



Boston Scientific

Advancing science for life™



Coordination of Time-Dependent Simulation Parameters Using the Application Builder in COMSOL Multiphysics®

COMSOL Conference Boston, October 2024

Dr. Paul Belk*, Dr. Anna Harrington*

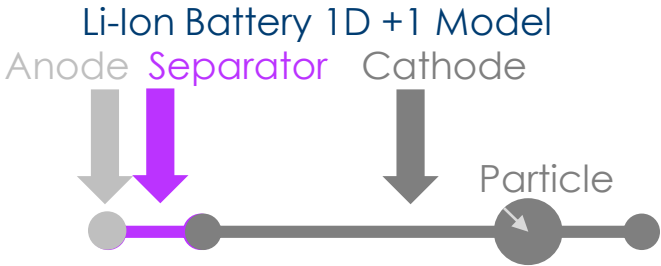
*Boston Scientific Corporation, Boston, MA



Overview

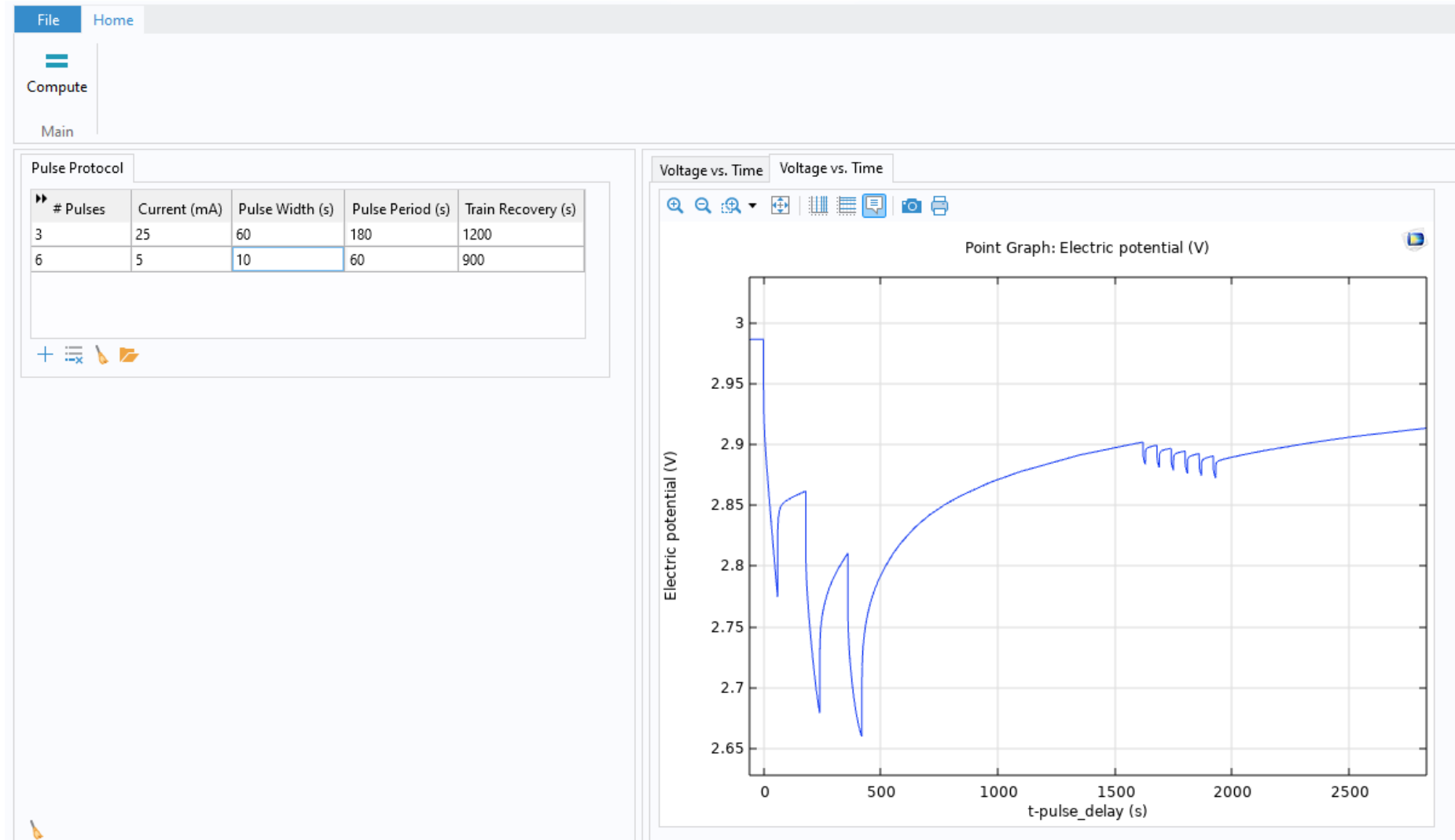
- Time-dependent simulations are most interesting when external factors change
- The simulation needs to be evaluated before and after each change
 - Simulator might not “notice” on its own
 - Explicit evaluation required
 - If the *user* is specifying changes, it can be hard for the *designer* to anticipate them, even with events
- Even designers (Model Builder) can have trouble accounting for all changes
- The App builder (using Java code) can do this automatically
- This is essential for app users... and very handy even for Model designers

Simple Example: Current Pulse



Alternative Use Cases

- Boundary conditions
 - Concentration
 - Temperature
 - Flow Rate
 - Pressure
 - Stress
- Material parameters





Specify Time-Dependent Solver Settings

The screenshot displays the Model Builder interface on the left and the Settings Properties panel on the right. The left pane shows a tree view of the model structure, with 'Time-Dependent Solver 1' selected under 'Solver Configurations'. The right pane shows the configuration for this solver, with two red boxes highlighting specific settings: 'Times to store' set to 'Steps taken by solver' and 'Steps taken by solver' set to 'Intermediate'.

Model Builder

- Comsol2024_UI_3.mph (root)
 - Global Definitions
 - Parameters 1
 - Default Model Inputs
 - Materials
 - Component 1 (comp1)
 - Definitions
 - Applied Current (i_{app})
 - Piecewise 1 (pw1)
 - View 1
 - Geometry 1
 - Materials
 - Lithium-Ion Battery (liion)
 - Multiphysics
 - Mesh 1
 - Study 1
 - Step 1: Current Distribution Initializat
 - Step 2: Time Dependent
 - Solver Configurations
 - Solution 1 (sol1)
 - Compile Equations: Current I
 - Dependent Variables 1
 - Stationary Solver 1
 - Solution Store 1 (sol2)
 - Compile Equations: Time Dep
 - Dependent Variables 2
 - Time-Dependent Solver 1**

Settings Properties

Time-Dependent Solver

Compute to Selected = Compute

Label: Time-Dependent Solver 1

General

Defined by study step: Step 2: Time Dependent

Time unit: s

Output times: 0,100,160,660,2000 s

Times to store: Steps taken by solver

Store every Nth step: 1

Relative tolerance: 0.001

Absolute Tolerance

Time Stepping

Solver type: Implicit

Method: BDF

Steps taken by solver: Intermediate

Interpolate solution at end time

Initial step: (1)[s] s

Maximum step constraint: Automatic

Maximum BDF order: 2

Minimum BDF order: 1



Link Data Structures to UI

The screenshot shows a software interface with a table and a properties panel. The table has the following data:

# Pulses	Current (mA)	Pulse Width (s)	Pulse Period (s)	Train Recovery (s)
1	25	60	180	1200

The properties panel shows the following settings:

- Name: table1
- Show headers
- Automatically add new rows
- Sortable
- Sources: Declarations
 - Array 2D Double
 - pulses

Below the screenshot is a diagram showing the relationship between three components:

- Model Builder (Piecewise function)
- Application Builder (Table Object)
- Methods (Java Code)

Arrows indicate the following relationships:

- Model Builder and Application Builder are connected by a double-headed arrow.
- Methods (Java Code) has an arrow pointing to Model Builder.
- Methods (Java Code) has an arrow pointing to Application Builder, labeled "Declarations".



- ✓ Step 0: Create UI in App Builder to specify time-dependent factors
- ✓ Step 1: Link data structures to UI
- ✓ Step 2: Specify Time Dependent Solver Settings
- ✓ Step 3: Comment Method Code
 - Step 4: Record code to set up conditions
 - Step 5: Record code to set up evaluation times
 - Step 6: Modify Method Code
 - Step 7: Fire and Forget

Method Code Structure

```
1 // Get data from pulse table ==== CUSTOM
2
3 // Modify piecewise function ==== RECORD
4
5 // populate square pulses for each row ==== CUSTOM
6
7 // setup square pulse in piecewise function ==== RECORD
8
9 // find transition time for pulses ==== CUSTOM
10
11 // setup transition time for piecewise function ==== RECORD
12
13 // Find evaluation times ==== CUSTOM
14
15 // set times for evaluation ==== RECORD
```



Record Code to Set Up Conditions



The screenshot displays the software's interface. On the left, a red box highlights a green circular button labeled "Record Code". The top toolbar contains several sections: "Libraries" with "Utility Class", "Java External Java Library", and "External C Library"; "Edit" with "Revert to Saved"; "Code" with "Language Elements", "Model Expressions", and "Stop Recording" (highlighted in blue); "Check Syntax"; "Go to Node"; "Use Shortcut" (B+C); "Create Local Variable" (a=); "Continue"; "Step"; and "Step Into".

The "Application Builder" window is open, showing a tree view of the project structure. The "Methods" folder is expanded, and "calcPulses_0" is selected. The code editor on the right shows the following code:

```
1 // Get data from pulse table ==== CUSTOM
2
3 // Modify piecewise function ==== RECORD
4
5 |
6
7 // populate square pulses for each row ==== CUSTOM
8
9 // setup square pulse in piecewise function ==== RECORD
10
11 // find transition time for pulses ==== CUSTOM
12
13 // setup transition time for piecewise function ==== RECORD
14
15 // Find evaluation times ==== CUSTOM
16
17 // set times for evaluation ==== RECORD
```



Record Code to Set Up Conditions

The screenshot shows the COMSOL Multiphysics software interface. The top menu bar includes File, Home, Definitions, Geometry, Materials, Physics, Mesh, Study, Results, and Developer. The main workspace is divided into several panes. On the left is the Model Builder tree, which shows a hierarchy starting with 'Comsol2024_UI_3.mph (root)', followed by 'Global Definitions', 'Component 1 (comp 1)', and 'Definitions'. Under 'Definitions', 'Piecewise 1 (pw1)' is selected. The right pane shows the 'Settings' for the 'Piecewise' function. The 'Label' is 'Piecewise 1' and the 'Function name' is 'pw1'. The 'Definition' section shows the 'Argument' as 't', 'Extrapolation' as 'Constant', 'Smoothing' as 'Continuous first derivative', and 'Transition zone' as 'Absolute size'. The 'Size of transition zone' is set to '0.001'. There is a checkbox for 'Smooth at endpoints' which is unchecked. Below this, there is a table for 'Intervals' with columns for 'Start', 'End', and 'Function'.

Start	End	Function
-10	0	0
0	60	0.025

- While Recording Code:**
- Create Piecewise Function
 - Set size of Transition zone
 - Edit Start, End and Function Intervals



Record Code to Set Up Conditions

The screenshot shows the software interface with the 'Method' menu open. The 'Stop Recording' option is highlighted. The 'Application Builder' tree view on the left shows the project structure, with 'calcPulses_1' selected under 'Methods'. The main window displays a code editor with the following code:

```
1 // Get data from pulse table ==== CUSTOM
2
3 // Modify piecewise function ==== RECORD
4
5 model.component("comp1").func("pw1").set("smooth", "contd1");
6 model.component("comp1").func("pw1").set("zonelengthtype", "absolute");
7 model.component("comp1").func("pw1").set("smoothzone", 0.001);
8 model.component("comp1").func("pw1").setIndex("pieces", -10, 0, 0);
9 model.component("comp1").func("pw1").setIndex("pieces", 0, 0, 1);
10 model.component("comp1").func("pw1").setIndex("pieces", 0, 0, 2);
11 model.component("comp1").func("pw1").setIndex("pieces", 0, 1, 0);
12 model.component("comp1").func("pw1").setIndex("pieces", 60, 1, 1);
13 model.component("comp1").func("pw1").setIndex("pieces", 0.025, 1, 2);
14 model.component("comp1").func("pw1").set("argunit", "s");
15 model.component("comp1").func("pw1").set("fununit", "A");
16
17
18 // populate square pulses for each row ==== CUSTOM
19
```



Record Code to Set Up Conditions

```
Code | Debug
Preview Main Window x Protocol: pulseTable x calcPulses_0 x Graphics
1 // Get data from pulse table ==== CUSTOM
2
3 // Modify piecewise function ==== RECORD
4
5 model.component("comp1").func("pw1").set("arg", "t");
6 model.component("comp1").func("pw1").set("smooth", "contd1");
7 model.component("comp1").func("pw1").set("zonelengthtype", "absolute");
8
9 model.component("comp1").func("pw1").set("argunit", "s");
10 model.component("comp1").func("pw1").set("fununit", "A");
11
12
13 // populate square pulses for each row ==== CUSTOM
14
15 model.component("comp1").func("pw1").setIndex("pieces", -10, 0, 0);
16 model.component("comp1").func("pw1").setIndex("pieces", 0, 0, 1);
17 model.component("comp1").func("pw1").setIndex("pieces", 0, 0, 2);
18 model.component("comp1").func("pw1").setIndex("pieces", 0, 1, 0);
19 model.component("comp1").func("pw1").setIndex("pieces", 60, 1, 1);
20 model.component("comp1").func("pw1").setIndex("pieces", 0.025, 1, 2);
21
22 // setup square pulse in piecewise function ==== RECORD
23
24 // find transition time for pulses ==== CUSTOM
25
26 // setup transition time for piecewise function ==== RECORD
27 model.component("comp1").func("pw1").set("smoothzone", 0.001);
28
```



Record Code to Set Up Evaluation Times

The screenshot shows the COMSOL Multiphysics Model Builder interface. The 'Model Builder' tree on the left shows the hierarchy: Comsol2024_UI_3.mph (root) > Global Definitions > Parameters 1 > Default Model Inputs > Materials > Component 1 (comp1) > Definitions > Applied Current (i_app) > Piecewise 1 (pw1). The 'Settings' pane on the right is open to the 'Time Dependent' study. The 'Study Settings' section is expanded, showing 'Time unit' set to 's' and 'Output times' set to '0,100,160,660,2000' s. The 'Output times' field is highlighted with a red box. The 'Tolerance' is set to 'Physics controlled'.

While Recording Code:
Edit Output times



Record Code to Set Up Evaluation Times

The screenshot shows a software interface with a toolbar at the top. The toolbar is divided into two sections: 'Code' and 'Debug'. In the 'Code' section, the 'Stop Recording' button is highlighted with a blue box. Other buttons in the 'Code' section include 'Language Elements', 'Model Expressions', 'Check Syntax', 'Go to Node', 'Use Shortcut', and 'Create Local Variable'. The 'Debug' section includes 'Continue', 'Step', 'Step Into', 'Step Out', and 'Stop'. Below the toolbar, there is a window titled 'calcPulses_0' containing a code editor. The code editor shows a series of lines of code, with the following lines highlighted in a red box:

```
16 model.component("comp1").func("pw1").setIndex("pieces", 0, 0, 1);
17 model.component("comp1").func("pw1").setIndex("pieces", 0, 0, 2);
18 model.component("comp1").func("pw1").setIndex("pieces", 0, 1, 0);
19 model.component("comp1").func("pw1").setIndex("pieces", 60, 1, 1);
20 model.component("comp1").func("pw1").setIndex("pieces", 0.025, 1, 2);
21
22 // setup square pulse in piecewise function ==== RECORD
23
24 // find transition time for pulses ==== CUSTOM
25
26 // setup transition time for piecewise function ==== RECORD
27 model.component("comp1").func("pw1").set("smoothzone", 0.001);
28
29 // Find evaluation times ==== CUSTOM
30
31 // set times for evaluation ==== RECORD
32 model.study("std1").feature("time").set("tlist", "0,100,160,660,2000");
33
```



Method: Inputs

```
// Get data from pulse table == CUSTOM
// determine number of pieces in piecewise function.
int nPieces = 0;
double minDur = pulses[0][2]; // for transition times
for (int row = 0; row < matrixSize(pulses)[0]; ++row) {
    nPieces += pulses[row][0]*2; // each row also has a recovery piece
    if (pulses[row][2] < minDur) minDur = pulses[row][2];
    if (pulses[row][3] < minDur) minDur = pulses[row][3];
}
int curPiece = 0;
double xTime = minDur/10; // set the transition time based on the shortest pulse

// for each piece (change) in the piecewise, we place two evaluation times
double[] times = new double[2*(nPieces)+4];
```

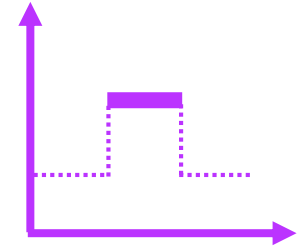


Method: Modify Piecewise Function

```
// setup square pulse in piecewise function == RECORD  
for (int pNum = 0; pNum < nPulses; ++pNum) {
```

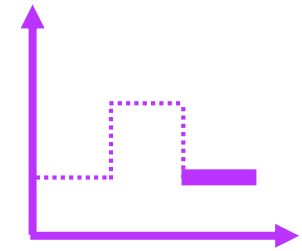
```
// rising edge
```

```
times[nTime++] = pTime-xTime+firstPulse;  
times[nTime++] = pTime+xTime+firstPulse;  
model.component("comp1").func("pw1").setIndex("pieces", pTime, curPiece, StartTime);  
pTime += duration;  
model.component("comp1").func("pw1").setIndex("pieces", pTime, curPiece, EndTime);  
model.component("comp1").func("pw1").setIndex("pieces", current, curPiece, Amplitude);
```



```
// falling edge
```

```
++curPiece;  
times[nTime++] = pTime-xTime+firstPulse;  
times[nTime++] = pTime+xTime+firstPulse;  
model.component("comp1").func("pw1").setIndex("pieces", pTime, curPiece, StartTime);  
pTime += (period-duration);  
model.component("comp1").func("pw1").setIndex("pieces", pTime, curPiece, EndTime);  
model.component("comp1").func("pw1").setIndex("pieces", 0.0, curPiece, Amplitude);
```





Method: Modify Evaluation Times

```
// Setup transition time for piecewise functions ==== RECORD  
model.component("comp1").func("pw1").set("smoothzone", xTime/5);
```

```
// Find evaluation times ==== CUSTOM  
String timeStr = "0";  
for (int i = 1; i < nTime; ++i) {  
    timeStr += ",";  
    timeStr += toString(times[i]);  
}
```

```
// set times for evaluation ==== RECORD  
model.study("std1").feature("time").set("tlist", timeStr);
```



Advanced Capabilities

File Ribbon Tab 1

Check Initialize Compute Pulse Study Compute EIS

Ribbon Section 1

Geometry Cathode Recipe Protocol Physics

Default background current: 68 uA

Pulses

Delay to first pulse: 4E6 s

# Pulses	Current (mA)	Pulse Width (s)	Pulse Period (s)	Train Recovery (s)
3	25	60	180	600
6	12	10	60	300
3	25	60	180	600

initial DOD: 1.7 %

simulation time: 70[d]

Capacity used at background 114.2 mAh

Capacity used by pulses 2.7 mAh

Final DOD 10.5 %

Simulation Log of Parameter Output vs. Nominal:
Pulse 1)
Pulse 2)

Voltage vs. Time Pulse Output Voltage vs mA hrs Voltage vs DOD PulseInfo Nyquist Plot

Cathode Voltage vs Time (Pulses)

Electric potential (V)

time from pulse start (s)



Key Takeaways

- We demonstrate how to customize and model irregularly changing boundary conditions across a wide range of time scales.
- The powerful Application Builder creates a user-friendly app which is linked to a customized piecewise function in the model.
- To ensure an optimal simulation, evaluation times are recorded and specified to resolve pulse events across large time scales.
- This automated approach saves time, reduces errors, and can be applied to many different physics.

Contact

Dr. Paul Belk

Paul.Belk@bsci.com