 = expr2;
  ...
end
```

`EventPredicate` is an expression that defines when an event occurs. It must be an expression of event type. The variables in the body of the `when` clause must be declared as event variables. When the event predicate returns true, all the variables in the body of the `when` clause simultaneously get updated to the new values. The `when` clause can have optional `elsewhen` branches. However, an `else` branch is not allowed.

The new `events` section in a component file manages the event updates. The `events` section can contain only `when` clauses. The order of `when` clauses does not matter.

The following example implements a triggered delay component:

```
component Triggered
  inputs
    u = 0; % input signal
    triggered = 0; % control signal
  end
  variables(Event=true)
    x = 0;
  end
  outputs
    y = 0;
  end
  equations
    y == x;
  end
  events
    when edge(triggered>0)
      x = u;
    end
    end
  end
```

For more information, see Discrete Event Modeling.

Integral Operator, integ: Specify time integration of an expression

The new `integ` operator lets you perform time integration of an expression in the `equations` section of a Simscape file without declaring and initializing extra variables.

The full syntax is:

```
integ(expr, t_L)
```

where:

- `expr` is a Simscape language expression.
- `t_L` is the lower integration limit, specified as a delay time relative to current time. This operand is optional.

The upper integration limit is the current simulation time. If you omit the lower limit, the integration starts from simulation start time.

Improved attributes for member visibility and access

The new attribute `ExternalAccess` controls visibility of members in the user interface, that is, outside Simscape language. In other words, this attribute controls whether parameters, variables, and other members are visible and modifiable in block dialog boxes, simulation logs, variable viewer, and so on. The attribute is applicable to all member classes and has the following values:

- `modify` — The member is modifiable in the block dialogs and visible in the logs and viewer.
- `observe` — The member is visible in the logs and viewer, but not modifiable.
- `none` — The member is visible nowhere outside the language.

The default value of the `ExternalAccess` attribute for a member depends on the value of the `Access` attribute for that member.

`Access`	**Default `ExternalAccess`**
`public`	`modify`
`protected`	`observe`
`private`	`observe`

Members in the base class with `Access=private` are forced to have `ExternalAccess=none`, to avoid potential collision of names between the base class and the derived class.

When `Access` is explicitly set to `private` or `protected`, it does not make sense to explicitly set `ExternalAccess=modify` . In this situation, the compiler issues a warning and remaps `ExternalAccess` to `observe`.

Compatibility Considerations

The `ExternalAccess` attribute replaces the member attribute `Hidden` (which you could previously set for all member classes, such as parameters, variables, and so on, including member components of a composite component). The model attribute `Hidden` (which you can set for components to control whether they show up in a generated library or report) is not affected.

You do not currently need to update the existing component files. Members with a `Hidden` attribute compile without warnings. In a future release, implicit specification of the `Hidden` attribute on a member will start to produce warnings when you attempt to build the component. Eventually, the `Hidden` attribute for members will be removed. When writing new component files, use the new syntax.

If you want to update your existing component files, replace `Hidden=true` with `ExternalAccess=observe`.

Simscape language improved ease of use

In an effort to lift restrictions, simplify the syntax, and extend the language capabilities, the following enhancements have been implemented in this release:

- Arbitrary order of sections in a Simscape file. There may still be no more than one setup section per component. Otherwise, a component file can now contain multiple instances of declaration blocks or implementation sections of the same type, such as branches, equations, and so on. The sections can appear in any order.
- `der` operator now supports higher order derivatives. For example, `der(der(x))` is the second order time derivative of x.
- All parameters are now variable-sized. In previous releases, only parameters passed directly to `tablelookup` function could have variable size. Now, you no longer need to specify the `Size=variable` attribute for table data parameters. You can also do arithmetic on table data parameters:

  ```
  component MyLookup
    parameters
      ud = [ 0 1 2 ];
      yd = [ 0 1 2 ];
    end
    inputs
      u = 0;
    end
    outputs
      y = 0;
    end
    equations
      y == tablelookup( ud, yd+1, u );
    end
  end
  ```
- Parameters may now be N-dimensional. This functionality is primarily intended for the three-dimensional lookup tables, introduced in this release (see “Lookup Table Improvements: Define 3D tables, interpolate using Akima splines” on page 14-2).
- Derived private parameters and variables. You can derive a parameter value by specifying an expression containing other parameters. For example:

```
component HasDerivedParameter
  parameters
    Par1 = 0;
    Par2 = 0;
  end
  parameters(Access=private)
    DerivedParameter = Par1 + Par2;
  end
  equations
    % Use DerivedParameter here.
  end
end
```

If `Par1` or `Par2` is modified in the setup section, then `DerivedParameter` will be the new sum of the two (and that change is applied at the moment of the setup assignment). The deriving relationship is broken only if the derived parameter is modified in the setup section.

The derived parameter must be declared with the `Access=private` attribute. The expression used to derive the parameter cannot be time-dependent, that is, it cannot contain operators such as `time`, `der`, `delay`, or `integ`. You can use the same method to initialize a variable.

- Nested parameter and variable access in composite components. For example, if component C contains component B, which in turn contains component A, both C and B can assign values to parameters and variables of component A. In case of conflicting assignment, the “outermost” one (in this example, component C) takes precedence.
- Use of member component parameters and variables in the composite component equations. Similar to the previous example, you can now use parameters and variables of component A in the equations of components B and C. For example:

```
component B
  parameters
    p1 = 0;
  end
  components
    a = A(v1 = 10); % Sets initial condition of a.v1, as in previous releases.
  end
  equations
    der(a.v1) == p1; % Uses variable from component A here. This is new
  end
end
```

- Simple setup function. In general, you cannot specify a block parameter as run-time if the underlying component uses it in the setup function. However, if the setup is restricted to simple operations like error-checking, you can declare it as:

```
function setup %#simple
   ...
end
```

In this case, many of the parameters used in the setup function can be selected as run-time parameters. For more information, see Simple Setup.

Additional MATLAB operators allowed in Simscape language

The following MATLAB operators are now allowed in a Simscape file:

Name	Discontinuous
`ones`	
`zeros`	
`cat`	
`horzcat`	
`vertcat`	
`sum`	
`prod`	
`mod`	Yes
`any`	Yes
`all`	Yes
`min`	Yes
`max`	Yes
`ndims`	
`length`	
`size`	
`numel`	
`isempty`	
`isequal`	Possibly, if arguments are real and have the same size and commensurate units
`isinf`	Yes
`isfinite`	Yes
`isnan`	Yes
`double`	
`int32`	Yes
`uint32`	Yes

Some of these operators were previously allowed only in the declaration section. They are now allowed in all the other sections as well.

All arguments that specify size or dimension must be unitless constants or unitless compile-time parameters.

Improved strictness and accuracy

As a continuation of improvements made in R2015b (see "Units handling refinements in Simscape language" on page 16-2), several more semantic changes have been made to increase robustness and consistency of simulation results:

- Fully typed parameter modification. In the old compiler, parameters could be updated with a value that either had commensurate unit or was unitless. In case of unitless value, the parameter preserved the original unit. In the new compiler, like in MATLAB, you can update a parameter with a value that has different size and unit. As a result, updating to a unitless value no longer preserves the original unit, which introduces a compatibility issue.

- Assertions on `let` variables. The old compiler effectively inlined `let` variable declarations into the code, and then added implicit assertions to the appropriate branches. The new compiler computes the values of `let` variables once per call, which greatly improves performance. As a result, if your `let` statement contains functions such as `sqrt(u)`, `log(u)`, and so on, the implicit assertion `u>=0` is added up front. Therefore, if the `in` part contains an `if` statement, with `u<0` in one of the branches, it triggers the assertion even if the `let` variable is used in another branch.
- Assertion for `a^b` where `a < 0 && ~isinteger(b)`. For minor time steps, the compiler still evaluates power as:

  ```
  if a < 0
    return -( (-a) ^ b );
  else
    return a^b;
  end
  ```

  However, it now issues a warning assertion if the above rule is violated for an accepted time step.

Compatibility Considerations

When you rebuild your custom libraries for use with R2016a, you can get compilation errors or simulation-time assertions if your Simscape files violate the new, stricter, language rules.

For example, consider the following component:

```
component MyComp
 components
   r1 = resistor;
   r2 = resistor(R=2);
 end
 function setup
   r1.R = 1;
 end
end
```

Updating a parameter to a unitless value no longer preserves the original unit, and therefore you will now get a compilation error about units being not commensurate. To avoid the error, specify fully typed values when modifying parameters. For example:

```
component MyComp
 components
   r1 = resistor;
   r2 = resistor(R={2, 'Ohm'});
 end
 function setup
   r1.R = {1, 'Ohm'};
 end
end
```

Another example concerns the assertions on `let` variables. Consider the following code:

```
  let
    v = sqrt(u);
  in
```

```
    if u < 0
      y == 0;
    else
      y == v;
    end
  end
```

Even though variable `v` is used only in the `else` branch, the implicit assertion is added at the beginning:

```
  assert(u >= 0);
  let
    v = sqrt(u);
  in
    if u < 0
      y == 0;
    else
      y == v;
    end
  end
```

Any negative value of `u`, instead of successfully executing through the `if` branch, will now trigger the assertion. To avoid the error, rewrite the file as follows:

```
  if u < 0
      y == 0;
  else
      y == sqrt(u);
  end
```

Change to Thermal Liquid domain definition

The energy flow rate replaces the thermal flux as a Through variable in the Foundation Thermal Liquid domain.

The new Through variable is part of a broader change to the energy equations used in Thermal Liquid components. The updated equations are now more readable and extensible, enabling you to more easily check and add energy terms. The equations omit kinetic energy terms in a common approximation that has negligible impact on simulation results.

Compatibility Considerations

If you created custom components for the Thermal Liquid domain, you might need to update your code to reflect the new domain definition.

New base classes for Thermal Liquid components

Components in the Thermal Liquid domain inherit from new base classes. Simscape software continues to support the old base classes but these components no longer depend on them. This change affects the converter, pipe, restriction, reservoir, and source components.

Compatibility Considerations

The new base classes omit certain variables defined in the old base classes. If you created custom components that inherit from these component classes and call any of the missing variables, you might need to update your code. This issue does not affect custom components that inherit directly from the old base classes.

Renamed Thermal Liquid component source files

Component files for Pipe (TL), Translational Mechanical Converter (TL), and Rotational Mechanical Converter (TL) blocks have been renamed. Use the new file names to open the files and view the source code for the blocks. The table shows the old and new files names.

New File Name	Old File Name
pipe_resistive	resistive_tube
pipe_resistive_compressibility	resistive_tube_with_dynamic_compressibility
pipe_resistive_compressibility_inertia	pipeline_segment
translational_converter	translational_converter_steady_compressibility
translational_converter_compressibility	translational_converter_dynamic_compressibility
rotational_converter	rotational_converter_steady_compressibility
rotational_converter_compressibility	rotational_converter_dynamic_compressibility

Foundation Library and Simulation

Run-Time Parameters: Speed up simulation tasks and modify component parameter values without regenerating C code

You can now modify certain block parameters between simulation runs without regenerating C code or triggering the diagram update.

Unlike Simulink blocks, where all parameters are tunable at run time unless otherwise specified, the default for Simscape block parameters is that they are modifiable at compile time. To make a parameter modifiable at run time, you have to designate it as such.

Tunable parameters (that is, parameters that you can modify during a simulation run) are not supported for Simscape blocks. If a Simscape parameter is designated as run-time, it means that you can modify its value between simulation runs without recompiling the model.

To specify run-time parameters:

1. Set the preference to show the run-time parameters drop-down in block dialogs. On the MATLAB Toolstrip, click **Preferences**. In the left pane of the Preferences dialog box, select **Simscape**, and then in the right pane, select the **Show run-time parameter settings** check box. Click **OK**.
2. Open a block dialog box in your model. Because of the preference setting, there is now an extra drop-down box next to each parameter. By default, all these drop-down boxes display `Compile-time`.
3. For the parameter that you want to modify, switch the drop-down value to `Run-time`. If the parameter currently has a numeric value, create a MATLAB variable and replace the parameter value with this variable.
4. In the tool bar at the top of the model window, click **Enable Fast Restart**.
5. You can now run the simulation repeatedly and change the variable value between simulations without recompiling the model.

For more information, see Run-Time Parameters.

Simulation Speed Improvements: Up to 5x speed improvement for simulations using generated code

In this release, there are significant speed improvements both for desktop simulations and for those using generated code. For example, a benchmark medium-sized electrical network, with approximately 200 nodes, shows the following results:

Simulation type	R2015b Time	R2016a Time	Speed Improvement
Desktop simulation	10.4 s	8.0 s	1.3 times
Simulink Coder™ Rapid Simulation Target (RSim)	12.2 s	6.1 s	2.0 times
Simulink Coder (GRT) with local solver	15.2 s	3.1 s	4.9 times

Stream Logging Data to Disk: Increase capacity for Simscape simulation results

In previous releases, logged simulation data was saved in a workspace variable of type `simscape.logging.Node`, named as specified by the **Workspace variable name** configuration parameter. Saving data to the workspace can slow down the simulation and consume memory. Data logging options, such as **Decimation** or **Limit data points**, helped alleviate the issue by limiting the number of saved data points. You could also try logging data for a subset of blocks, or using time intervals.

The new method of streaming logged data to disk significantly increases the data logging capacity, because you are no longer limited by the system memory. You can now log simulation data for all blocks in a large model, or for selected blocks for a very long simulation time.

To stream data to disk, open the MATLAB Preferences dialog box, go to the **Simscape** pane, and select the **Stream data to temporary disk directory** check box. When this preference is turned on, the simulation data, in the form of a `simlog` object generated during simulation, is stored in an HDF5 file in a temporary directory. The workspace variable of type `simscape.logging.Node`, named as specified by the **Workspace variable name** configuration parameter, still gets created, but instead of storing all the simulation data it references the `simlog` object in the temporary file. The temporary file persists as long as there is a logging variable name in the workspace that references it.

You view and analyze logged simulation data exactly as before, with all the interaction between the workspace variable and the stored object happening behind the scenes.

There are two new functions that handle saving logged data to a file (other than the temporary file) and loading it from a file into the workspace:

- `simscape.logging.export(`*`simlog`*`,'`*`fileName`*`')` stores the simulation data in a specified HDF5 file. *`simlog`* is the simulation log variable name. *`fileName`* is the name and path to destination file.
- *`var`* `= simscape.logging.import('`*`fileName`*`')` creates a workspace variable of type `simscape.logging.Node`, which references the `simlog` object in the specified HDF5 file. *`var`* is the variable name. If you do not assign a variable name when calling the function, then the workspace variable name is `ans`.

By default, the **Stream data to temporary disk directory** preference is turned off, and data logging works the same as in previous releases.

For more information, see Stream Logging Data to Disk.

Simscape Component Block: Efficiently create custom components by selecting Simscape language file directly from block

In previous releases, generating Simscape blocks from textual component files required building the custom block libraries. You can still use the `ssc_build` command to generate a custom block library from a complete package of Simscape component files. However, the new Simscape Component block in the Utilities library lets you generate a Simscape block directly from a textual component file, skipping the library build process.

For more information, see Selecting Component File Directly from Block.

Simscape Results Explorer Unit Selection: Select y-axis units directly in plot pane

When you display logged simulation data in Simscape Results Explorer, the data along the *x*-axis is always time, in seconds. However, you can now change the *y*-axis units directly on the plot.

Each of the plots now has a drop-down arrow next to the unit name for the *y*-axis. When you click this arrow, a context menu appears containing names of all the units in the unit registry that are commensurate with the current plot unit, as well as two other options:

- `Default` — Use the default unit.
- `Specify` — Type the unit name or expression in a pop-up window and click **OK**. The specified unit name or expression must be commensurate with the current plot unit.

Once you select the option you want, the drop-down menu collapses and the plot is redrawn in specified units.

For more information, see Plot Simulation Data in Different Units.

Compatibility Considerations

Under **Plot options**, you can switch the **Plot signals** option from `Separate` to `Overlay`, to overlay multiple plots on the same axes. In previous releases, you could overlay plots with commensurate *y*-axis units. Now, if you want to overlay several plots, select the same *y*-axis unit for all these plots.

PS Lookup Table (3D) Block: Graphically define implicit equations that require lookup tables with three independent variables

The new PS Lookup Table (3D) block computes an approximation to some function `f=f(x1,x2,x3)` given the `x1`, `x2`, `x3`, `f` data points. The three inputs and the output are physical signals. See the block reference page for details.

Compatibility Considerations

With the addition of three-dimensional table lookup, the axis naming schema has moved from `(x,y,z)` to `(x1,x2,x3)`. Parameter names in the PS Table Lookup (1D) and PS Table Lookup (2D) blocks have been changed, to make them consistent with the PS Lookup Table (3D) block. The parameter names in the block source have also been changed, to make them consistent with the new naming schema.

If you use these blocks directly in your models, there is no compatibility impact. However, if you use these blocks as composite component members in Simscape language, you might need to update your code to use the new parameter names in the block source.

Block	Old Parameter Name	New Parameter Name
PS Table Lookup (1D)	`x_t`	`x`
PS Table Lookup (1D)	`y_t`	`f`
PS Table Lookup (2D)	`x_t`	`x1`
PS Table Lookup (2D)	`y_t`	`x2`

Block	Old Parameter Name	New Parameter Name
PS Table Lookup (2D)	`z_t`	`f`

Validation of Signal Units: Ensure consistent units specification on Simulink signals connected to Simscape physical networks

You can now specify physical units on Simulink signals. For details, see Simulink Units: Specify, visualize, and check consistency of units on interfaces.

If you specify physical units on a Simulink signal connected to a Simulink-PS Converter or a PS-Simulink Converter block, this unit must be consistent with the unit specified inside the block. On a Simulink-PS Converter block, you specify the unit using the **Input signal unit** parameter. On a PS-Simulink Converter block, you specify the unit using the **Output signal unit** parameter. If the parameter value does not match the physical unit of the Simulink signal connected to the block, you get a warning.

If the block parameter value is `1`, it matches the Simulink signal unit designated as `1`.

Compatibility Considerations

The default Simscape unit registry has been made consistent with the Simulink unit database. If the Simscape and Simulink databases used different symbols for the same unit, both symbols are now in the Simscape database and you can use either one.

Quantity	Unit	Simscape Symbol	Simulink Symbol
Energy	British thermal unit	`Btu`	`Btu_IT`
Power	Horsepower	`HP`	`HP_DIN`
Viscosity absolute	Poise	`Poise`	`P`
Viscosity kinematic	Newt	`Newt`	`newt`

In several cases, there was a conflict between the Simulink and Simscape symbols.

Unit	Simscape Symbol	Simulink Symbol
Coulomb	`c`	`C`
Degree Celsius	`C`	`degC`
Degree Rankine	`R`	`degR`
Roentgen	-	`R`

This conflict resulted in the following changes to Simscape units, with potential compatibility impact:

- The `C` symbol for degree Celsius has been replaced with `degC`. Both `c` and `C` now indicate Coulomb.
- The `R` symbol for degree Rankine has been replaced with `degR`.
- The `Fh` symbol was not part of the conflict, but the `degF` symbol has been added for consistency. You can use either one for degree Fahrenheit.

If your legacy models use Simulink-PS Converter or PS-Simulink Converter blocks with unit specified as `C` or `R`, you get a compile-time error. Open the Upgrade Advisor, select **Check model for block upgrade issues**, and click **Run This Check** to find and modify all blocks with outdated units.

If there is no conflict between the old and new symbols (for example, `Btu` and `Btu_IT`), your legacy models do not need to be updated. Only the new symbol (in this case, `Btu_IT`) will appear in the dialog box drop-downs for commensurate units, but you can type `Btu` in unit expressions. However, It is recommended that you use the new symbol names going forward.

To distinguish between absolute and relative temperatures, Simscape unit manager lets you select whether to apply affine conversion (see Thermal Unit Conversions). By contrast, Simulink database contains a separate set of temperature units to indicate relative temperatures: `deltaK`, `deltadegC`, `deltadegF`, `deltadegR`. These units have also been added to the Simscape database. If you use them, affine conversion does not apply. The old method, of specifying regular temperature units and selecting whether to apply the affine conversion, works without change; however, you can get warnings about unit mismatch from the Simulink side of your model.

If you add a new unit to your unit registry, by using the `pm_addunit` function, and your unit definition conflicts with the one in the Simulink database, you will get a warning that the Simscape unit is not compatible with the Simulink unit database. If you add a unit that does not exist in the Simulink database, you will get a warning about undefined unit. Note that this warning applies only to the Simulink database; the Simscape physical network works as expected. For information on how to turn off these warnings, see Working with Simulink Units.

Another issue affects legacy models where multiple Simulink-PS Converter or PS-Simulink Converter blocks share the same Simulink signal. For example, you can no longer connect multiple Simulink-PS Converter blocks with different units to the same Simulink source, such as a Constant block.

Thermal Resistor block in the Electrical Elements library

The Thermal Resistor block represents a temperature-dependent resistor. When the temperature at the thermal port is T, the resistance is $R = R_0(1+\alpha(T–T_0))$, where R_0 is the nominal resistance at the reference temperature T_0 and α is the temperature coefficient.

This block was previously part of the Simscape Electronics™ Passive Devices library. The ability of the Simscape Electronics Resistor block to model thermal effects, introduced in R2015b, rendered the Thermal Resistor block superfluous. Therefore, this block has now been moved to the Simscape Electrical Elements library.

There is no compatibility impact on existing models.

Mass Flow Rate & Thermal Flux Sensor (TL) block renamed

The energy flow rate replaces the thermal flux as a Through variable in the Foundation Thermal Liquid domain. To reflect this change, the Mass Flow Rate & Thermal Flux Sensor (TL) block has been renamed to Mass & Energy Flow Rate Sensor (TL). The updated block senses not the thermal energy flow rate but the total energy flow rate—the sum of thermal energy and pressure-volume work terms.

Compatibility Considerations

The variable measured through port Phi of the Mass & Energy Flow Rate Sensor (TL) block has changed. Older models dependent on thermal energy flow rate measurements can give inaccurate simulation results.

Thermal Liquid volumetric flow rate source and sensor blocks

The Thermal Liquid library now provides source and sensor blocks for setting and sensing volumetric flow rates:

- Volumetric Flow Rate Source (TL)
- Controlled Volumetric Flow Rate Source (TL)
- Volumetric Flow Rate Sensor (TL)

The sources are ideal and adiabatic. Friction losses and heat exchange with the source environment are ignored. Use the Volumetric Flow Rate Source (TL) block for constant flow rates and the Controlled Volumetric Flow Rate Source (TL) for variable flow rates.

Custom environment pressure in Thermal Liquid converter blocks

Thermal Liquid converter blocks provide a new option to specify the external environment pressure. Affected blocks include Translational Mechanical Converter (TL) and Rotational Mechanical Converter (TL). You can set the environment pressure to the standard atmospheric value of 0.101325 MPa or to a custom value that you specify. The environment pressure adds to the total pressure the converters must overcome to generate force or torque.

Numerical improvements in Thermal Liquid blocks for greater model fidelity

Various block parameters and equations have changed to provide greater model fidelity. Changes include:

- Revised Nusselt number calculations in the Pipe (TL) block — In the laminar regime, the Nusselt number is now a constant. This change reflects the negligible thermal entrance effects assumed in pipe segments. In the turbulent regime, the Nusselt number now follows from the Gnielisnki correlation. The new correlation improves numerical accuracy relative to the Colburn equation that it replaces.
- Scaling of minimum thermal conductance in the Thermal Liquid Settings (TL) block — The minimum thermal conductance in a thermal liquid network now scales with component geometry. This parameter sets a lower bound on thermal conductance which, during flow reversals, helps to smooth out heat flow transitions. The new geometric scaling enables you to more accurately model slender components with large aspect ratios.
- Updated discharge coefficient definition in Local Restriction (TL) and Variable Local Restriction (TL) blocks — The new discharge coefficient parameter relates the actual mass flow rate to its theoretical value according to the expression

$$\dot{m} = C_d S_r \sqrt{\frac{2\rho\Delta p}{1 - (S_r/S)^2}},$$

where:

- $\dot{m}$ is the mass flow rate.
- C_d is the discharge coefficient.
- S_r is the restriction area.
- S is the pipe area.
- ρ is the liquid density.
- Δp is the pressure drop across the restriction.

The previous definition corresponds to the more aptly named flow coefficient, given by

$$\dot{m} = C_v S_r \sqrt{2\rho\Delta p}$$

where C_v is the flow coefficient. The change in coefficient definition has a negligible impact on simulation results when the restriction area is less than one quarter of the adjacent pipe area.

- New cross-sectional area parameter for thermal conduction calculations in the Rotational Mechanical Converter (TL) block. The block uses the new parameter to determine the aspect ratio of the converter, L/S, where S is the thermal conduction length and S is the specified cross-sectional area. The thermal conduction length is assumed equal to the cross-sectional diameter.

Sign reversal of thermal expansion coefficient

The sign of the **Isobaric coefficient of thermal expansion** parameter in the Thermal Liquid Settings (TL) block has changed. The updated block automatically reverses the coefficient sign in older models so you do not have to. The new sign convention is the same used by the National Institute of Standards and Technology (NIST) in the fluid properties database REFPROP:

$$\alpha = \frac{1}{V}\left(\frac{\partial V}{\partial T}\right)_p = -\frac{1}{\rho}\left(\frac{\partial \rho}{\partial T}\right)_p,$$

where:

- α is the isobaric coefficient of thermal expansion
- V is the volume
- T is the temperature
- p is the pressure
- ρ is the density

Functionality being removed or changed

Simscape Language Keyword Name	What Happens When You Use the Keyword?	Use This Instead	Compatibility Considerations
`Hidden` attribute for members	Still runs	`ExternalAccess`	See Compatibility Considerations under “Improved attributes for member visibility and access” on page 14-4.

R2015aSP1

Version: 3.13.1

Bug Fixes

R2015b

Simscape Language

Units handling refinements in Simscape language

To increase robustness and consistency of simulation results, several semantic changes have been made with respect to how Simscape language handles math operations on physical units:

- When you declare a member as a value with unit, or use `{value,'unit'}` in equations, `value` must be a unitless expression. In previous releases, the `value` expression could already have a unit, which sometimes resulted in double conversions by the unit manager.
- Arguments passed to transcendental functions must be dimensionless. Examples of transcendental functions include the exponential function, the logarithm, and the trigonometric functions. Previously, these functions accepted a value with unit as an argument, but did not handle units. For example, if *p* is pressure, `log(p)` would take a logarithm of the value but ignore the units, and therefore have different results depending on whether pressure was defined in Pa or MPa.

  Now you have to pass a dimensionless argument to these functions, by converting to units of `1` before applying the function. In the example above, instead of `log(p)`, you now have to use `log(value(p,'Pa'))`, or equivalently `log(p/{1,'Pa'})`.

Compatibility Considerations

When you rebuild your custom libraries for use with R2015b, you can get `ssc_build` errors if your Simscape files violate the new, stricter, rules of units handling. Modify the source file to convert to units of `1`, as described above.

Foundation Library and Simulation

Two-Phase Fluid domain and block library

The Foundation library now contains a two-phase fluid domain and Two-Phase Fluid block library. This library contains chambers, reservoirs, local restrictions, and energy converters, sources and sensors, as well as a Two-Phase Fluid Properties (2P) block, which controls fluid properties for the attached circuit.

Use these blocks for modeling two-phase fluid systems, where the working agent is part liquid and part vapor.

For more information, see the block reference pages. See also Two-Phase Fluid Domain for information on the two-phase fluid domain definition.

Variable priority option None for initialization

Variable initialization priority `None` is now available in block dialogs, in addition to `High` and `Low`. In previous releases, all three values were available only in the Simscape language, when writing component files. When using a block, you could only specify `High` or `Low` priority, or leave the variable as `Unused`. With this change, the options available in the **Variables** tab of the block dialog box align exactly with the options available in the Simscape file.

Each variable has a beginning value before the initialization algorithm finds consistent values for the system of equations. When the variable priority is `None`, the algorithm starts at this beginning value but does not remember the beginning value as the algorithm finds a solution. When the variable priority is `Low`, the beginning value becomes a target for the algorithm and the algorithm tries to stay close to the target. When the variable priority is `High`, the beginning value becomes a target for the algorithm and the algorithm tries to meet the target exactly.

The component author specifies default initialization priority, beginning value, and unit for each variable in the Simscape file. When using the block, you can override these default settings for any of the public variables (that is, variables that appear on the **Variables** tab of the block dialog box) by selecting the **Override** check box for that variable and specifying your own values.

If you clear the **Override** check box next to a variable name, then its **Priority**, **Beginning Value**, and **Unit** fields switch back to defaults specified in the component file. However, if you select the check box again, these fields will retain their last specified value for when they were overridden.

Periodic Operators library

The Physical Signals library now has a Periodic Operators sublibrary, which contains the following blocks:

- PS Constant Offset Estimator
- PS Harmonic Estimator (Amplitude, Phase)
- PS Harmonic Estimator (Real, Imaginary)
- PS RMS Estimator

Use these blocks for various aspects of periodic signal measurements.

Speed and efficiency improvements for simulation of switched linear systems

Performance optimizations implemented in this release allow switched linear models to run faster in desktop simulation, in both Normal and Accelerator modes. The average simulation time for switched linear systems is around 1.3x faster than in previous release. The actual speedup varies depending on the particular model.

To take advantage of the new optimizations, switched linear models:

- Cannot contain blocks that register sample times, such as Counter or Random Number blocks.
- Cannot log simulation statistics. Open the Configuration Parameters dialog and, on the **Simscape** pane, clear the **Log simulation statistics** check box.

Domain-specific line style propagation through block icons

Domain-specific line styling, introduced in R2014b, now applies to the block icons as well. Relevant parts of each block icon assume domain-specific line styles and colors, with the connection lines continuing into the icon, so that the block diagram closely resembles a circuit schematic. The connection lines now have rounded corners. These enhancements improve the readability and clarity of block diagrams.

Rotational Inerter and Translational Inerter blocks

The new Rotational Inerter and Translational Inerter blocks represent two-port inertia in rotational and translational mechanical systems, respectively.

Use these blocks in high performance suspension systems, to decouple weave and roll modes, or in applications where you need to model a passively tuned mass-spring-damper response.

Real-Time Performance Advisor checks for physical models

If you have a Simulink Real-Time license, the new `Execute real-time application` activity mode in Performance Advisor lets you optimize your model for real-time execution. This mode includes several checks specific to physical models. These checks are organized in the **Simscape checks** folder, with subfolders for the add-on products, such as **SimDriveline checks** or **SimElectronics checks**. The top-level **Simscape checks** are applicable to all physical models. Each of the subfolders contains checks that target specific blocks from that add-on product.

To access the checks, in the Performance Advisor window, under **Activity**, select `Execute real-time application`. In the left pane, expand the **Real-Time** folder, and then the **Simscape checks** folder. Run the top-level Simscape checks. If your model contains blocks from an add-on product, also run the checks in the subfolder corresponding to that product.

For more information on using the Performance Advisor, see Automated Performance Optimization.

Real-Time Simulation Documentation Enhancements

The documentation on real-time and hardware-in-the-loop simulation of Simscape models has been revised and expanded. See Real-Time Simulation.

Variable Viewer link to block diagram renamed

The Variable Viewer option that highlights the relevant block in the block diagram has been renamed from **Highlight block** to **Go to block**, to avoid ambiguity in SimMechanics™ models.

When you right-click in the **Name** column of any row in the Variable Viewer table, a context menu opens with the following options:

- **Go to block** — Highlights the corresponding block in the block diagram, opening the appropriate subsystem if needed. If the row represents a variable, highlights the parent block for this variable.
- **Open block dialog** — Opens the corresponding block dialog box (for a variable, opens the parent block dialog box). This enhancement makes it easy to go from the Variable Viewer to the **Variables** tab in the block dialog box, to modify the variable priorities and targets.

For more information, see Variable Viewer.

Naming schema change within the data logging variable structure

When you enable simulation data logging, the workspace variable containing the logged data mimics the structure of the model. Nodes within the variable have names that are derived from the blocks and variables in the model. However, each node has to be a valid and unique MATLAB identifier, and therefore the block names undergo changes, such as replacing spaces with underscores.

In previous releases, this name processing also involved a built-in string-to-integer function that stripped leading zeros from numbers in block names. For example, for a Resistor block named `R0001` in the block diagram, the corresponding logging variable node was named `R1`. If your block diagram contained two resistors, `R1` and `R0001`, then one node was named `R1` and the other `R2`, to satisfy the unique identifier rule. As a result, the logging variable nodes could have names that did not match the block diagram.

The current naming schema preserves the leading zeros in numbers included in block names. In the example above, the node corresponding to resistor `R0001` is now named `R0001`.

Compatibility Considerations

If your models have blocks with names that contain numbers with leading zeros (such as `R0001`), then the names of the corresponding nodes in the data logging structure have changed compared to the previous release. If you have scripts that rely on these node names being the same, you need to update them.

R2015a

Foundation Library and Simulation

Variable Viewer link to block diagram

The Variable Viewer tool now provides a direct link to the relevant block in the block diagram.

When you right-click in the **Name** column of any row in the Variable Viewer table, a context menu opens with the following options:

- **Highlight block** — Highlights the corresponding block in the block diagram, opening the appropriate subsystem if needed. If the row represents a variable, highlights the parent block for this variable.
- **Open block dialog** — Opens the corresponding block dialog box (for a variable, opens the parent block dialog box). This enhancement makes it easy to go from the Variable Viewer to the **Variables** tab in the block dialog box, to modify the variable priorities and targets.

For more information, see Variable Viewer.

Improved solver efficiency for model initialization

The initial conditions solver speed increased significantly in this release without sacrificing the robustness.

The initial conditions solver works much faster in general. If the initial condition solve fails, the reaction time is much faster as well. The solver also reacts almost instantaneously when you press CTRL-C.

Sparkline plots for logged data

You can view sparkline mini-plots of logged simulation data for selected blocks and variables directly on the model canvas.

Before using this functionality, you must enable data logging, for all or some of the blocks, and run the simulation. For more information, see Data Logging.

To view the sparkline plots, in the model window, from the top menu bar, select **Display** > **Simscape** > **Toggle Sparklines When Clicked**. This action adds the check mark next to the **Toggle Sparklines When Clicked** menu option, and you can start selecting blocks to display sparkline plots of logged data for their variables.

When you select a block you can see sparkline plots for its variables. Hover over the variable name to see the plot. You can select which variables to plot (the first three are shown by default).

If you select a block for which simulation data is not being logged, it displays `No variables` instead of the sparkline plots. Right-click the block, select **Simscape** > **Log simulation data**, and rerun the simulation.

Repeatedly selecting a block toggles the display of its sparkline plots on and off.

To clear all plots and start again with a clean canvas, select **Display** > **Simscape** > **Remove All Sparklines**. Then you can select more blocks and variables to display their sparkline plots.

Repeatedly selecting the **Toggle Sparklines When Clicked** menu option toggles the ability to view the sparkline plots for the model on or off, as indicated by the check mark.

For more information, see View Sparkline Plots of Simulation Data.

Data logging for models created using ssc_new

When you create a new model using the `ssc_new` function, simulation data logging for this model is now turned on by default. Data logging configuration parameters are automatically set to the following values:

- **Log simulation data** — `All`.
- **Log simulation statistics** — Off.
- **Open viewer after simulation** — Off.
- **Workspace variable name** — `simlog`.
- **Decimation** — `1`.
- **Limit data points** — On.
- **Data history (last N steps)** — `10000`.

Using data logging is a best practice for Simscape models because it provides access to important simulation and analysis tools. For more information, see Data Logging.

Pressure setting and name change for Thermal Liquid reservoir blocks

The two reservoir blocks in the Thermal Liquid/Elements library serve as reference points in a pipe network, where you can specify an arbitrary reference temperature. In previous versions, the pressure in these blocks was always set to atmospheric pressure. Now you have an option to specify an arbitrary value for the reference pressure, as well as temperature.

Both blocks now have a new parameter, **Reservoir pressure specification**, which lets you select between two values:

- `Atmospheric pressure` (default)
- `Specified pressure` — If you select this value, an additional parameter, **Reservoir pressure**, appears in the dialog box to let you enter the reference pressure value.

The block names have been changed:

Old Name	New Name
Temperature Reservoir (TL)	Reservoir (TL)
Controlled Temperature Reservoir (TL)	Controlled Reservoir (TL)

There is no compatibility impact. When you open an existing model containing these blocks, it is updated automatically to use the new version of the blocks.

New examples

Examples introduced in this version are:

- Lithium-Ion Battery Pack With Fault
- Transmission Line

R2014b

Simscape Language

Refrigeration example modeling two-phase fluid flow

A new example shows how you can use Simscape language to model two-phase fluid flow. The example includes a custom two-flow domain and a library of custom components based on this domain.

To open the custom library, type `two_phase_lib` at the MATLAB Command prompt. Double-click any block in the library to open its dialog box, and then click the **Source code** link in the block dialog box to open the Simscape source file for this block in the MATLAB Editor. The custom domain file is located in the same folder as the component files.

To open the Two-Phase Flow Refrigeration Model built of these custom components, type `ssc_refrigeration` at the MATLAB Command prompt.

Foundation Library and Simulation

Domain-specific line styles for representing physical connections

Each Simscape domain now uses a distinct default color and line style for the connection lines. Physical signal lines now also have a distinct style and color. This enhancement improves the readability of block diagrams.

Improved interface for connecting Simscape blocks

The following usability enhancements have been made in this release:

- Visual hint for an invalid connection attempt — You cannot connect ports that belong to different domains, but in previous releases it was not always obvious why the connection was not being made. Now, as you are trying to connect a Simscape port, the cursor appearance changes just as it approaches a port or line that belongs to a different domain, to indicate that the connection here is invalid.
- If you place a Simscape block with two ports (one on each side) over a connection line that belongs to the appropriate domain, the block gets inserted automatically.
- When you connect Simscape blocks, the ports on either side of the connection line disappear. Physical signal output ports also disappear. Physical signal input ports remain, to show the signal direction.

Option to configure logging of Simscape simulation results per block

In previous releases, once you enabled simulation data logging, it included data for the whole model. The new `Use local settings` option lets you log data for selected blocks only. The basic workflow is as follows:

1. Enable simulation data logging on a block-by-block basis. In the model window, from the top menu bar, select **Simulation** > **Model Configuration Parameters**. In the Configuration Parameters dialog box, in the left pane, select **Simscape**, then set the **Log simulation data** parameter to `Use local settings`. Click **OK**.
2. Select the blocks in your model by right-clicking on each block and then, from the context menu, selecting **Simscape** > **Log simulation data**. A check mark appears in front of the **Log simulation data** option.
3. Simulate the model. When the simulation is done, the simulation data log contains only the data from the selected blocks.

To stop logging data for a previously selected block, right-click on it and select **Simscape** > **Log simulation data** again to remove the check mark.

If you set the **Log simulation data** parameter to `All`, the simulation log will contain data from the whole model, regardless of the block selections. Setting the **Log simulation data** parameter to `None` disables data logging for the whole model.

Flat view option and improved diagnostics for Variable Viewer

You can now control the number of rows in the Variable Viewer by switching between the tree view (the default) and the flat view. By default, the Variable Viewer opens in tree view, with variable nodes grouped under the parent port, block, and subsystem nodes.

To switch to the flat view, click Flat in the Variable Viewer toolbar. The table now contains one row per variable, with the **Name** column showing the full path to the variable, in the following format:

`SubsystemName --> BlockName --> PortName --> VariableName`

If the Variable Viewer is in flat view, the buttons that expand and collapse nodes are disabled. To switch back to the tree view, click Tree in the Variable Viewer toolbar.

Another improvement is better diagnostics of the overall status. A message in the bottom-right corner of the Variable Viewer window tells you whether the viewer displays the variables at start, imported variables (when you load an initial state or SimState), or after update diagram (when the previously computed actual values become unavailable).

Simscape Results Explorer

Simscape Results Explorer is an interactive tool that lets you navigate and plot the simulation data logging results. This tool is an enhanced version of the Simscape Simulation Results Explorer (`ssc_explore.m`) previously posted on MATLAB Central. It is now part of the product. To use the tool:

1. In the model window, from the top menu bar, select **Simulation** > **Model Configuration Parameters**. In the Configuration Parameters dialog box, in the left pane, select **Simscape**.
2. Set the **Log simulation data** parameter to `All` or `Use local settings`, to enable data logging.
3. Select the **Open viewer after simulation** check box. Click **OK**.
4. If using local settings, select the blocks in your model as described in “Option to configure logging of Simscape simulation results per block” on page 18-3.
5. Simulate the model. When the simulation is done, the Simscape Results Explorer window opens. It contains the simulation log tree hierarchy in the left pane. When you click on a node in the left pane, the corresponding plot appears in the right pane.

You can control whether the Simscape Results Explorer window is reused when you rerun the simulation, or a new window is opened after the next simulation run, by linking and unlinking the window.

When you first open the Simscape Results Explorer window, it is linked to the current MATLAB session. This means that when you run a new simulation, the results in the window will be overwritten. To retain the current results and open a new window after the next simulation, click the button located in the toolbar above the left pane. The button appearance changes to and, when the new window opens after simulation, that window will be linked to the session. Only one window can be linked to the session, so if you have multiple windows open, linking one of them (by clicking on its button) unlinks the previous one.

Simscape logging into Simulink single output

If you select the **Save simulation output as single object** check box on the **Data Import/Export** pane of the Configuration Parameters dialog box, Simscape log data will now be part of the single output object. In this case, it is not stored as a separate workspace variable.

If the **Save simulation output as single object** check box is not selected, simulation data is stored in the current workspace, with the name of the workspace variable specified by the **Workspace variable name** configuration parameter. The default variable name is `simlog`.

This enhancement makes Simscape data logging compatible with the `parfor` command. For more information, see Save simulation output as single object.

Compatibility Considerations

In previous releases, simulation data was stored as a variable in the base workspace. Now, if the single output is not enabled (which is the default), this variable is stored in the current workspace, instead of the base workspace. If the single output is enabled, simulation data becomes part of the single output object and is not stored as a separate workspace variable. Therefore, if you have created custom scripts or functions that rely on accessing the `simlog` variable in the base workspace, you will have to modify them accordingly.

Simscape logging integration with Simulation Data Inspector

You can now configure your model to automatically record Simscape logging data, along with the rest of simulation data obtained from a model run, using the Simulation Data Inspector. The basic workflow is as follows:

1. Set up your model to log simulation data. In the model window, from the top menu bar, select **Simulation > Model Configuration Parameters**. In the Configuration Parameters dialog box, in the left pane, select **Simscape**, then set the **Log simulation data** parameter to `All` or `Use local settings`. If using local settings, select the blocks to log data from, as described in “Option to configure logging of Simscape simulation results per block” on page 18-3.
2. Enable data recording. In the Configuration Parameters dialog box, in the left pane, select **Data Import/Export**, then select the **Record logged workspace data in Simulation Data Inspector** check box.
3. Simulate the model. When the simulation is done, a notification bar appears in the Simulink Editor.
4. In the notification bar, click the link to open the Simulation Data Inspector and view the results.

For detailed information on how to configure and use the Simulation Data Inspector, see Inspect Signal Data with Simulation Data Inspector.

Infinite resistance and port terminator blocks

Infinite resistance blocks let you specify initial difference for the appropriate Across variable (such as voltage, pressure, temperature) between two nodes without affecting model equations. The following new blocks in Foundation library implement infinite resistance for their respective domains:

- Infinite Flow Resistance (TL)
- Infinite Hydraulic Resistance
- Infinite Pneumatic Resistance
- Infinite Resistance
- Infinite Thermal Resistance

These blocks have no parameters, but their **Variables** tab lets you set the priority and initial target value for the appropriate Across variable.

Similarly, the conserving port terminator blocks now display the **Variables** tab, which lets you set the priority and initial target value for the appropriate Across variable at a node. The affected blocks are:

- Cap (TL)
- Hydraulic Cap
- Open Circuit
- Perfect Insulator
- Rotational Free End
- Translational Free End

The PS Terminator block has been added to the Sinks sublibrary of the Physical Signals library. It lets you cap physical signal output ports that do not connect to other blocks. Unlike conserving ports in physical modeling, or Simulink output ports, unconnected physical signal output ports do not generate warnings. However, you can use a PS Terminator block for clarity, to indicate that the signal was not inadvertently left unconnected.

Performance improvements for linear systems in Normal and Accelerator modes

Linear optimizations have been extended to include affine systems (for example, models containing blocks like DC Voltage Source), which results in faster simulation. Also, increased efficiency of generated code for linear and affine systems provides improved performance in Accelerator mode and for code generation.

R2014a

Simscape Language

ssc_build library build process that no longer requires C compiler

In previous releases, the library build process, whether performed using the `ssc_build` command or using the `ssc_mirror` command with the `buildmirror` flag set to `true`, required a C compiler. Before running either of these commands for the first time, you had to set up your compiler by running `mex -setup`.

In-memory execution of the library build process, implemented in this release, increases the process speed and eliminates the need for a C compiler. Therefore, you no longer need to have a C compiler available on your machine in order to build a custom block library.

priority attribute for setting relative priority of variable target values during initialization

The new initialization process, implemented in this release, involves block-level variable initialization. That is, a Simscape block dialog box now has an additional **Variables** tab, which lists all the public variables specified in the underlying component file, along with the initialization priority, target initial value, and unit of each variable. The block user can change the variable priority and target, prior to simulation, to affect the model initialization. The default values for variable priority, target value, and unit come from the variable declaration in the component file.

To enable the block author to specify default priority for a variable, a new attribute is now available. The `priority` attribute can have one of the three values: `priority.high`, `priority.low`, and `priority.none`. The default is `priority.none`, which is equivalent to leaving out the `priority` attribute entirely.

To specify a high or low default priority for a component variable, declare the variable as a field array. For example, the following declaration initializes variable `t` (spring deformation) as 0 mm, with high priority:

```
variables
    t = { value = { 0 , 'mm' }, priority = priority.high }; % Spring deformation
end
```

The old way of declaring variables still works. In fact, leaving out the `priority` attribute (that is, using `priority.none`) is suitable in most cases. The block user can control the variable initialization priority, as needed, by using the **Variables** tab of the block dialog box.

For example, you can declare the same variable `t` as follows:

```
variables
    t = { 0 , 'mm' }; % Spring deformation
end
```

In this case, the **Variables** tab in the block dialog box will have the **Spring deformation** variable listed initially as `Unused` (which means its priority and target are not used in the initialization process). The block user can modify the variable priority, as well as the target value and unit, in the **Variables** tab of the block dialog box prior to simulation.

If there are no top-level public variables declared in the component file (for example, if the top-level variables are declared as hidden), the **Variables** tab does not appear in the block dialog box. The same is true for composite components, because they also have no top-level public variables.

For more information on block-level variable initialization, see “Variables tab for specifying target value and priority for new initialization process” on page 19-4.

Script for replacing through and across statements with branches and equations

The `ssc_update` script is now available to help you update the legacy component files that contain `across` and `through` statements. The scripts replaces the `through` statements with the corresponding `branches` section, and adds the equations equivalent to the `across` statements to the `equations` section of the file.

For more information on the old and new syntax, see Compatibility Considerations under “branches section for defining the relationship between component Through variables and nodes” on page 20-2.

To run the script, at the MATLAB command prompt, type:

```
ssc_update package
```

where *package* is the name of a top-level package directory, without the leading `+` character. If you run the `ssc_update` command from inside the package directory structure, you can omit the argument. The script updates all the legacy component files located in the package. For more information, see `ssc_update`.

Foundation Library and Simulation

Variables tab for specifying target value and priority for new initialization process

New initialization process, implemented in this release, gives you more control over model initialization. Most of the Foundation library blocks now have a new **Variables** tab, which lists all the public variables specified in the underlying component file, along with priority, initial value, and unit. In most cases, the default value for each of these is `Unused`. Once you select the check box next to a variable name, you can specify its priority (`High` or `Low`), target initial value, and unit.

If the underlying component has no top-level public variables (such as a composite component, or one with the top-level variables declared as hidden), then the block dialog box does not have the **Variables** tab. For that reason, the Utilities library blocks, most of the Foundation library sensors and sources, and the absolute majority of blocks in the add-on products are not affected by this change.

The values you specify during block-level variable initialization are not the actual values of the respective variables, but rather their target values at the beginning of simulation ($t = 0$). Depending on the results of the initial conditions solve, some of these targets may or may not be satisfied. The solver tries to satisfy the high-priority targets first, then the low-priority ones:

- At first, the solver tries to find a solution where all the high-priority variable targets are met exactly, and the low-priority targets are approximated as closely as possible. If the solution is found during this stage, it satisfies all the high-priority targets. Some of the low-priority targets might also be met exactly, the others are approximated.
- If the solver cannot find a solution during the first stage, it issues a warning and enters the second stage, where `High` priority is relaxed to `Low`. That is, the solver tries to find a solution by approximating both the high-priority and the low-priority targets as closely as possible.

After you initialize the block variables and prior to simulating the model, you can open the Variable Viewer to see which of the variable targets have been satisfied. For more information on block-level variable initialization and Variable Viewer, see Variable Initialization.

Compatibility Considerations

In previous releases, several Foundation library blocks contained parameters that let you specify an initial value for an internal block variable at the start of simulation. These parameters have now been removed. The following table lists the initialization parameters that have been removed from block dialogs and the names of the corresponding block variables:

Block Name	Parameter Name	Variable Name
Capacitor	**Initial voltage**	**Capacitor voltage**
Constant Volume Chamber (TL)	**Fluid initial pressure** **Fluid initial temperature**	**Pressure** **Temperature**
Constant Volume Hydraulic Chamber	**Initial pressure**	**Pressure (gauge)**

Block Name	Parameter Name	Variable Name
Constant Volume Pneumatic Chamber	**Initial pressure** **Initial temperature**	**Pressure** **Temperature**
Fluid Inertia	**Initial flow rate**	**Flow rate**
Hydraulic Piston Chamber	**Initial pressure**	**Pressure (gauge)**
Inductor	**Initial current**	**Inductor current**
Inertia	**Initial velocity**	**Rotational velocity**
Mass	**Initial velocity**	**Velocity**
Mutual Inductor	**Winding 1 initial current** **Winding 2 initial current**	**Primary current** **Secondary current**
Pneumatic Piston Chamber	**Initial pressure** **Initial temperature**	**Pressure** **Temperature**
Rotational Hard Stop	**Initial angular position**	**Angular position**
Rotational Spring	**Initial deformation**	**Deformation**
Rotary Pneumatic Piston Chamber	**Initial pressure** **Initial temperature**	**Pressure** **Temperature**
Thermal Mass	**Initial temperature**	**Temperature**
Translational Hard Stop	**Initial position**	**Position**
Translational Spring	**Initial deformation**	**Deformation**
Variable Hydraulic Chamber	**Initial pressure**	**Pressure (gauge)**

Legacy models using these blocks are not affected by this change. If a block had an initialization parameter, then, once you open the model in the current release, this parameter value is automatically mapped to the corresponding block variable, which assumes it as the target value with `High` priority. The simulation results stay the same.

However, if you have a custom composite component (written in Simscape language) that uses one of these Foundation library blocks and references an initialization parameter, trying to build the library or simulate a model containing the custom block produces an error, because the referenced parameter is no longer available.

For example, if you have a custom composite component that contains an Inertia block:

```
component DC_Motor
[...]
  parameters
     motor_inertia = { 0.01, 'g*cm^2' };       % Inertia
     init_velocity = { 0, 'rad/s' };           % Initial velocity
     [...]
  end
  components(Hidden=true)
     motorInertia = foundation.mechanical.rotational.inertia(inertia = motor_inertia,
                    initial_velocity = init_velocity);
     [...]
  end
[...]
end
```

`ssc_build` will produce an error similar to the following:

```
Error using ne_updatelibraryitem>lBuild (line 35)     File: C:\Work\libraries\+MyElecLibrary\DC_Motor.ssc
Line: 32 Reference to parameter     'initial_velocity' is invalid.
```

Update the custom component by removing all references to the initialization parameter:

```
component DC_Motor
[...]
  parameters
     motor_inertia = { 0.01, 'g*cm^2' };       % Inertia
     [...]
  end
  components(Hidden=true)
     motorInertia = foundation.mechanical.rotational.inertia(inertia = motor_inertia);
     [...]
  end
[...]
end
```

Variable Viewer for analyzing results of new initialization process

A new analysis tool, available for models containing Simscape blocks and blocks from add-on products, provides the back end for the block-level variable initialization by letting you view the variable targets, priority, and actual initial values prior to simulation. To open the tool, in the top menu bar of the model window, select **Analysis** > **Simscape** > **Variable Viewer**. For more information, see Variable Viewer.

Statistics Viewer that displays variable source and number of eliminated variables

The Statistics Viewer analysis tool has the following enhancements:

- New top-level statistic, **1-D/3-D Interface**, lists the connections between the two types of physical networks. It appears only for models that connect blocks from SimMechanics Second Generation library to Simscape blocks, or blocks from other add-on products.
- The new **Sources** section lists variable sources for the selected statistic. If you select a connection under the **1-D/3-D Interface** statistic category, the **Sources** section lists the source and destination for this connection. If you select a statistic with a nonzero value under the **1-D Physical System** category, the **Sources** section lists all the variables that fall under this statistic.

  For each variable, the **Source** column contains the full path to the variable, starting from the top-level model, with a link to the relevant block. If you click the link in the **Source** column, the corresponding block is highlighted in the block diagram. The **Value** column contains the name of the variable, as it would appear in the **Variables** tab of the block dialog box.
- Additional statistics under the **1-D Physical System** category: **Number of eliminated variables** (further categorized as algebraic and differential variables) and **Number of dynamic variable constraints**. Eliminated variables are continuous variables that are eliminated during optimization and are not seen by the solver. Dynamic variable constraints are constraints involving only dynamic variables and inputs. Such constraints result in Index-2 differential algebraic equations and therefore can cause numerical difficulties or slow down your simulation.

For more information, see Model Statistics.

Fundamental Reluctance block

The new Fundamental Reluctance block in the Magnetic Elements library provides a simplified model of a magnetic reluctance, that is, a component that resists flux flow. Unlike the Reluctance block, which computes reluctance based on the geometry of the section being modeled, the Fundamental Reluctance block lets you specify a value of reluctance directly as a block parameter.

Hydro-mechanical converter blocks with fluid compressibility option

The Rotational Hydro-Mechanical Converter and Translational Hydro-Mechanical Converter blocks now contain a drop-down **Compressibility** parameter, with the default value `Off`. If you select `On`, additional parameters appear in the block dialog to let you account for fluid compressibility within the converter block itself. In previous releases, you had to connect a converter to a Hydraulic Piston Chamber block to account for fluid compressibility.

Compatibility Considerations

Existing models are not affected by this change, because compressibility in converters is off by default. However, this change makes the Hydraulic Piston Chamber block obsolete. It is recommended that you specify fluid compressibility directly in the converter blocks, because the new method provides more accurate results and also because the Hydraulic Piston Chamber block may be removed in a future release.

Handling of pressure or temperature below absolute zero during simulation

You can now set the models that use the hydraulic domain to either warn or stop simulating with an error when absolute pressures fall below absolute zero. The default behavior is to stop simulating with an error, which is the same as in previous releases. You can change this by using the Custom Hydraulic Fluid block, or the Hydraulic Fluid block (available with SimHydraulics® block libraries), to have the simulation continue with a warning. See the block reference pages for details.

You can also set the models that use the pneumatic domain to either warn or stop simulating with an error when pressures or temperatures fall below absolute zero. The default behavior is to stop simulating with an error. This check was not performed during simulation in previous releases. You can change the default behavior by using the Gas Properties block, to have the simulation continue with a warning. See the block reference page for details.

Input filtering options for 1-D/3-D connections

The Solver Configuration block now lets you control whether input filtering is applied automatically for models that connect blocks from SimMechanics Second Generation library to Simscape blocks, or blocks from other add-on products. It also lets you specify a global filtering time constant value for all the 1-D/3-D connections within the network. See the block reference page for details.

Software-in-the-loop simulation for physical models

In previous releases, software-in-the-loop (SIL) simulation was not supported for models containing Simscape blocks or blocks from the add-on products. This limitation is now removed.

Change in default settings for ssc_new

The `ssc_new` function, which creates a new Simscape model populated by required and commonly used blocks, now uses a different default solver and absolute tolerance. Here is the summary of changes:

	Old setting	**New setting**
Solver	`ode15s (stiff/NDF)`	`ode23t (mod. stiff/ Trapezoidal)`
Absolute tolerance	`auto`	`1e-3`

Existing models are not affected. When you create new models with `ssc_new`, they will use the new settings. For more information, see the `ssc_new` reference page.

Functionality being removed or changed

Simscape Language Keyword Name	**What Happens When You Use the Keyword?**	**Use This Instead**	**Compatibility Considerations**
`across`	Still runs	Create equivalent equations	Run `ssc_update` to update the components in an existing package. See “Script for replacing through and across statements with branches and equations” on page 19-3.
`through`	Still runs	`branches`	Run `ssc_update` to update the components in an existing package. See “Script for replacing through and across statements with branches and equations” on page 19-3.

R2013b

Simscape Language

branches section for defining the relationship between component Through variables and nodes

In previous releases, the Through and Across variables in a component file were connected to the domain Through and Across variables in the `setup` section, using the `through` and `across` statements. The syntax of these statements was nonintuitive and somewhat confusing.

Starting in this release, define the relationship between the Through variables by using the new `branches` section, located after the `setup` section in the component file. The `branches` section starts with the `branches` keyword, contains one or more branch statements, and ends with the `end` keyword.

Each branch statement has the syntax:

```
a : node1.a -> node2.a;
```

which clearly indicates direction, from `node1` to `node2`. Therefore, `a` is subtracted from the conserving equation identified by `node1.a`, and `a` is added to the conserving equation identified by `node2.a`. If the component has multiple nodes, indicate branches by writing multiple statements in the `branches` section. For syntax and examples, see the `branches` reference page.

To establish the relationship between the Across variables, use the `equations` section of the component file. Add an equation that connects the component Across variable with the respective variables at the component nodes. If there is more than one Across variable, add multiple equations, connecting each variable with its respective nodes.

For more information, see Defining Relationship Between Component Variables and Nodes.

Compatibility Considerations

You do not currently need to update the existing component files. The `across` and `through` statements compile without warnings. In a future release, they will start to produce warnings when you attempt to build the component. Eventually they will be removed. When writing new component files, use the new syntax.

If you want to update your existing component files, the following table summarizes the old and new syntax.

Old Syntax	New Syntax
`through( a, node1.a, node2.a );`	`a : node1.a -> node2.a ;`
`across( a, node1.a, node2.a );`	`a == node1.a - node2.a ;`

For example, suppose you have an electrical component that uses the old syntax:

```
component my_resistor
  nodes
    p = foundation.electrical.electrical;
    n = foundation.electrical.electrical;
  end
  variables
```

```
    i = { 0, 'A' };
    v = { 0, 'V' };
  end
  parameters
    R = { 1, 'Ohm' };   % Resistance
  end
  function setup
    across( v, p.v, n.v );  % voltage across
    through( i, p.i, n.i ); % current through
  end
  equations
    v == i*R;
  end
end
```

Here is the same component rewritten using the new syntax:

```
component my_resistor
  nodes
    p = foundation.electrical.electrical;
    n = foundation.electrical.electrical;
  end
  variables
    i = { 0, 'A' };
    v = { 0, 'V' };
  end
  parameters
    R = { 1, 'Ohm' };  % Resistance
  end
  branches
    i : p.i -> n.i ;   % current through
  end
  equations
    v == p.v - n.v;    % voltage across
    v == i*R;
  end
end
```

import statement enabling simplified access to other component classes

With the introduction of composite components in R2012b, class member declarations now include user-defined types, that is, component classes. An import mechanism provides a convenient means to accessing classes defined in different scopes, with the following benefits:

- Allows access to model class names defined in other scopes without a fully qualified reference
- Provides a simple and explicit view of dependencies on other packages

The `import` statement can have the following syntax:

```
import package_or_class;
```

or

```
import package.*
```

The first syntax is a qualified import, which imports a specific package or class. The second one is an unqualified import, which imports all subpackages and classes under the specified package.

For more information, see Importing Domain and Component Classes.

connect statement support for vector and matrix physical signals

Simscape language now supports nonscalar (vector-valued or matrix-valued) physical signals in `connect` statements. For more information, see Nonscalar Physical Signal Connections.

Foundation Library and Simulation

Thermal Liquid domain and block library

The Foundation library now contains a thermal liquid domain and Thermal Liquid block library. This library contains thermohydraulic elements, such as chambers, reservoirs, local restrictions, and hydro-mechanical converters. It also contains thermal liquid sources and sensors, as well as a Thermal Liquid Settings (TL) block, which controls thermal liquid domain properties for the attached circuit.

Use these blocks for modeling applications such as:

- Transportation of heated liquid in pipeline networks
- Actuator warm-up due to viscous stresses
- Heat generation and dissipation in complex systems, such as aircraft hydraulic systems and associated heat exchangers

For more information, see the block reference pages. See also Thermal Liquid Domain for information on the thermal liquid domain definition. The Across variables are pressure and temperature, and the Through variables are mass flow rate and thermal flux. Note that the product of each pair of the Through and Across variables (pressure and mass flow rate, temperature and thermal flux) is not power, and therefore these result in a pseudo-bond graph.

Simscape model statistics viewer

A new analysis tool, available for models containing Simscape blocks and blocks from add-on products, lets you view Simscape statistics, such as the number of continuous and discrete variables, number of zero-crossing signals, and number of joints and constraints. To open the tool, in the top menu bar of the model window, select **Analysis** > **Simscape** > **Statistics Viewer**.

For more information, see Simscape Model Statistics.

Removal of laminar-turbulent zero-crossings in hydraulic blocks

Hydraulic blocks in the Foundation library no longer produce zero-crossings upon transition between the laminar and turbulent regimes during simulation. This enhancement results in increased simulation efficiency for hydraulic models.

New examples

Examples introduced in this version are:

- Hydraulic Fluid Warming due to Losses
- Optimal Pipeline Geometry for Heated Oil Transportation
- Water Hammer Effect

Functionality being removed or changed

Simscape Language Keyword Name	What Happens When You Use the Keyword?	Use This Instead	Compatibility Considerations
`across`	Still runs	Create equivalent equations	See Compatibility Considerations under “branches section for defining the relationship between component Through variables and nodes” on page 20-2.
`through`	Still runs	`branches`	See Compatibility Considerations under “branches section for defining the relationship between component Through variables and nodes” on page 20-2.

R2013a

Simscape Language

Variable-size domain parameters

Variable-size component parameters have been implemented in Release R2012a, to support the `tablelookup` function. You can now declare variable-size domain parameters and propagate them to components. Variable-size parameters are still not allowed in the component `equations` section outside of the `tablelookup` function. For more information, see Using Lookup Tables in Equations and Propagation of Domain Parameters.

Vector and matrix physical signals

Simscape language now supports nonscalar (vector-valued or matrix-valued) physical signals in its `inputs` and `outputs` declarations. All signals in such vector or matrix should have the same units. For example, the following declaration

```
inputs
  I = {zeros(3), 'm/s'}; % :left
end
```

initializes a component input as a 3-by-3 matrix of linear velocities.

Simulink-PS Converter and PS-Simulink Converter blocks have also been enhanced to handle vector and matrix physical signals.

Foundation Library and Simulation

Random Number and Uniform Random Number blocks

Two new Physical Signal blocks have been added to the Sources library:

- The Random Number block generates normally (Gaussian) distributed random numbers.
- The Uniform Random Number block generates uniformly distributed random numbers.

The block behavior is the same as that of the respective Simulink blocks, except that they generate a physical signal rather than a unitless Simulink signal.

Perfect Insulator block for thermal domain

The new Perfect Insulator block in the Thermal Elements library models a thermal element with no thermal mass and perfect insulation. Use this block as an insulation for thermal ports to prevent heat exchange with the environment and to model an adiabatic process.

Initialization and diagnostic improvements

When you use local solver in a model, an alternative initialization technique is automatically employed if the standard initialization techniques fail. This alternative technique attempts to find consistent states, within numerical tolerance, by taking a small finite step from the user-specified initial states. Therefore, if the alternative technique succeeds, a warning is issued to the command line that user-specified initial conditions may not be satisfied. Employing this alternative technique increases the initialization robustness, especially when there are nonlinear constraints among dynamic states of a model.

The missing reference node diagnostics have been improved to include information about the particular block and variable that needs a reference node. This is especially helpful when multiple domains are involved in the model.

Model Advisor checks for outdated blocks and for physical unit consistency

Model Advisor user interface now contains two checks specific to Simscape models:

- **Check consistency of block parameter units** notifies you about differences between the declared and the actual settings of block parameter units. The check is triggered by default when you run Model Advisor on your model. You can also run this check individually by selecting **By Product | Simscape** or **By Task | Modeling Physical Systems**. The check detects block parameters in which the specified unit is not directly convertible into the default unit expected by the block. For example, it alerts you if a block parameter is declared with the unit of `rad/s` but the value specified in your pre-R2013a model is in `Hz`. This situation can be problematic because of the new unit definition for Hz in R2013a (see “Unit definition of Hz now consistent with SI” on page 21-4).

  After you run the check, a table of results appears in the right pane of the Model Advisor window. Each cell in the first column of the table contains a link to the problematic block, and the corresponding cell in the second column contains the name of parameter in question, the expected unit, and the specified unit.

Clicking on a link highlights the corresponding block in the model. Double-click the highlighted block, verify the parameter unit setting and correct it, if necessary. Then save and reload the model.

- **Check for outdated Simscape blocks** detects a pre-R2013a version of AC Voltage Source and AC Current Source blocks in your model. The check is triggered by default when you run Model Advisor. You can also run this check individually by selecting **By Product | Simscape**.

  After you run the check, a list of links to the outdated blocks appears in the right pane of the Model Advisor window. Clicking on a link highlights the corresponding block in the model.

  To update the blocks, scroll down the right pane of the Model Advisor window and click the **Update** button.

  - If the automatic update is successful, the Results box displays a message that all blocks have been updated to the current Simscape version.
  - If the message says that some of the blocks could not be updated automatically, rerun the check and manually replace the outdated blocks with the latest version from the block library.

Unit definition of Hz now consistent with SI

The unit definition for Hz is now `1/s`, in compliance with the SI unit system. In previous releases, the unit definition for Hz was `rev/s`, consistent with the definition of frequency as cycles per second in an electrical context, or revolutions per second in a mechanical context. The old unit definition allowed you to specify frequency in angular units (such as `rad/s` or `rpm`) and write frequency-dependent equations without requiring the `2*pi` conversion factor. The main reason for changing the Hz unit definition is to give the block author responsibility for frequency units and their correct interpretation for that block. While a sinusoidal source might reasonably be given a frequency in units of Hz or rpm, angular frequency has no relevance when frequency refers to a nonrotational periodic signal such as the frequency of a PWM source.

As a result of the new unit definition for Hz, frequency units and angular velocity units are no longer directly convertible, and using one instead of the other may result in unexpected conversion factors applied to the numerical values by the block equations.

Drop-down lists of suggested units in block dialogs have been updated to reflect this change. For example, if your block has a **Frequency** parameter with the default unit of `Hz`, the drop-down list for this parameter now contains only units directly convertible to Hz (such as `kHz`, `MHz` and `GHz`) and does not contain the angular velocity units (such as `rpm`, `deg/s` and `rad/s`). You can still type a unit expression representing angular frequency into the units combo box, and the block will accept it as commensurate with the expected parameter unit, but it is your responsibility to make sure that the specified unit works correctly with the block equations.

For more information, see Units for Angular Velocity and Frequency.

Compatibility Considerations

Two Foundation library blocks, AC Current Source and AC Voltage Source, have been affected by this change. In previous releases, you could specify the **Frequency** parameter for these blocks either in units of `Hz` or in angular units, such as `rad/s` or `rpm`. Starting with Release R2013a, you must specify the **Frequency** parameter in units of `Hz` or directly convertible to Hz (such as `1/s`, `kHz`, `MHz` and `GHz`) because the internal equation of the block now uses the `2*pi` conversion factor to account for the `1/s` unit definition.

If you have a pre-R2013a model that contains these blocks, update it by running the Model Advisor check, **Check for outdated Simscape blocks**, or by using the `slupdate` utility, and then save the model. A related Model Advisor check, **Check consistency of block parameter units**, notifies you about differences between the declared and the actual settings of block parameter units. For more information, see "Model Advisor checks for outdated blocks and for physical unit consistency" on page 21-3.

If you have custom Simscape libraries written in R2012b or earlier, and you have used `Hz`, `kHz`, `MHz`, and `GHz` as parameter units, then you will need to update your Simscape code to take account of the Hz unit change. Previously the Simscape unit manager automatically converted any value entered in units of `Hz` into units of `rad/s` before computation. Therefore, now you need to introduce a factor of `2*pi` into block equations to convert to `rad/s` and retain the old functionality.

New format for saving simulation data log objects

Data logging functionality lets you log simulation data to the workspace, in the form of a workspace variable. As with any workspace variable, you can save the data log to a MAT-file, and then load the file into workspace at a later date to query and analyze the data.

Starting with Release R2013a, a new format for saving the data log objects in a MAT-file has been introduced. The new format reduces the disk space usage and memory consumption, and makes it faster to save and load simulation data logs.

Compatibility Considerations

There is no backward incompatibility. That is, you can load previously saved MAT-files containing simulation data with no restrictions.

However, there is a forward incompatibility. MAT-files containing simulation data log objects saved in the new format (starting with Release R2013a) cannot be opened in older versions of MATLAB software (Release R2012b or earlier).

New examples

The following example has been introduced in this version:

- Engine Cooling System

Functionality being removed or changed

Simscape Language Keyword Name	What Happens When you use the Keyword?	Use This Instead	Compatibility Considerations
`throughs`	The change was introduced in Release R2009a. The keyword has now been removed, and using the old syntax produces an error, instead of a warning, when you attempt to build the component.	`variables(Balancing=true)`	See “Simscape Language Syntax Changes” on page 29-3 in Release R2009a.

R2012b

Simscape Language

Connection of components within a Simscape file

In physical modeling, there are two types of models:

- Behavioral — A model that is implemented based on its physical behavior, described by a system of mathematical equations. An example of a behavioral block implementation is the Variable Orifice block.
- Composite — A model that is constructed out of other blocks, connected in a certain way. An example of a composite, or structural, block implementation is the 4-Way Directional Valve block (available with SimHydraulics block libraries), which is constructed based on four Variable Orifice blocks.

In previous versions, Simscape language supported only behavioral models, that is, models defined by equations. To create a model containing multiple interconnected components, you had to define each component in a separate file, deploy it as a custom block, and then use masked subsystems in block diagrams to connect these blocks into a single composite model.

Now, additional language constructs let you create composite models directly in a Simscape file. A component file may now contain two additional blocks:

- A `components` declaration block, which begins with a `components` keyword and is terminated by an `end` keyword. This block contains declarations for all the constituent components. Each component is defined with its full path to the top-level package directory. Specify the required component parameters by declaring a corresponding parameter in the top-level `parameters` declaration block, and then passing this value on to the constituent component.

  For example, the following code includes a Foundation library Resistor block in your custom component file, with a default resistance of 10 Ohm:

  ```
  component MyCompositeModel1
  ...
    parameters
       p1 = {10, 'Ohm'};
       ...
    end
    components(Hidden=true)
       r1 = foundation.electrical.elements.resistor(R=p1);
       ...
    end
  ...
  end
  ```
- A `connections` block, located after the `setup` section, which begins with a `connections` keyword and is terminated by an `end` keyword. This block contains information on how the constituent components' ports are connected to one another, and to the external inputs, outputs, and nodes of the top-level component.

  For example, the following code includes the Foundation library Voltage Sensor and Electrical Reference blocks in your custom component file, connects the negative port of the voltage sensor to ground, and connects the physical signal output port of the voltage sensor to the external output of the composite component, located on the right side of the resulting block icon:

```
component MyCompositeModel2
...
  outputs
     Out = { 0.0, 'V' }; % V:right
     ...
  end
  components(Hidden=true)
     VoltSensor = foundation.electrical.sensors.voltage;
     Grnd = foundation.electrical.elements.reference;
     ...
  end
  connections
     connect(Grnd.V, VoltSensor.n);
     connect(VoltSensor.V, Out);
     ...
  end
end
```

For more information, see Creating Composite Components.

floor, ceil, fix, and round functions

You can now use the following MATLAB functions in the `equations` section of the Simscape file:

- `floor` performs rounding toward negative infinity.
- `ceil` performs rounding toward positive infinity.
- `fix` performs rounding toward zero.
- `round` performs rounding toward the nearest integer.

The PS Floor, PS Ceil, and PS Fix blocks in the Physical Signals/Nonlinear Operators sublibrary of the Foundation library have been reimplemented using the Simscape language and the corresponding function. The PS Round block has been added to the Nonlinear Operators sublibrary. It performs rounding toward the nearest integer.

Foundation Library and Simulation

Speed and efficiency improvements for simulation of switched linear systems

Switched linear systems are systems that have multiple configurations during simulation, but each configuration is linear. The changes in configuration during simulation may be due to deployment of switches or other elements. A new specialized simulator, implemented for switched linear systems, reduces the number of states and accelerates simulation. The specialized simulator is automatically employed based on the system structure; you do not have to select or enable it explicitly.

Zero-crossing statistics for Simscape logging

If you log simulation data for a Simscape model, you now have an option to log simulation statistics, including zero-crossing data. By default, the zero-crossing data is not logged. If you select the **Log simulation statistics** check box on the **Simscape** pane of the Configuration Parameters dialog box, the simulation log variable contains an additional `SimulationStatistics` node for each block that can produce zero crossings. You can then plot and analyze this data similar to other data logged to the workspace during simulation.

Counter and Repeating Sequence blocks that facilitate discrete sampling

Two new blocks in the Physical Signals/Sources library facilitate discrete sampling in physical modeling:

- The Counter block repeatedly increments the output signal by 1 with every time step, in the range between the minimum (reset) value and the maximum value. You can optionally specify an initial signal value, different from the reset value, and an initial time offset. Use this block, in conjunction with other physical signal blocks, to model discrete behaviors.
- The Repeating Sequence block outputs a periodic piecewise-linear signal. You can optionally specify an initial signal value and an initial time offset. The repeating sequence consists of a number of linear segments, connected to each other. Use this block to generate various types of physical signals, such as pulse, sawtooth, stair, and so on.

Open-circuit terminator blocks for electrical, hydraulic, and mechanical domains in Foundation library

Physical Network block diagrams do not allow unconnected Conserving ports. Previously, if you wanted to leave a port unconnected (open-circuit) you had to add an extra sensor, which cluttered the block diagram.

Now the following blocks represent domain-specific open-circuit terminators:

- The Open Circuit block represents an electrical terminal that draws no current. Use this block to terminate electrical ports that you want to leave open-circuit.
- The Hydraulic Cap block represents a hydraulic plug, that is, a hydraulic port with zero flow through it. Use this block to terminate hydraulic ports that you want to cap.

- The Rotational Free End block represents a mechanical rotational port that rotates freely, without torque. Use this block to terminate mechanical rotational ports that you want to leave unconnected.
- The Translational Free End block represents a mechanical translational port that moves freely, without force. Use this block to terminate mechanical translational ports that you want to leave unconnected.

Viewable and customizable source files for additional Foundation library blocks

In R2009a, many blocks in the Foundation library were implemented using the Simscape language. In 2012b, most of the remaining blocks have been converted.

You can now view the source files for most of the Foundation library blocks. When you open the block dialog box, it contains a link:

```
View source for BlockName
```

Click this link to open the Simscape source file for the block in the MATLAB Editor. To customize the block for your application, edit the source file and save it in a package directory. Some of the features, such as drop-down lists in block dialog boxes, are not yet available for custom blocks. For more information on packaging Simscape source files, see Simscape File Deployment.

Compatibility Considerations

The PS Math Function block now issues a simulation-time error when the input falls out of the expected domain for the particular function used. For example, if set to `sqrt`, the PS Math Function block now issues an error if it receives negative input during simulation.

New examples

Examples introduced in this version are:

- Lithium Battery Cell - One RC-Branch Equivalent Circuit
- Lithium Battery Cell - Two RC-Branch Equivalent Circuit
- Asynchronous PWM Voltage Source
- Discrete-Time PWM Voltage Source

R2012a

Modeling Delays in Simscape Language

The new `delay` construct in Simscape language lets you refer to past values of expressions in the `equations` section of the Simscape file. For more information, see the delay reference page.

New Blocks for Modeling Delays

The new Delays sublibrary of the Physical Signals library contains two blocks:

- PS Constant Delay block returns the input signal delayed by a specified time, which is constant throughout the simulation. You specify the delay time as a block parameter.
- PS Variable Delay block delays the input signal by a variable time. You specify the delay time and the input history as additional inputs.

Use these blocks to model idealized transport phenomena at system level.

1D and 2D Interpolation Available in Simscape Language

The new `tablelookup` function in Simscape language lets you interpolate expressions in the `equations` section of the Simscape file. It computes an output value by interpolating the input value against a set of data points in a one-dimensional or two-dimensional table, and supports three interpolation and two extrapolation options. This functionality is similar to that of the Simulink and Simscape Lookup Table blocks. It allows you to incorporate table-driven modeling directly in your custom block, without the need of connecting an external Lookup Table block to your model. For more information, see the tablelookup reference page.

The PS Lookup Table (1D) and PS Lookup Table (2D) blocks in the Foundation library now use the `tablelookup` function in the Simscape language. The block functionality and user interface remain the same.

Input Filtering Usability Enhancements

In previous releases, input filtering was automatically turned on whenever you used an explicit solver in a Simscape model. To turn off input filtering when using an explicit solver, you had to supply first derivative of the input signal as an additional input signal to the Simulink-PS Converter block. For models using other types of solvers, input filtering was not available.

Now input filtering is independent of the solver used in the model. You can control whether you filter input or provide time derivatives for each input signal individually, by configuring the Simulink-PS Converter block connected to that input signal. You can:

- Set the **Filtering and derivatives** parameter to `Filter input`, and select whether you want to use the first-order or second-order filter. Input filtering makes the input signal smoother and generally improves model performance. The additional benefit is that the Simscape engine computes the time derivatives of the filtered input. The first-order filter provides one derivative, while the second-order filter provides the first and second derivatives. If you use input filtering, it is very important to select the appropriate value for the filter time constant.
- Set the **Filtering and derivatives** parameter to `Provide input derivative(s)`, and provide either just the first time derivative, or the first and the second time derivatives, through additional input ports on the Simulink-PS Converter block.

By default, input signals are used as is, without performing input filtering or otherwise providing time derivatives of the input signal. If you use an explicit solver, It is recommended that you provide input derivatives by selecting one of the options listed above. If you do not provide input derivatives, and the solver you use requires them, you get an error message indicating how many input derivatives you need to provide. For more information, see Harmonizing Simulink and Simscape Solvers and the Simulink-PS Converter block reference page.

Compatibility Considerations

Input filtering is no longer automatically turned on for models using explicit solvers. Therefore if you have an existing model that uses an explicit solver, its performance and simulation results may be different from the previous version.

Because Simscape solver no longer automatically provides the required input derivatives, you may get an error message indicating how many input derivatives you need to provide.

To preserve the old behavior, open each Simulink-PS Converter block and set the **Filtering and derivatives** parameter to `Filter input`, while keeping the value of the **Input filtering time constant** parameter unchanged. If any of the Simulink-PS Converter blocks in your model had input derivatives provided as additional signals, set the **Filtering and derivatives** parameter for such block to `Provide input derivative(s)`.

Zero Damping Allowed for Resistive Elements

In previous releases, resistive blocks in the Foundation library required positive damping coefficients. Negative damping values are nonphysical. However, zero damping values are useful for model checking and testing, For example, if you use a Rotational Damper block as part of a customized gear model, it is convenient to be able to set the **Damping coefficient** parameter to `0` temporarily, to compute undamped responses.

The following blocks now allow zero damping values, to support model testing:

- Resistor (Electrical Elements library)
- Linear Hydraulic Resistance (Hydraulic Elements library)
- Rotational Damper (Rotational Elements library)
- Translational Damper (Translational Elements library)

Changes to Simscape Demos

The following demo has been added in Version 3.7:

Demo Name	Description

Variable Transport Delay (`ssc_transport_delay`)	Provides an example of modeling a variable transport delay using Simscape language. The Transport Delay subsystem models signal propagation through media moving between the Input and the Output terminals. The media velocity may vary, thus it is specified through the block port. The distance between the terminals is constant and it is specified as a block parameter. To see the implementation details, look under mask of the Transport Delay subsystem, then right-click the Variable Transport Delay block and select **View Simscape source**.

R2011b

Simscape Language Enables User-Defined Diagnostics During Simulation

The new `assert` construct in Simscape language lets you implement run-time error messages when the custom block is used in a model. In the component file, you specify the condition to be evaluated, as well as the error message to be output if this condition is violated. When the custom block based on this file is used in a model, it will output this error if the condition is violated during simulation. For more information, see Programming Run-Time Errors and Warnings.

Compatibility Considerations

Run-time error checking has now been implemented in Foundation library, and may affect the behavior of your existing models. For example, in previous releases, `sqrt`(x) for negative numbers was calculated as $\sqrt{|x|} \cdot sign(x)$ (e.g., `sqrt`(–1) = –1). Now it will give a run-time error if the input becomes negative.

Run-time error checks have been implemented for the following functions:

Function	Condition
`sqrt`(*x*)	$x < 0$
`log`(*x*)	$x <= 0$
`log10`(*x*)	$x <= 0$
1 / *x*	*x* == 0, if *x* is an expression involving only constants and parameters

The following Foundation library blocks are affected:

- PS Math Function block — when the inputs of `log`, `log10`, `sqrt`, and `1/u` violate the conditions listed above, the block will now generate a run-time error.
- PS Divide block — when the denominator is zero, the block will now generate a run-time error.

Existing models using these blocks, which ran successfully in previous releases due to lack of error checking, will now generate a run-time error if one of the above conditions is violated.

New Block for Modeling Discrete Delays

The Asynchronous Sample & Hold block, in the new Discrete sublibrary of the Physical Signals library, sets output Y equal to input U when the rising edge of the trigger input becomes greater than zero. Use this block, in conjunction with other physical signal blocks, to model discrete and event-based behaviors.

Specialized Simulator for Linear Systems

The new specialized simulator, implemented for linear systems, reduces number of states and accelerates simulation. The specialized linear simulator is automatically employed based on the system structure, you do not have to select or enable it explicitly.

Rebuilding of Custom Block Libraries Now Required

Due to further scalability improvements, as well as other internal enhancement to facilitate add-on product functionality, you have to rebuild your custom block libraries once you upgrade to Version 3.6

(R2011b). It is required that you rebuild your custom block libraries for use with each new version of Simscape software. For more information, see When to Rebuild the Custom Library.

To rebuild the libraries, run `ssc_build` on the component Simscape files. If you try to use the custom blocks without rebuilding the libraries, you will get an error message.

Running `ssc_clean` before `ssc_build` is strongly recommended but not required. If you run into errors after running `ssc_build`, run `ssc_clean` and then try running `ssc_build` again.

R2011a

Additional Hydraulic Sources

Two new blocks have been added to the Hydraulic Sources library:

- Hydraulic Constant Flow Rate Source block represents an ideal source of hydraulic energy that is powerful enough to maintain specified flow rate at its outlet regardless of the pressure differential across the source. The **Source flow rate** parameter specifies the flow rate through the source.
- Hydraulic Constant Pressure Source block represents an ideal source of hydraulic energy that is powerful enough to maintain the specified pressure differential between its inlet and outlet regardless of the flow rate through the source. The **Pressure** parameter specifies the pressure differential across the source.

Use these blocks for models where flow rate or pressure remain constant throughout simulation.

Improved Simulation Diagnostics

The following improvements in simulation diagnostics have been implemented, to aid debugging:

- Equation dependency diagnostics now point to the specific equations (with line number and file location info) within the component Simscape files.
- Equation dependency diagnostics have been extended to include switched-linear and nonlinear equations in the analysis.
- Equation dependency diagnostics are now triggered on nonlinear solver failures that occur after the start of simulation.
- Zero-crossing related warnings and error messages now point to the specific equations (with line number and file location info) within the component Simscape files.

Improved Scalability

Various scalability improvements have been implemented in this release, accelerating simulation of larger systems.

Compatibility Considerations

Due to this change, you have to rebuild your custom block libraries once you upgrade to Version 3.5 (R2011a). To rebuild the libraries, run `ssc_build` on the component Simscape files. If you try to use the custom blocks without rebuilding the libraries, you will get an error message.

Running `ssc_clean` before `ssc_build` is strongly recommended but not required. If you run into errors after running `ssc_build`, run `ssc_clean` and then try running `ssc_build` again.

Improved Algorithms for Algebraic Loop Detection and Zero-Crossing Robustness

The following simulation algorithm improvements have been implemented in this release:

- False algebraic loop detection and prevention—The improved algorithm can now recognize false algebraic loops and prevent them from affecting simulation results.

- Performance and zero-crossing robustness improvements—Zero-crossing detection algorithm has been optimized to ignore zero-crossings that do not result in model behavior changes during simulation.

Change in Evaluating Unit Expressions

In the Unit Manager parser, multiplication (`*`) used to mistakenly have higher precedence than division (`/`). This issue is now fixed. When evaluating unit expressions, `*` and `/` now have the same precedence and are evaluated based on left associativity.

Compatibility Considerations

Due to this change, custom units specified in the unit registry may now evaluate differently. For example, if you have added a unit `m/s*s`, in previous releases it was evaluated as `m/(s*s) = m/s^2`. It will now evaluate to `(m/s)*s = m`. You can use parentheses to preserve the old behavior.

R2010b

Simulation Data Logging Enhancements

The following data logging enhancements have been implemented in this version:

- `plot` and `plotxy` commands have been added. The `plot` command lets you plot logged data against time, while the `plotxy` command plots two sets of data against each other. Also, `plot` and `plotxy` methods are now available for `simscape.logging.Node` and `simscape.logging.Series` objects. For more information, see the respective reference pages.
- New configuration parameter, **Decimation**, lets you downsample logged data by skipping time steps. For more information, see Data Logging Options in the *Simscape User's Guide*.
- Simulation data is now logged according to the value specified for the **Output options** parameter in the **Data Import/Export** pane of the Configuration Parameters dialog box.

Compatibility Considerations

For models that do not use the default value of the **Output options** parameter in the **Data Import/Export** pane of the Configuration Parameters dialog box, logged simulation data may change compared to the previous release.

Zero-Crossing Handling Enhancements

The creation and detection of zero-crossing conditions in Simscape models have been improved.

New Implementation of Relational Functions Without Creating Simulink Zero-Crossing Conditions

The Simscape language now implements relational functions in a different way from how relational operators are implemented. Unlike relational operators, these relational functions now do not create Simulink zero-crossing conditions.

For more information about zero crossings in Simscape models, see Creating and Detecting Zero Crossings in Simscape Models in the Simscape User's Guide.

For more information about creating models in the Simscape language, see the Simscape Language Guide and the `equations` syntax.

Improved Zero-Crossing Detection Diagnostics

If your model generates a Simulink warning or error about zero crossings involving Simscape blocks, the warning or error message now specifies which blocks are generating this zero-crossing diagnostic message.

If you globally disable zero-crossing detection in a Simulink model containing Simscape blocks, and if you are using a variable-step solver without a local solver, you now receive a diagnostic warning or error. You can control which message that you receive in the **Simscape** pane of the model Configuration Parameters dialog box, through the **Zero-crossing control is globally disabled in Simulink** drop-down list. This option supports context-sensitive or “What's This?” help, by default accessed through right-clicking the item.

See Harmonizing Simulink and Simscape Solvers in the Simscape User's Guide.

C++ Code Generation Support

You can now generate C++ code from Simscape models. Encapsulated C++ code generation is not supported.

For more information, see Code Generation and Limitations in the Simscape User's Guide.

Sparse Solver Enhancement

The implementation and control of matrix linear algebra in the Solver Configuration block have been improved and simplified. Whether you choose the sparse solver or the full solver, your linear algebra choice is now implemented in both model simulation and code generated from the model.

For more information, see the Solver Configuration block reference page.

Component Descriptor Is No Longer Inherited from the Base Class

In Simscape Language, the name of the block built from a component file generally corresponds to the component file name. You can provide a more descriptive name for the block by adding a comment line immediately following the component declaration. This comment line is called the descriptor. For more information, see How to Customize the Block Name.

If you use subclassing, the subclass inherits all of the members (parameters, variables, nodes, inputs and outputs) from the base class (for more information, see Subclassing and Inheritance). In previous releases, descriptor was one of the inherited properties. That is, if you had a subclass component without a descriptor, the resulting block name was derived from the descriptor of the base class, instead of the component file name. This could create issues with library building if both the subclass component and the base class component were in the same sublibrary.

Starting with this release, descriptor is no longer inherited from the base class. In other words, if a component file does not contain a comment line immediately following the component declaration, the component name is always used for the block name.

Compatibility Considerations

For existing subclass components, if the component does not have a descriptor line, upon rebuilding the library the block name will change compared to the previous release. To preserve the old name, add a descriptor line to the subclass component file.

Documentation Enhancements

The Model Simulation chapter has been expanded and improved with revised and new sections, including:

- How Simscape Models Represent Physical Systems
- How Simscape Simulation Works
- Setting Up Solvers for Physical Models
- Customizing Solvers for Physical Models
- Real-Time Simulation

- Finding an Operating Point
- Linearizing at an Operating Point
- Finding and Using Operating Points in an Electronic Circuit

Changes to Simscape Demos

The following demo has been added in Version 3.4:

Demo Name	Description
Electrical Transformer (`ssc_transformer`)	Presents a view inside a transformer core using the electromagnetic blocks from the Magnetic block library.

R2010a

Magnetic Blocks Added to Foundation Library

Foundation library now contains magnetic domain and Magnetic block library. This library contains electromagnetic elements, such as reluctances, actuators, and electromagnetic converters, as well as magnetic sensors and sources.

Use these blocks to model magnetic circuits that can be represented by a one-dimensional flux flow, for example, solenoids and transformers.

Magnetic block models are based on the following assumptions:

- The magnetic system is assumed lossless. You can model losses in interconnecting systems instead – in electric systems using resistors and in mechanical systems using friction.
- Modeling of superconductors (with zero relative permeability) is not supported.

For more information see the block reference pages. See also Magnetic Domain for information on the magnetic domain definition. The Across variable is magnetomotive force (mmf), and the Through variable is flux. Note that these result in a pseudo-bond graph, because the product of mmf and flux is energy, not power.

Simulation Data Logging Now Available

You can now log simulation data to workspace for debugging and verification purposes. Data logging lets you analyze how internal block variables change with time during simulation. For example, you may want to see that the pressure in a hydraulic cylinder is above some minimum value, or compare it against the pump pressure. If you log simulation data to workspace, you can later query, plot, and analyze it without rerunning the simulation. For more information, see Data Logging in the *Simscape User's Guide*.

Simscape Diagnostics Improvements

Simscape error messages triggered by initial dynamic state inconsistencies and by nonlinear solver convergence failures are now more detailed. These messages report specific components of your models that may have caused the error.

Sparse Jacobian Support

You can now use a Jacobian method with an implicit Simulink solver in your Simscape models. You can choose a method yourself or allow Simulink to determine an appropriate Jacobian method for you. Depending on the sparsity pattern and number of states of your model, your simulation may be more efficient.

See Choosing a Jacobian Method for an Implicit Solver in the Simulink documentation.

Ability to Generate Simscape Language Equations from Symbolic Expressions

If you have Symbolic Math Toolbox™ software, you can use the `simscapeEquation` function to generate Simscape language equations from symbolic expressions. For more information, see Generating Simscape Equations in the Symbolic Math Toolbox documentation.

Placing Simscape Blocks in Nonvirtual Subsystems

Nonvirtual subsystems that support continuous states include Enabled subsystems and Atomic subsystems. These subsystems can contain Simscape blocks. However, physical connections and physical signals must not cross nonvirtual boundaries. For more information, see Restricted Simulink Tools in the *Simscape User's Guide*.

Compatibility Considerations

Simscape solver no longer permits physical connections and physical signals to cross nonvirtual subsystem boundaries, because the semantics of these types of connections are unclear. If either a physical signal or a physical connection crosses a nonvirtual boundary, the solver issues an error upon simulation. To resolve the issue, place all blocks belonging to a given Physical Network in the same nonvirtual subsystem.

Trimming and Linearization Documentation Enhancements

The documentation on Simscape model trimming and linearization has been revised and expanded. See Finding an Operating Point and Linearizing at an Operating Point.

Changes to Simscape Demos

The following demos have been added in Version 3.3:

Demo Name	**Description**
Circuit Breaker (`ssc_circuitbreaker`)	Implements a simple circuit breaker model.
Solenoid with Magnetic Blocks (`ssc_solenoid_magnetic`)	Shows how to model a solenoid using the electromagnetic blocks from the new Magnetic block library.

R2009b

Pneumatic Blocks Added to Foundation Library

Foundation library now contains pneumatic domain and Pneumatic block library. This library contains pneumatic elements, such as orifices, chambers, and pneumatic-mechanical converters, as well as pneumatic sensors and sources.

Use these blocks to model pneumatic systems, for applications such as:

- Factory automation — basic pneumatic linear/rotational actuators, valves (variable orifices), and air supply
- Robotics — robotic arms and haptic interfaces
- Gaseous transportation systems and pipelines

You can also use these blocks to model dry air and low pressure flows, for example, for HVAC applications.

Pneumatic block models are based on the following assumptions:

- Working fluid is an ideal gas satisfying the ideal gas law.
- Specific heats at constant pressure and constant volume, c_p and c_v, are constant.
- Processes are adiabatic, that is, there is no heat transfer between components and the environment (except for components with a separate thermal port).
- Gravitational effects are neglected.

For more information see the block reference pages, as well as Modeling Pneumatic Systems in the *Simscape User's Guide*.

Compatibility Considerations

To avoid duplicate block names in different Simscape domains and increase naming consistency across domains, the following hydraulic blocks have been renamed:

Old Block Name	New Block Name
Constant Area Orifice	Constant Area Hydraulic Orifice
Constant Volume Chamber	Constant Volume Hydraulic Chamber
Piston Chamber	Hydraulic Piston Chamber
Resistive Tube	Hydraulic Resistive Tube
Variable Area Orifice	Variable Area Hydraulic Orifice
Variable Chamber	Variable Hydraulic Chamber
Ideal Hydraulic Flow Rate Sensor	Hydraulic Flow Rate Sensor
Ideal Hydraulic Pressure Sensor	Hydraulic Pressure Sensor
Ideal Hydraulic Flow Rate Source	Hydraulic Flow Rate Source
Ideal Hydraulic Pressure Source	Hydraulic Pressure Source

Old models containing any of these blocks will be updated automatically once you open and save them.

New and Enhanced Switches

The following switching capability enhancements have been implemented in Foundation libraries:

- New Physical Signal PS Switch block has been added to the Nonlinear Operators library. It contains three physical signal input ports, a physical signal output port, and one parameter, **Threshold**. If the second input is greater than or equal to the threshold, then the output is connected to the first input. Otherwise, the output is connected to the third input. The second input never connects to the output.
- Electrical Switch block has been enhanced to use a value specified in the **Threshold** parameter (rather than zero) for opening and closing the switch.

Intermediate Terms in Simscape Language Equations

You can now introduce intermediate terms in Simscape Language equations by using the `let` and `in` keywords. This functionality helps increase the equation readability, as well as avoid duplicating information by defining an intermediate term once and then using it in multiple equations. For more information, see the Simscape Language Guide.

Local Solver Support in Physical Networks

The Solver Configuration block now lets you use sample-based local solver with a specific sample time. In sample-based simulation, all the Physical Network states, otherwise represented as continuous, become discrete states. The solver updates the states once per time step. This option is especially useful for code generation, or hardware-in-the-loop (HIL) simulations. For more information, see the Solver Configuration reference page.

Simulink Manifest Tool Support

Dependency analysis tools for Simscape files have been added in this release. They consist of the following command-line options:

- `simscape.dependency.file` — Perform dependency analysis for a single Simscape file.
- `simscape.dependency.lib` — Perform dependency analysis for a Simscape custom library.
- `simscape.dependency.model` — Perform dependency analysis on a model containing Simscape and Simulink blocks.

Manifest reports generated using Simulink Manifest Tools now also include model dependencies for the Simscape blocks. For more information, see Checking File and Model Dependencies in the *Simscape Language Guide*.

SimState Support

Simscape software now supports Simulink SimState feature, introduced in R2009a. This feature allows you to save all runtime data necessary for restoring the simulation state of a model. For more information, see Saving and Restoring the Simulation State as the SimState in the *Simulink User's Guide*.

Note When using SimState to save and restore simulations of models involving Simscape blocks, please ensure both `'DstWorkSpace'` and `'SrcWorkSpace'` to be `'base'` by using:

```
simset('DstWorkspace', 'base', 'SrcWorkspace', 'base')
```

Model Reference Accelerator Mode Support

Simscape and its add-on products now fully support Model Reference Accelerator Mode, both for model simulation and for code generation.

Physical Port Rotation for Simscape Blocks

When you rotate a regular Simulink block, its ports are by default reordered after rotation, to maintain the left-right and top-down block diagram orientation convention used in control system modeling applications. This convention is not applicable to physical modeling and is potentially confusing, because it results in effectively rotating and flipping the block at the same time.

Therefore, starting with Version 3.2 (R2009b), when you rotate a Simscape block (including blocks from add-on products), its ports are not reordered. This behavior is similar to that of the masked blocks with **Port Rotation** set to `Physical`. For illustration of differences between the default port rotation type and the physical port rotation type, see Changing a Block's Orientation in the *Simulink User's Guide*.

Compatibility Considerations

This change in the behavior of the ports after block rotation may result in visually crossed connection lines in some of your existing block diagrams with rotated blocks. The effect is purely cosmetic and has no impact on actual model connections or simulation.

R2009a

Simscape Language Source Protection

Simscape language files can be protected to enable model sharing without disclosing the component or domain source. You can then share the protected (executable) files without disclosing the file content (similar to P-code vs M-code). While Simscape source files have the extension `.ssc`, Simscape protected files have the extension `.sscp`.

Use the `ssc_protect` command to protect individual files and directories.

Use the `ssc_mirror` command to create a protected copy of a whole package, along with a custom block library built from it.

For more information, see Using Source Protection for Simscape Files.

Expanded MATLAB Support of Simscape Files

MATLAB support of Simscape files has been expanded:

- If you issue the `open` command on a Simscape file, the file will open in the MATLAB Editor. The Simscape file must be on the MATLAB path, or in a package residing in a directory on the MATLAB path. For more information on packaging Simscape files, see Organizing Your Simscape Files.

  If you issue the `open` command on a Simscape protected file (`*.sscp`), the corresponding Simscape source file (`*.ssc`) will open, provided it exists in the same directory as the Simscape protected file.
- Issuing the `help` command on a Simscape file displays the domain or component description, that is, all the comments immediately following the domain or component declaration, in the MATLAB Command Window.
- MATLAB Editor now supports syntax highlighting of Simscape files, similar to M-files. For more information, see Adjust Editor Appearance.

Viewable and Customizable Source Files for Foundation Library Blocks

You can now view the source files for many Foundation library blocks. When you open the block dialog box, it contains a link:

`View source for` *`BlockName`*

Click this link to open the Simscape source file for this block in the MATLAB Editor. To customize the block for your application, edit the source file and save it in a package directory. For more information, see Simscape File Deployment.

Compatibility Considerations

The block source has been optimized, with some previously defined but unused variables eliminated. Therefore, when you load an old model containing Foundation blocks, you might get warnings, for example:

```
Warning: In instantiating linked block 'model/R1' : Resistor block (mask) does not
have a parameter named 'current_Log'.
```

You can safely ignore these warnings. Once you save the model, the warnings will disappear.

Simscape Language Syntax Changes

The following changes have been implemented in Simscape language:

- The `throughs` keyword has been obsoleted. Use `variables(Balancing=true)` to declare Through variables in a domain.
- The `equation` keyword has been changed to `equations`.
- The name of a Simscape file must match the name of the component or domain it defines. If this is not the case, you will get an error when trying to build a library or use the block in a model.

For more information, see the Simscape Language Guide.

Compatibility Considerations

The changes are relatively minor, but may require modifying your existing Simscape files. The following table summarizes the old and new syntax.

Old Syntax	New Syntax
`throughs`	`variables(Balancing=true)`
`equation`	`equations`

Increased Efficiency of Simscape Language Equations Processing

Simscape language equations are now processed more efficiently, reducing the time required to process equations with multiple `if` statements.

New Physical Signal Blocks to Facilitate Rounding

Three new Physical Signal blocks have been added to the Nonlinear Operators library:

- PS Ceil block performs rounding of the signal toward positive infinity
- PS Floor block performs rounding of the signal toward negative infinity
- PS Fix block performs rounding of the signal toward zero

Model Reference Accelerator Mode Support

Simscape and its add-on products now support Model Reference Accelerator Mode for model simulation, but not for code generation. Model Reference Accelerator Mode for code generation is supported only by SimMechanics nd SimDriveline™ software.

Changes to Simscape Demos

The following demo has been added in Version 3.1:

Demo Name	Description

Creating A New Circuit (`ssc_new_elec`)	Use this demo as a template for creating a new electrical model. Open the demo and use **File > Save As** to save it under the desired model name. Then delete the unwanted components and add new ones. This demo also opens an Electrical Starter Palette, which contains links to the most often used electrical components.

R2008b

Simscape Language

New Simscape language extends the Simscape modeling environment by enabling you to create new components that do not exist in the Foundation library or in any of the add-on products. It is a dedicated textual physical systems modeling language with the following characteristics:

- Derives from MATLAB and familiar to those who use MATLAB
- Contains additional constructs specific to physical modeling and excludes constructs that have nothing to do with physical modeling
- Incorporated into the Simscape modeling interface
- Not focused on algorithm development

The Simscape language is intended to make modeling physical systems easy and intuitive. It lets you create new physical domains and components as textual files and then use them in Simscape block diagrams to model the desired physical effects. For more information, see the Simscape Language Guide.

Explicit Solvers

It is now possible to choose any variable-step or fixed-step solver for models containing Simscape blocks. Note, however, that implicit solvers, such as ode14x, ode23t, and ode15s, are still a better choice for a typical model. In particular, for stiff systems, implicit solvers typically take many fewer timesteps than explicit solvers, such as ode45, ode113, and ode1.

By default, you will get a warning when using an explicit solver for a model containing Simscape blocks. For models that are not stiff, however, explicit solvers can be effective, often taking fewer timesteps than implicit solvers. Depending on the type of your model, you can configure your preferences to either turn off this warning (if your model is not stiff) or even change it into an error (to avoid inadvertent use of explicit solvers), by using the **Simscape** pane of the Configuration Parameters dialog box.

If you use an explicit solver, it requires time derivatives of the input signals. By default, needed input derivatives are provided by filtering the input through a low-pass filter. The derivative of the filtered input can then be computed by the Physical Networks simulation engine. The new **Derivatives** tab in the Simulink-PS Converter dialog box lets you turn off input filtering and instead provide the first derivative of input as an additional input signal to the Simulink-PS Converter block. For more information, see the Simulink-PS Converter block reference page.

Because input filtering can appreciably change the input signal and drastically affect simulation results if the time constant is too large, a warning is issued when input filtering is used. The warning indicates which Simulink-PS Converter blocks have their input signals filtered. This warning can also be turned off (or changed to an error) by changing the preferences on the **Simscape** pane of the Configuration Parameters dialog box.

New Ways to Model Variable Chambers

There are now two blocks that let you model fluid compressibility in variable chambers:

- Piston Chamber block lets you model fluid compressibility in a chamber created by the piston in a cylinder. It replaces the Variable Volume Chamber block, available in previous releases.
- Variable Chamber block lets you model fluid compressibility in variable volume chambers of any shape. The instantaneous value of the chamber volume is provided by using a physical signal port.

Compatibility Considerations

The Variable Volume Chamber block, available in previous releases, has been deprecated. It has been replaced by the Piston Chamber block in other (structural) blocks and in demos shipped with the product. If you have used the Variable Volume Chamber block in your models, it will continue to work. Going forward, however, use the Piston Chamber block to model fluid compressibility in cylinder chambers.

Model Reference Support

Simscape software now supports the Simulink model referencing functionality in Normal mode. Other Simulink models can now reference Simscape models in normal (non-code-generation) execution. Simscape models continue to be able to reference Simulink models (that do not contain Simscape blocks) in normal execution. See Limitations for more details.

More Solver Performance and Robustness Enhancements

Version 3.0 contains multiple further enhancements to simulation algorithms, resulting in improved robustness and reliability.

R2008a

Trimming Now Available for Simscape Models

Finding and managing operating points by trimming has been implemented for models that include Simscape and SimHydraulics blocks. Simulink Control Design™ product is required for using this functionality. For more information, see Finding an Operating Point.

Thermal Unit Conversions Now Supported

You can now specify temperature for your thermal models in a variety of units, including degrees Celsius, Fahrenheit, and Rankine. The unit manager automatically handles conversions between thermal units.

Thermal units sometimes require an affine conversion, that is, a conversion that performs both multiplication and addition. In situations when you deal with a relative, rather than absolute, temperature, you need to convert using just the linear term. Thermodynamic variables in block dialogs are automatically tagged as appropriate and handled by the unit manager. However, when an input or output signal is related to thermodynamic variables and contains units of temperature, you must decide whether affine conversion needs to be applied. The Simulink-PS Converter and PS-Simulink Converter block dialogs now contain the **Apply affine conversion** checkbox. If you select it, the unit manager uses the affine conversion, otherwise it applies the default linear conversion.

For more information, see Thermal Unit Conversions, as well as the Simulink-PS Converter and PS-Simulink Converter block reference pages.

The `pm_addunit` command has also been modified to support affine conversions. Its second argument, `conversion`, may now be either a positive real scalar or a 1x2 array. If this argument has two elements, then it is specifying an affine conversion, with the first element (a positive real number) being the linear conversion coefficient, and the second being the offset.

Enhancement to Specifying Units

Simscape block dialogs have drop-down combo boxes for units next to a parameter value. You can either select a unit from the drop-down list, or type a commensurate unit name (or a mathematical expression with unit names) directly into the units combo box of the block dialog. For more information, see How to Specify Units in Block Dialog Boxes.

Similarly, the Simulink-PS Converter and PS-Simulink Converter block dialogs now contain a drop-down list, which is prepopulated with some common input or output units. You can either select a unit from the list or type a unit name, or a mathematical expression with unit names. Note that you must still match the unit type:

- For a PS-Simulink Converter block, these units must be commensurate with the units of the input physical signal coming into the block.
- Signal units that you specify in a Simulink-PS Converter block must match the input type expected by the Simscape block connected to it.

New Blocks

Version 2.1 contains two new blocks:

- Gyrator block in the Electrical Elements library simulates an ideal gyrator, which can be used to implement an inductor with a capacitor.

- PS Abs block in the Physical Signals library returns absolute value of input signal.

Enhancements to Simulation Algorithms

Version 2.1 contains multiple enhancements to simulation algorithms, resulting in improved robustness and reliability.

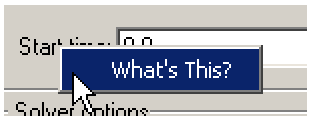
“What’s This?” Context-Sensitive Help Available for Simulink Configuration Parameters Dialog

R2008a introduces “What's This?” context-sensitive help for parameters that appear in the Simulink Configuration Parameters dialog, including those on the **Simscape** pane. This feature provides quick access to a detailed description of the parameters, saving you the time it would take to find the information in the Help Browser.

To use the "What's This?" help, do the following:

1. Place your cursor over the label of a parameter.
2. Right-click. A **What's This?** context menu appears.

   For example, the following figure shows the **What's This?** context menu appearing after a right-click on the **Start time** parameter in the **Solver** pane.

3. Click **What's This?**. A context-sensitive help window appears showing a description of the parameter.

New Simscape Demo

The following demo has been added in Version 2.1:

Demo Name	Description
House Heating System (`ssc_house_heating_system`)	The demo represents a simple house heating system consisting of a heater, thermostat, and a house structure with four thermally distinguishable parts: inside air, house walls, windows, and roof. You can investigate system behavior with the heating system turned on or off, and plot the heat cost and indoor versus outdoor temperatures.

R2007b

Code Generation Now Available for Simscape Models

Code generation has been implemented for models that include Simscape and SimHydraulics blocks. For more information, see Code Generation .

New Thermal Block Libraries

Version 2.0 contains new block libraries of fundamental thermal elements, sensors, and sources:

- Conductive Heat Transfer
- Convective Heat Transfer
- Radiative Heat Transfer
- Thermal Mass
- Thermal Reference
- Ideal Heat Flow Source
- Ideal Heat Flow Sensor
- Ideal Temperature Source
- Ideal Temperature Sensor

Additional Physical Signal Blocks

The new Physical Signal blocks introduced in Version 2.0 are listed below:

- PS Constant
- PS Math Function
- PS Max
- PS Min
- PS Sign

Improved Simulation Performance

In Version 2.0, various solver improvements have led to improved simulation performance:

- Enhanced handling of dependent dynamic states (higher-index DAEs)

  Simscape can now handle dependencies among the dynamic states as long as they are linear in the states and independent of time and inputs. This allows you, for example, to connect capacitors in parallel (even with their parasitic series resistances set to 0), inductors in series, and so on.
- Significant reduction of the number of equations, which substantially increased simulation speed

  The typical speedup of your models is between 5 and 10 times. There are some models that are below and above this range. Also, the number of states and equations changed between releases. This means that you will have to reset any calculations that relied on the states (such as initial state setting).

The changes to the simulation technology are significant. You may find that some of your models may require different or tighter tolerances to converge, while others will require no change. Refer to the troubleshooting section in the User's Guide for help in finding the cause of a problem if simulation failed.

New Simscape Demos

The following demos have been added in Version 2.0:

Demo Name	**Description**
DC Motor Thermal Circuit (`ssc_dc_motor_thermal_circuit`)	The demo illustrates how the thermal behavior of a motor can be simulated in lumped parameters.
Round Rod Heat Conduction (`ssc_round_rod_heat_conduction`)	The demo illustrates the usage of thermal blocks for developing a model of a long iron rod that is heated with a heat source through its left face. The right face and the outer cylindrical surface are open to atmosphere, with a force heat convection.

R2007a

Product Introduction

Simscape software extends the Simulink product line with tools for modeling and simulating multidomain physical systems. It enables you to describe multidomain physical systems containing mechanical, hydraulic, and electrical components as physical networks.

Simscape key features are:

- Single modeling environment for modeling and simulating physical systems, such as mechanical, electrical, and hydraulic systems
- Foundation library of physical modeling building blocks and fundamental mathematical elements
- Connection blocks to bridge modeling domains
- Full simulation and limited editing capabilities for models built with SimMechanics, SimDriveline, or SimHydraulics blocks (no license for these products required as long as the products are installed)
- Ability to specify units of parameters and variables, with all unit conversion handled automatically

Simscape software can be used for a variety of automotive, aerospace, defense, and industrial equipment applications. Together with SimMechanics, SimDriveline, SimHydraulics, and SimPowerSystems™ (all available separately), Simscape lets you model complex interactions in electromechanical and hydromechanical systems.

Block Libraries Moved from SimHydraulics to Simscape

The Foundation and Utilities block libraries that used to be included in SimHydraulics (V1.0 and V1.1) are now part of Simscape product.

Compatibility Considerations

Several blocks that used to be in SimHydraulics V1.1 and are now part of Simscape software have undergone changes that have compatibility impact. These blocks are:

- Fluid Inertia
- Inertia
- Mass
- PS Integrator
- Rotational Spring
- Translational Spring

Each of these blocks has a parameter that specifies the initial condition for use in computing the block's initial state at the beginning of a simulation run. In this version, there is a difference in the way these initial conditions are computed, and as a result, the blocks work differently than they used to in the previous version. For details, see the block reference pages.