

Getting started with UHPC Protocol's new built-in variables, custom variables, and functions

2024-April-22

Originally introduced in v2.6.0, as custom user-defined variables, as of v2.10.0, UHPC Protocol additionally supports using pre-defined built-in variables within the 'Step Control Conditions' text boxes, as well as the 'Current' and 'Voltage' text boxes.

Built-in Variables

Several variables are updated automatically in UHPC Control and may be used in UHPC Protocol freely to quickly build efficient cycling protocols.

- Start Capacity (Ah)
 - Available as built-in custom variable **StartCap**
 - Calculated at start of test using the Capacity defined in UHPC Control's Capacity column.
- Last Step Charge Capacity (Ah)
 - Available as built-in custom variable **LStepCCap**
 - Calculated after each charge step
- Last Step Discharge Capacity (Ah)
 - Available as built-in custom variable **LStepDCap**
 - Calculated after each discharge step
- Last Cycle Charge Capacity (Ah)
 - Available as built-in custom variable **LCycleCCap**
 - Calculated after each Increment Cycle Counter
- Last Cycle Discharge Capacity (Ah)
 - Available as built-in custom variable **LCycleDCap**
 - Calculated after each Increment Cycle Counter

Example use 1

Users may wish to end a test if the capacity of a given cycle is below 80% of the starting capacity. By including an “end protocol” or “end loop” step condition on any or all steps after the first cycle, ending a test when the cycle capacity has degraded by 20% of the starting capacity is simple. Some users may refer to this as removing the cells when they reach 80% State of Health (SOH).

Step Control Conditions

CC-CV Charge

1 A to 4.2 V

Action +

When

:

End protocol

Any time

Last Cycle Charge Cap.

<

0.8*StartCap

Ah

Example use 2

Using a charge/discharge current based on the capacity of the cell is very common (C/10, C/3, or C/1). Not only are identical cells often slightly different in terms of starting capacity, but when cycling many times, this capacity changes across the lifetime of the cell. By dynamically choosing a current based on the last cycle charge or discharge capacity, a user can cycle a cell closer to the true desired C/XX current. For batch testing this can limit the influence of the cell-to-cell variation.

In the example below, there’s a sample protocol that uses a C/10 current initially, and then after the first full cycle, begins looping while continuously re-calculating the C/10 current as the cell cycles.

Protocol Control Step(s) +

Reset Step Limits

1	:	Constant Current Discharge	10	C/xx	to	2.8 V
2	:	Increment Cycle Counter				
3	:	CC-CV Charge	10	C/xx	to	4.2 V
4	:	CC-CV Discharge	10	C/xx	to	2.8 V
5	:	Repeat steps below	100	times		
6	:	Increment Cycle Counter				
7	:	CC-CV Charge	LCycleDCap/10	A	to	4.2 V
8	:	CC-CV Discharge	LCycleDCap/10	A	to	2.8 V

Example use 3

Looking at “Example use 2” above, some users may not wish for the current to change after the first cycle but still wish to calculate the initial true starting capacity and then cycle continuously with that. In this situation, a user can create their own custom variables that only update when desired.

Below, for “Step 4” we’ve added an update variable step condition using the Right Click -> Add Action feature. During, Step 4 var01 will be updated continuously as data saves and will contain the step capacity (or cycle capacity) for the first full charge cycle.

Step Control Conditions

CC-CV Discharge 10 C/xx to 2.8 V

Action +	When
: End step During CV	abs(I) < 50 C/xx
: Save data Any time	Δt > 5 seconds
: Update Variable Any time	Step Capacity = var01 Ah

Add Action
Delete Action
Add Action to All Steps

Now below, the updated protocol using var01/10 as the charge current will give each cycle a consistent (starting capacity / 10) charge current, while still accounting for initial cell variation. In this example, we only use “Update Variable” in step 4. Later steps only reference this variable.

Protocol Control Step(s) +

Reset Step Limits

1	: Constant Current Discharge	10 C/xx	to	2.8 V
2	: Increment Cycle Counter			
3	: CC-CV Charge	10 C/xx	to	4.2 V
4	: CC-CV Discharge	10 C/xx	to	2.8 V
5	Repeat steps below 100 times			
6	: Increment Cycle Counter			
7	: CC-CV Charge	var01/10 A	to	4.2 V
8	: CC-CV Discharge	var01/10 A	to	2.8 V

Note

Keep an eye on your units. UHPC Protocol will convert all values to root units *before* saving. For example, in the example below for a 1Ah charge. **VAR01** would be saved as 1 and **VAR02** would be saved as 1000.

:	Update Variable	Step Capacity	=	VAR01	Ah
:	Update Variable	Step Capacity	=	VAR02	mAh

Which could result in a 1mA charge in the first image below, but an attempt to charge at 1000A in the second image below!

Step Control Conditions

Constant Current Charge VAR01 mA to 3 V

Step Control Conditions

Constant Current Charge VAR02 A to 3 V

New Custom Variables

To create new variables, or overwrite an existing variable, select “Create New”, or select an existing variable from the “Update Variable” dropdown menu:

Step Control Conditions

Constant Current Charge 1 A to 4.2 V

Action + **When**

:	End step	Or	Step Capacity	>	1	Ah
:	Update Variable	Or	Step Capacity	=	VAR01	Ah

VAR01
Create New
VAR04
VAR03
VAR02

When “Create New” is selected, a textbox appears. **All variables must contain at least one letter and no spaces.** They should not contain other variable names.

The list of newly created Custom Variables will be erased upon closing UHPC Protocol. **To persist Custom Variables, create them as part of the Default Configuration.**

Default Configuration

Create or change configuration

Available Custom Variables will also be loaded into the session when Protocols with existing Custom Variables are loaded.

Note

Custom Variables do not need to follow the **VAR01**, **VAR02**, **VARXX** naming conventions. Users are free to define more intuitive names – **STEP1CAP**, **STEP2CAP** etc.

Custom Expressions

Additionally, right-side conditions and Current may be represented with a function containing variables to be executed in real time. **Voltage expressions are not valid at this time.**

Custom Expressions are intended to be used with Custom Variables but can also be used to execute simple algebra. The following are all valid expressions:

$$2 + \frac{10}{15 + 5}$$

$$2 + 10 / (15 + 5)$$

$$VAR_{01} + \frac{0.01 \times VAR_{02} \times 8}{12 + 7}$$

$$VAR01 + (0.01 * VAR02 * 8) / (12 + 7)$$

$$\frac{VAR_{01}}{VAR_{03}} - 5$$

$$VAR01 / VAR03 - 5$$

$$\frac{VAR_{01}}{VAR_{04} + 9}$$

$$VAR01 / (VAR04 + 9)$$

The expression below will be evaluated at the start of the step execution for Current, and throughout the step for End/Save Step Conditions:

Step Control Conditions

Constant Current Charge

A
 to
 V

Action +

		When	
:	End step	Or	step time > <input type="text" value="VAR01*10"/> hours
:	Save data	Or	ΔV > <input type="text" value="VAR02/15 + 5"/> V
:		Or	Δt > <input type="text" value="1"/> seconds
:	Update Variable	Or	Step Capacity = <input type="text" value="VAR04"/> Ah

Note that in some cases, expressions cannot be evaluated. For example, some variables may grow to illegal values over the course of a test. Depending on where the expression is called, UHPC Protocol manages these exceptions.

In cases where expressions cannot be evaluated:

- In the case of current:
 - o The test will halt in an error state
- In the case of a right-side condition:
 - o The test may continue while logging errors of which condition could not be evaluated.
This may result in steps running past their end conditions

Info

Longer variable names or expressions require more width and may reduce readability in the program GUI. If you are struggling to read or write the full expression, you may copy/paste the expression with the help of Notepad. **If Protocol refuses to paste, your expression likely contains illegal characters.** Remember that only alphanumeric characters and basic math evaluators are allowed, 0-9A-Z, . / * () + - ^.

Reducing complexity reduces chances of error. The logic in the code follows the BEDMAS order of operation, after replacing all variables with numbers.