HCA Tech Note 110

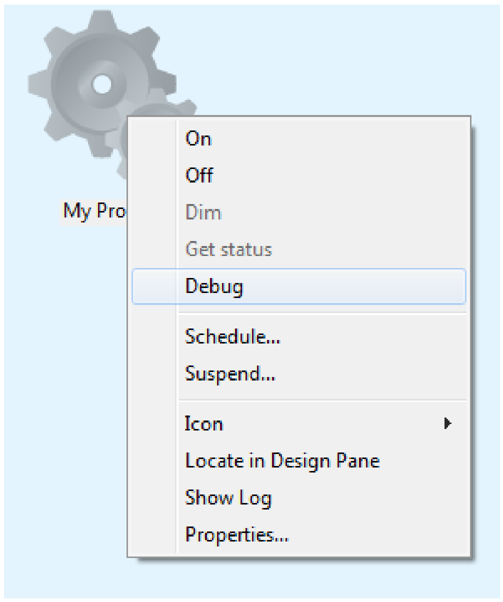
The HCA Visual Program Debugger (modified 17-Oct-2023)

The Visual Program debugger, a feature only in HCA Plus, can help you find problems in programs that you create. Like a development environment for more traditional programming languages, using the debugger you can stop execution of the program at various points, examine and modify state, and control the flow of execution.

Don't think of using the debugger only to find and fix problems. Programs can be complex if they must handle a variant of conditions, some of which might not occur often. Not all the sections of a program may get executed in the normal course of events. Using the debugger, you can execute those "little used" program paths so you know that the program is fully functional before deploying it into your home environment.

This technical note covers in some detail the top debugger features that you should know about. More detail is contained in the HCA Used Guide.

Starting a program in the debugger



The simplest way to start a program is to right-click on it and select debug from the popup menu.

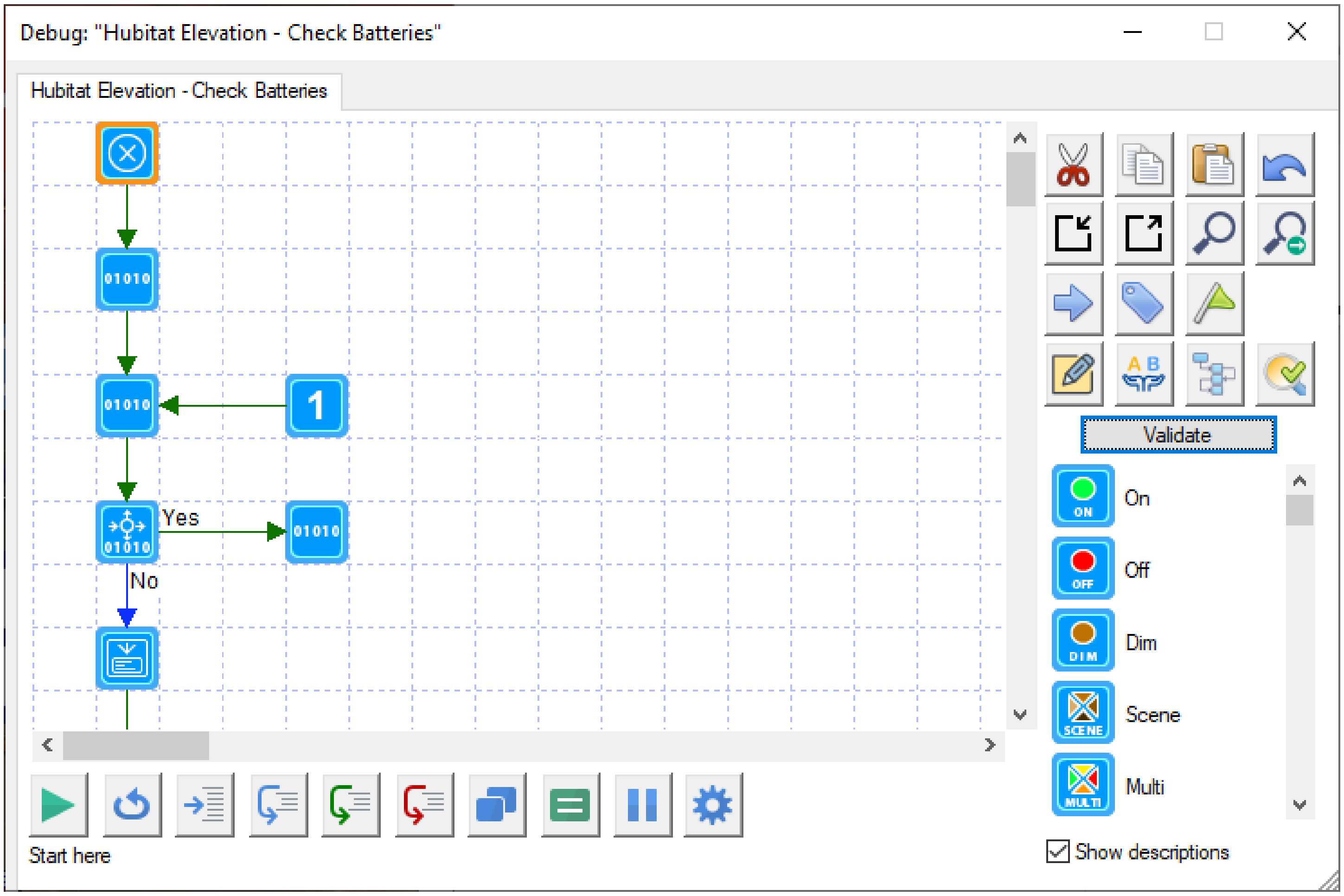
The program opens in the debugger which looks a lot like the Visual programmer with additional buttons in the debugger control area below the programming canvas.

The key fact here is that the program has not been started yet. When in the debugger, the current element – the element that is about to be executed – is outlined in orange. Since the program has not yet started, the Begin-Here element is the one outlined in orange.

The next thing you might notice is that the debugger opens in its own window separate from the main HCA window. You can minimize the debugger window to the task bar if you want. Since it is a separate Window you can continue to use all the features in HCA during a debugging session, for example controlling devices, changing schedules, using the interface tools, etc.

To close the debugger, use the "close button" – the "X" – in the upper right of the debugger window.

TechSup@HCATech.com
www.homecontrolassistant.com

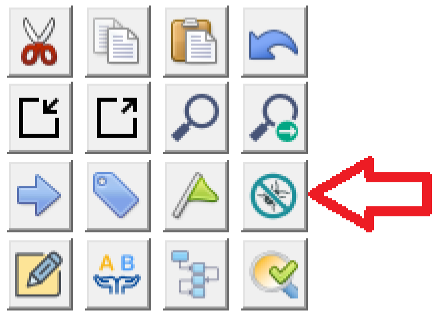

Debug: "Hubitat Elevation - Check Batteries"
Hubitat Elevation - Check Batteries
1
Yes
No
Validate
On
Off
Dim
Scene
Multi
Start here
Show descriptions

At the bottom of the debugger are ten buttons with somewhat cryptic icons. From left to right they are:

- Start
  This starts the programming running from the current element and execution continues until the program completes or it encounters a breakpoint.
- Restart
  Select as the current element the Begin-Here so that you can start over.
- Step
  Execute the current element and then stop.
- Step Over
  Don't execute the current element. Select as the current element the next element.

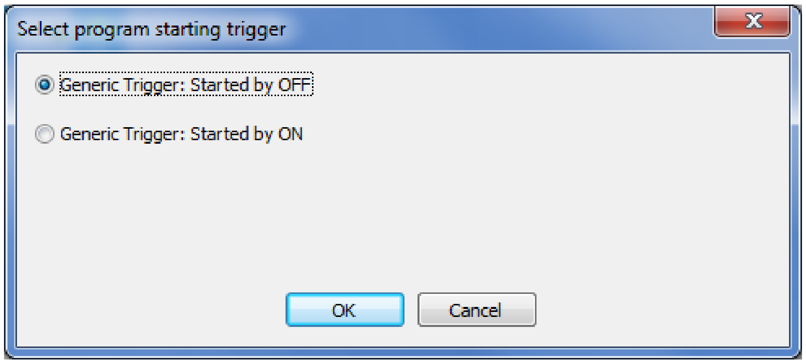
- Step Over Yes
  Don't execute the current element. Select as the current element the first element on the "Yes" path of a decision element.
- Step Over No
  Don't execute the current element. Select as the current element the first element on the "No" path of a decision element.
- Open Programs
  Open a dialog showing all the programs in your design. You can select one of those programs and it is opened in its own window. You might want to do this to view the program or to set breakpoints in it.
- Open Variable Viewer
  Opens a window that shows all the local variables, global variables, and program parameters in the current program and any programs on the stack.
- Open Breakpoints Viewer
  Opens a window that shows all breakpoints set and from there you can delete, enable, or disable them.
- Open Explorer
  The explorer window lets you examine any file, design, or JSON handle and evaluate expression you enter.

Top tip: If you are working on a program in the programmer and want to start the debugger, there is a button for that in the tools panel. It saves the program and that starts it in the debugger.



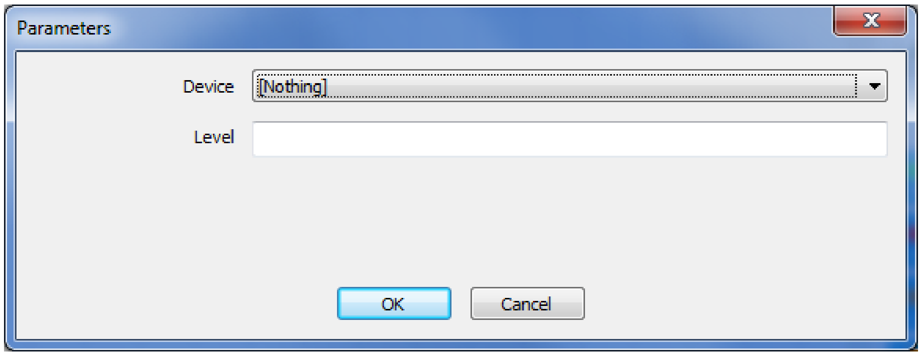
Choosing a starting trigger and parameter values

When starting a program running from its Begin-Here element, if the program has triggers or it has parameters, the debugger prompts you to select the starting trigger and/or parameter values.


Select program starting trigger
(● Generic Trigger: Started by OFF)
Generic Trigger: Started by ON
OK
Cancel

This program has two triggers. This style of trigger selection dialog shows up to 4 triggers. For programs with more than 4 triggers another style of dialog is used with a dropdown for choosing the trigger.

For programs with parameters, the debugger has you select values for them when starting the program.

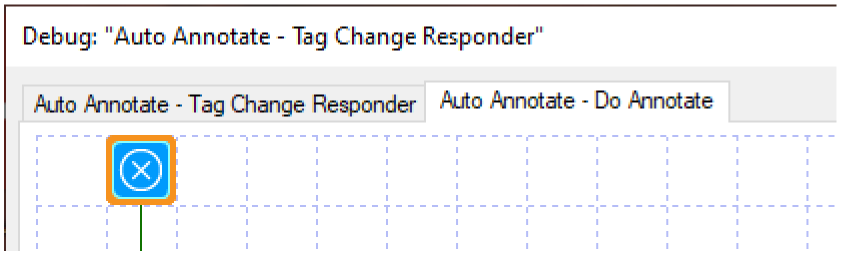


This program has two parameters – the first is an object and the second is a value. Each parameter selection/edit control shows the parameter name to its left. Object parameters have a dropdown showing the objects in your design for you to select from. Value parameters are entered as text and the debugger converts it to a value.

Executing elements

Once the program starts you can use the buttons in the debugger control area to control execution. You can use the “step” buttons to move from one element to the next and in that way, see the flow of your program element by element.

The “step over” buttons don’t execute the current element but move the current element to the next element. If it is a decision element you can use the “step over yes” or “step over no” buttons to follow the “yes” or “no” path when you step a decision element.


Debug: "Auto Annotate - Tag Change Responder"
Auto Annotate - Tag Change Responder
Auto Annotate - Do Annotate

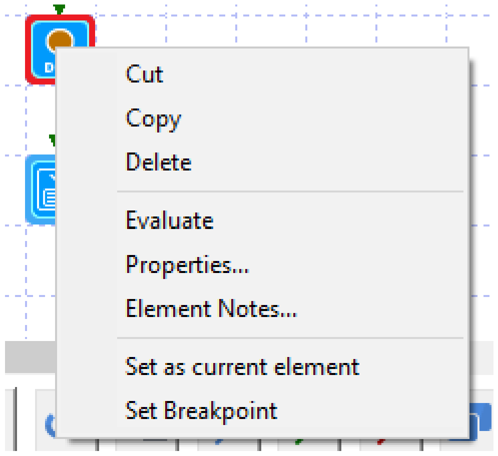
If the program being debugged starts another program using the Start-Program element, and if you "step into" that program or the program stops because of a breakpoint in that program, the debugger shows a tab for that other program in addition to the program that started it. As you can see in the picture above, in this new program tab the current element shows an orange border.

If you were to choose the other program tab – labeled "Auto Annotate – Tag Change Responder" in this example - you can see that it also has an element with an orange border. That element is the Start-Program element that started the program shown in the tab to its right. This makes it possible to always know what element your program will come back to when the program started by Start-Program completes.

Note: This is the one place where the element with the orange border has already been executed. As described above, normally the element with the orange border is the element about to be executed. This is done this way so you can quickly find the Start-Program element when moving from program tab to program tab.

Technical Tip: If you are more of a traditional programmer type you can quickly see that the tabs of the programs show the execution stack. The right-most is the program at the top of the stack and the first program from the left end is at the bottom of the stack.

When there is more than one program tab, only the buttons in the debugger control area are active for the right-most program tab since that is where execution is happening.



In addition to using the Start, Restart, and Step buttons which all work with the current element, you can also change the current element.

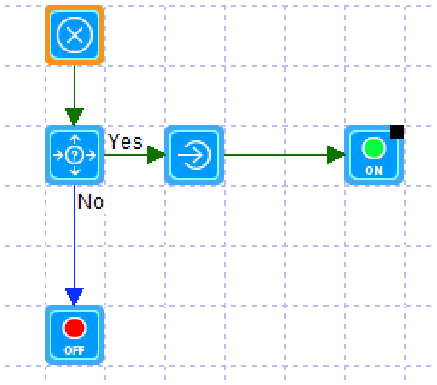
Right-click on an element and select *Set as current element* from the popup menu. This is an easy way to choose the element to next run from or step from.

Choosing the current element and using the Step button is a simple way to execute any element of your choice.

Setting breakpoints

While the "step" buttons are useful you can also mark an element so that before it executes, the debugger stops. These are called "breakpoints". The idea is that you mark certain elements in your program so that when execution gets to them the program stops and the debugger takes over.

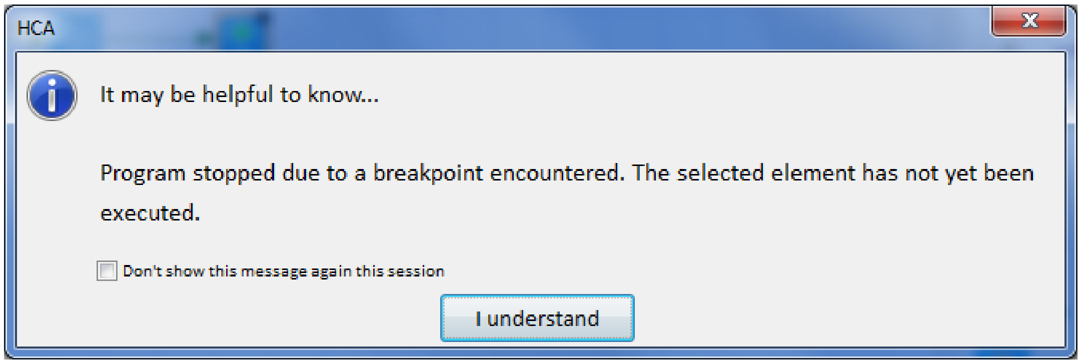
To set a breakpoint on an element right click on that element and choose *Set Breakpoint* from the popup menu. If the element already has a breakpoint the menu choice instead shows *Clear Breakpoint*.



When an element has a breakpoint, it shows with a black square in its upper right.

If the breakpoint is set on the element but is disabled, then the black square is instead grey.

When execution stops at a breakpoint HCA displays a popup:


HCA

It may be helpful to know...

Program stopped due to a breakpoint encountered. The selected element has not yet been executed.

Don't show this message again this session

Controlling Breakpoints

The seventh button in the debugger control area opens a selector for all programs in your design. This can be a handy tool when setting breakpoints. Suppose that your program contains Start-Program elements, and you want to set breakpoints in the programs that will be started when those elements execute. Use this button to open the list of programs, find the program and open it, and then set a breakpoint in it.

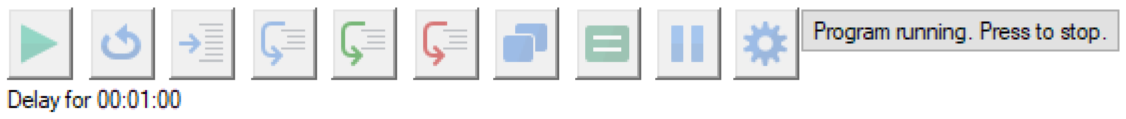
You can also open the Breakpoint Viewer window which shows the breakpoints you have set and if they are enabled or not. When created, a breakpoint is initially enabled but can also be disabled. A breakpoint that is disabled remains set as part of the element state but when that element is executed the debugger doesn’t stop.

Why would you want to disable a breakpoint? There may be situations where you want to run the program to completion and not stop but not lose the breakpoints you have already set in case the program isn’t working as you expect it should. If you delete the breakpoints you may find you need to go back and set them all again.

Note: A breakpoint is part of the element state, so it is stored with the element in the HCA file. In this way, you can set breakpoints and they persist between invocations of HCA if you have saved the file.

Slow and Fast elements

The debugger divides all elements into two classes: Fast execution and slow execution. Fast elements are ones that typically take only a second or two for HCA to execute. Elements like ON, Off, Exit, etc. are all fast. Other elements like the Delay element or elements that request device status can take a longer time to execute. When the debugger encounters an element that is slow, a button appears in the debugger control area.



If you press the *stop* button, then execution of the current slow element is aborted and control returns to the debugger. This provides a useful way for a program to have, for example, a delay element set for one hour and not have to wait for the full hour.

Viewing the Log

You can have any of the three log windows open during debugging. There is also a single window that has three tabs – one for each log. In this log viewer you can create, modify, or use log filters. The debugger configuration has an option so that the log can automatically show only program elements from the currently being debugged program. See the section below on configuring the debugger.

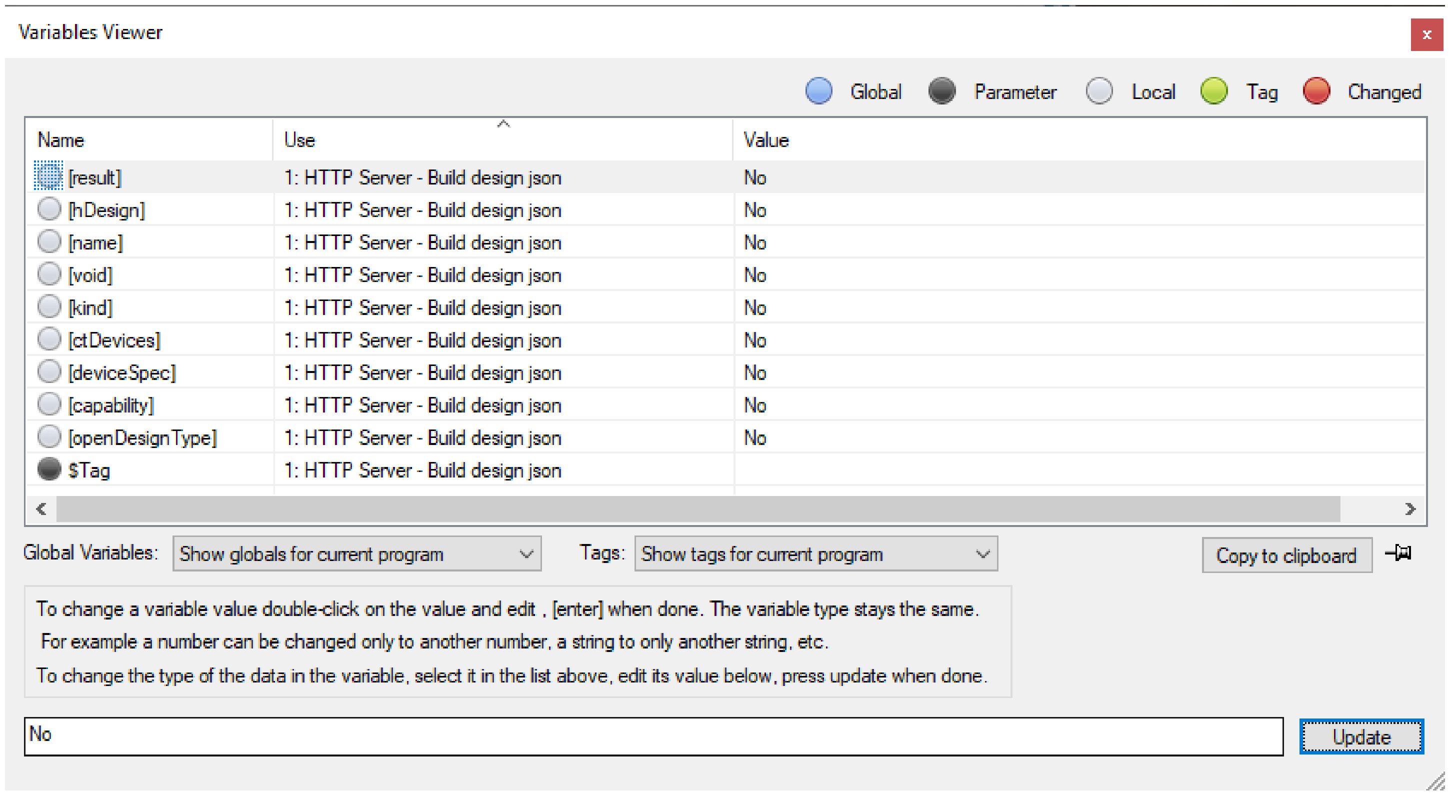
Viewing and modifying variables values

The variables viewer shows all the local variables and their values in the current program and any others on the stack. From this tab you can also modify their values if needed. As it says in the notes in the viewer window, you can change a variable value by editing the value "in-place" or by using the "Update" button at the bottom. The variable viewer also can show tags on the current program or on selected objects.

The viewer shows both local and global variables plus tag values. Before each variable is a colored dot. The color tells you important information:

- Light Gray – Local variable
- Dark Gray – Parameter
- Blue – Local variable
- Green - Tag
- Red – the value has changed since the last time the debugger halted

When the program stops for a breakpoint, any variable that has changed its value from when the program last stopped shows a red mark in front of that line.


Variables Viewer

● Global ● Parameter ○ Local ● Tag ● Changed

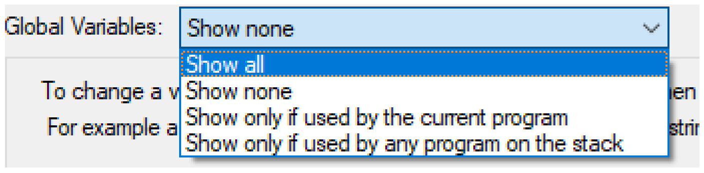
Name	Use	Value
[result]	1: HTTP Server - Build design json	No
○ [hDesign]	1: HTTP Server - Build design json	No
○ [name]	1: HTTP Server - Build design json	No
○ [void]	1: HTTP Server - Build design json	No
○ [kind]	1: HTTP Server - Build design json	No
○ [ctDevices]	1: HTTP Server - Build design json	No
○ [deviceSpec]	1: HTTP Server - Build design json	No
○ [capability]	1: HTTP Server - Build design json	No
○ [openDesignType]	1: HTTP Server - Build design json	No
● $Tag	1: HTTP Server - Build design json	

Global Variables: Show globals for current program
Tags: Show tags for current program

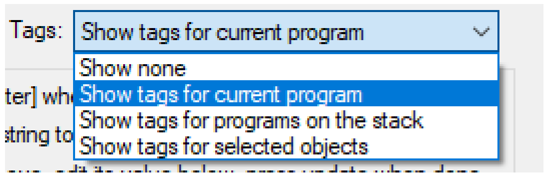
To change a variable value double-click on the value and edit , [enter] when done. The variable type stays the same.
For example a number can be changed only to another number, a string to only another string, etc.
To change the type of the data in the variable, select it in the list above, edit its value below, press update when done.

No

There is a dropdown option that lets you control which global variables show in the viewer. These options are:


Global Variables: Show none
- Show all
- Show none
- Show only if used by the current program
- Show only if used by any program on the stack

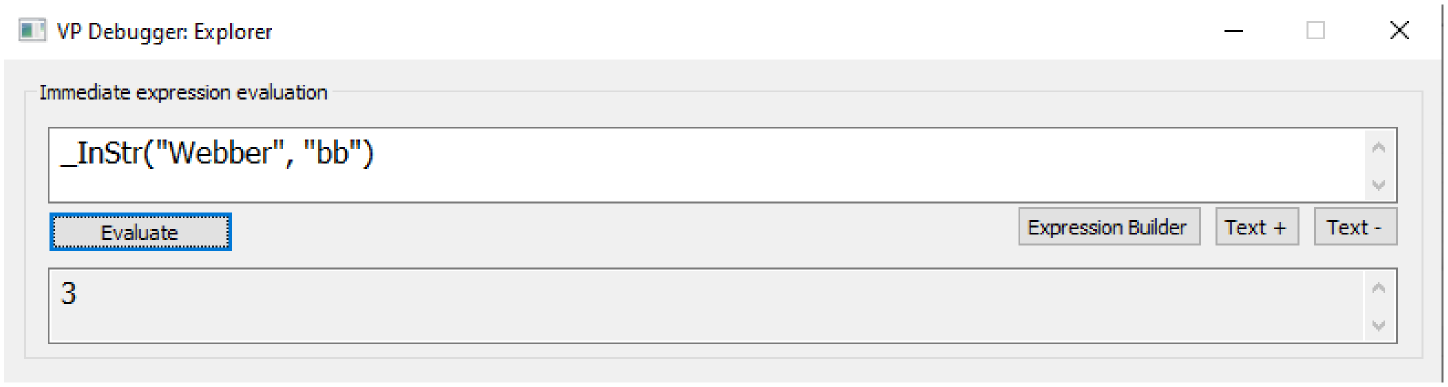
The action of each global variable display choice should be clear from the option text. There is a similar option for showing tag values:


Tags: Show tags for current program
- Show none
- Show tags for current program
- Show tags for programs on the stack
- Show tags for selected objects

The Explorer

The Explorer window is useful for two purposes: Evaluating expressions and viewing open handles.

The explorer can be used if you want to see the action of expressions used in the Compute and Compute-Test elements. For example, suppose you are unsure of what the _InStr function does, and does it return a position in the string where 1 is the first character or is it zero?



```
_InStr("Webber", "bb")
```

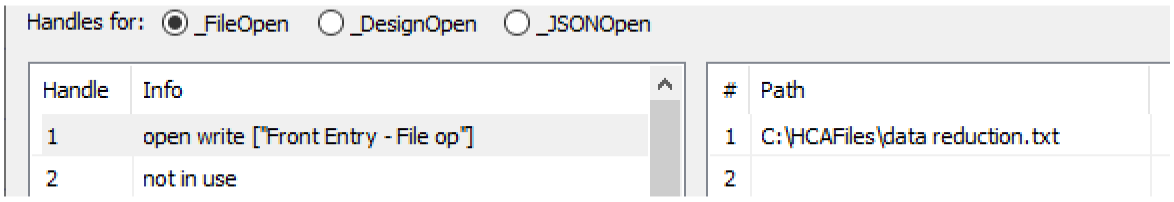
```
3
```

In this example I typed in the expression and pressed the "Evaluate" button. And the answer "3" tells me that this function numbers the characters in the string where the first character is 1. This is an easy example, but sometimes expressions can get very complex and handing a quick way to "play" with the expression to see what it does can be rather helpful.

The other use for the explorer is when you are using file, design, or Json handles.

Unfamiliar with handles? Look at the expressions chapter of the User Guide for the functions _FIleOpen, _DesignOpen, and _JSONOpen.

Exploring file handles



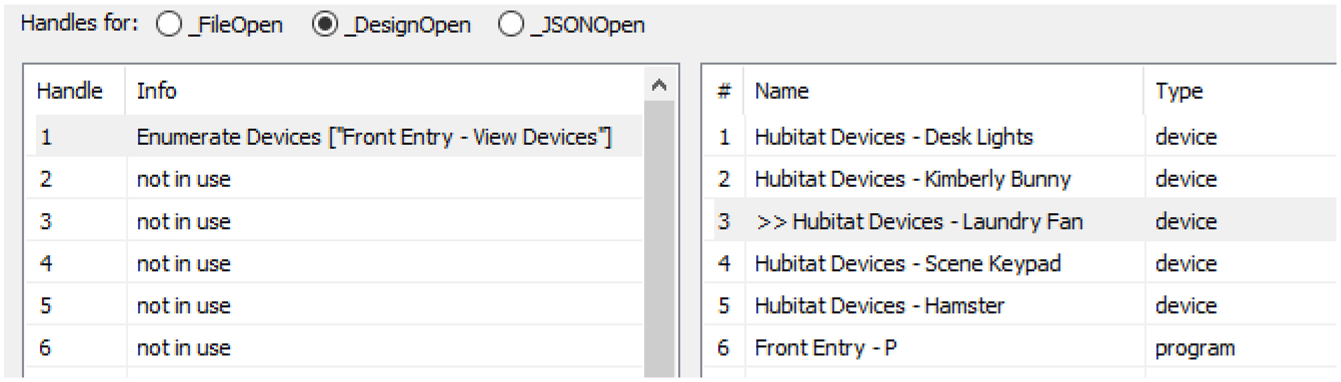
Handles for: (● _FileOpen) ○ _DesignOpen ○ _JSONOpen

Handle	Info
1	open write ["Front Entry - File op"]
2	not in use

#	Path
1	C:\HCAFiles\data reduction.txt
2	

The left-hand window shows the name of the program in square brackets that has opened the file, and if it was open for reading for writing. The right-hand window shows the path to the open file.

Exploring design handles



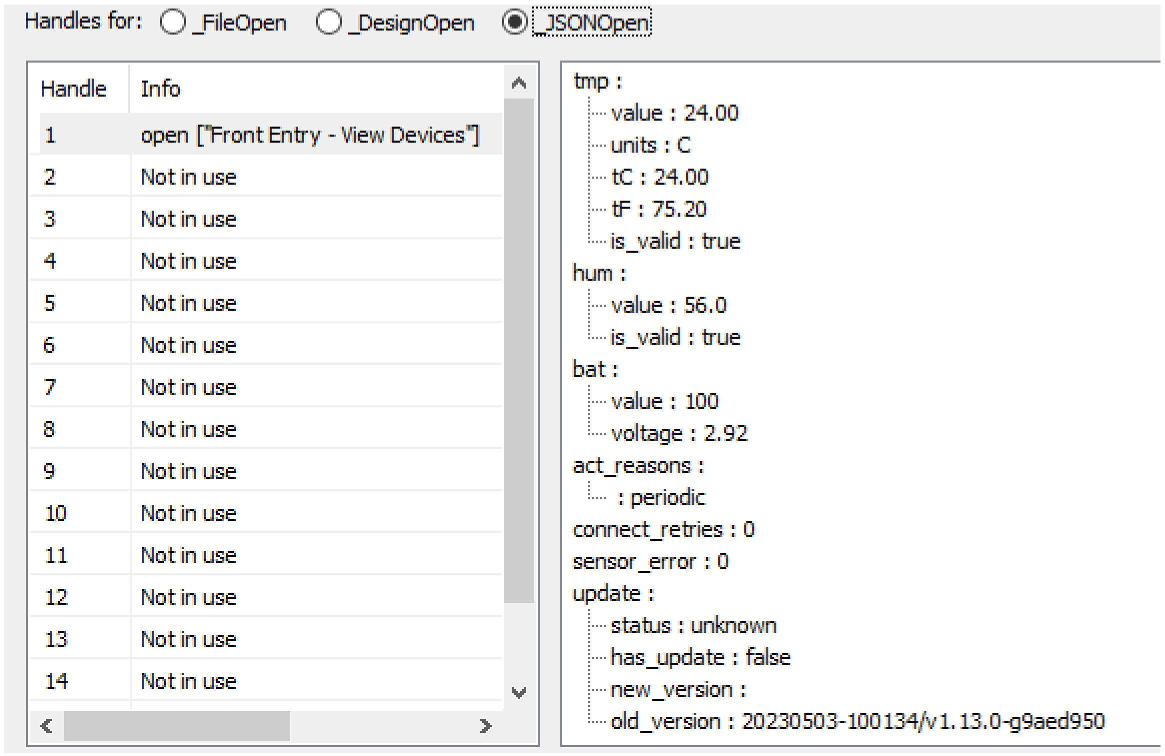
Handles for: ○ _FileOpen [✓ ● _DesignOpen] ○ _JSONOpen

Handle	Info
1	Enumerate Devices ["Front Entry - View Devices"]
2	not in use
3	not in use
4	not in use
5	not in use
6	not in use

#	Name	Type
1	Hubitat Devices - Desk Lights	device
2	Hubitat Devices - Kimberly Bunny	device
3	>> Hubitat Devices - Laundry Fan	device
4	Hubitat Devices - Scene Keypad	device
5	Hubitat Devices - Hamster	device
6	Front Entry - P	program

The left-hand window shows the name of the program in square brackets that opened the design handle. It also shows if any of the optional arguments were used to _DesignOpen. In this case the open used no optional arguments. The left-hand window shows the complete enumeration. The one selected in the list and marked with ">>" at the start is the value that _DesignName will return when it is next executed.

Exploring JSON handles



Handles for: ○ _FileOpen ○ _DesignOpen [✓ ● _JSONOpen]

Handle	Info
1	open ["Front Entry - View Devices"]
2	Not in use
3	Not in use
4	Not in use
5	Not in use
6	Not in use
7	Not in use
8	Not in use
9	Not in use
10	Not in use
11	Not in use
12	Not in use
13	Not in use
14	Not in use

```
tmp :
  ├ value : 24.00
  ├ units : C
  ├ tC : 24.00
  ├ tF : 75.20
  └ is_valid : true
hum :
  ├ value : 56.0
  └ is_valid : true
bat :
  ├ value : 100
  └ voltage : 2.92
act_reasons :
  └  : periodic
connect_retries : 0
sensor_error : 0
update :
  ├ status : unknown
  ├ has_update : false
  ├ new_version :
  └ old_version : 20230503-100134/v1.13.0-g9aed950
```

For JSON handles the name of the program that opened the handle is given in square brackets. In the right-hand window a display of the JSON.

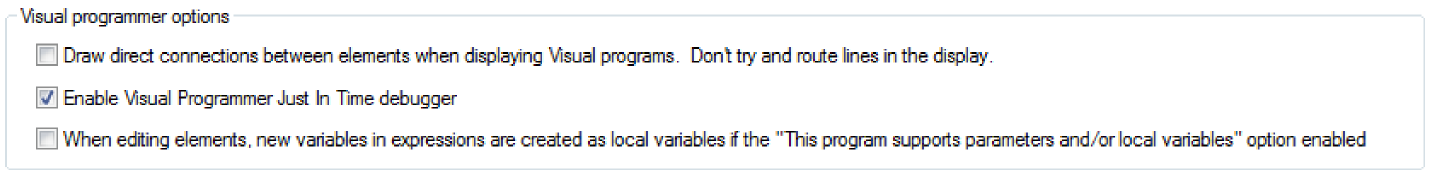
Finally, the explorer, when working with handles, has an option to close the selected handle.

Just in time debugging - JIT

You can use the debugger in two different ways: As described above you can start a program in the debugger, choose its starting trigger and parameter values, and then step though its execution.

HCA also has what is called "just in time debugging". The idea is that you set breakpoints in programs and then any time HCA during execution of that program encounters an element in a program with a breakpoint, the debugger starts, and you can then debug the program from that point on.

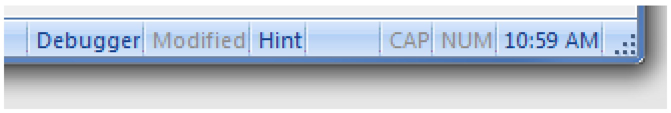
This feature must be enabled in HCA Options on the Visual Programmer tab.



Visual programmer options

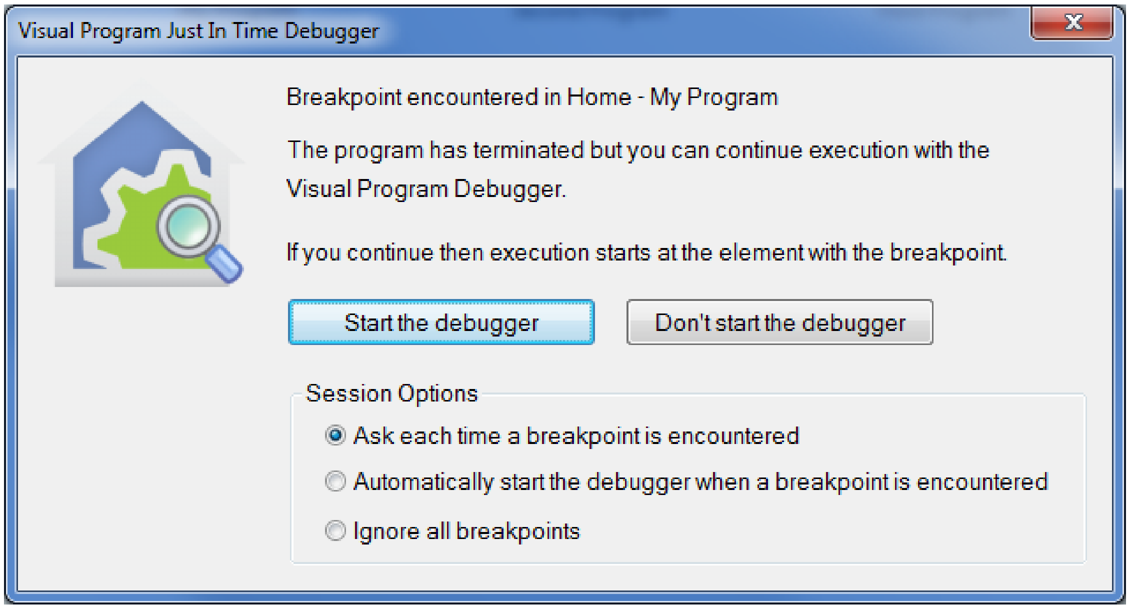
- [ ] Draw direct connections between elements when displaying Visual programs. Don't try and route lines in the display.
- [✓] Enable Visual Programmer Just In Time debugger
- [ ] When editing elements, new variables in expressions are created as local variables if the "This program supports parameters and/or local variables" option enabled

Once JIT is enabled, as a reminder in the main HCA Window status bar the "Debugger" indicator is turned "on".



Once JIT is enabled then any time a program is opened in the Visual Programmer the element popup menu contains the *Set* or *Clear* breakpoint choices. Using these menu choices you can set breakpoints on any element, close the Visual Programmer with OK, and then wait until the program is started by the normal program triggering mechanism. When breakpoints are encountered, the debugger takes over.

If JIT is enabled and an element with a breakpoint is encountered, this popup appears:


Visual Program Just In Time Debugger

Breakpoint encountered in Home - My Program

The program has terminated but you can continue execution with the Visual Program Debugger.

If you continue then execution starts at the element with the breakpoint.

Session Options

- [✓ Ask each time a breakpoint is encountered]
- Automatically start the debugger when a breakpoint is encountered
- Ignore all breakpoints

This popup tells you the name of the program with the breakpoint and lets you choose to start the debugger or not. It also lets you set options for the next time a breakpoint is encountered. That selection persists until you restart HCA.

If you are currently debugging a program and another program with a breakpoint is encountered, that breakpoint is ignored. Only one program can be in the debugger at a time. If an element has a breakpoint and JIT isn't enabled, then the breakpoint is ignored.

Making program changes while debugging

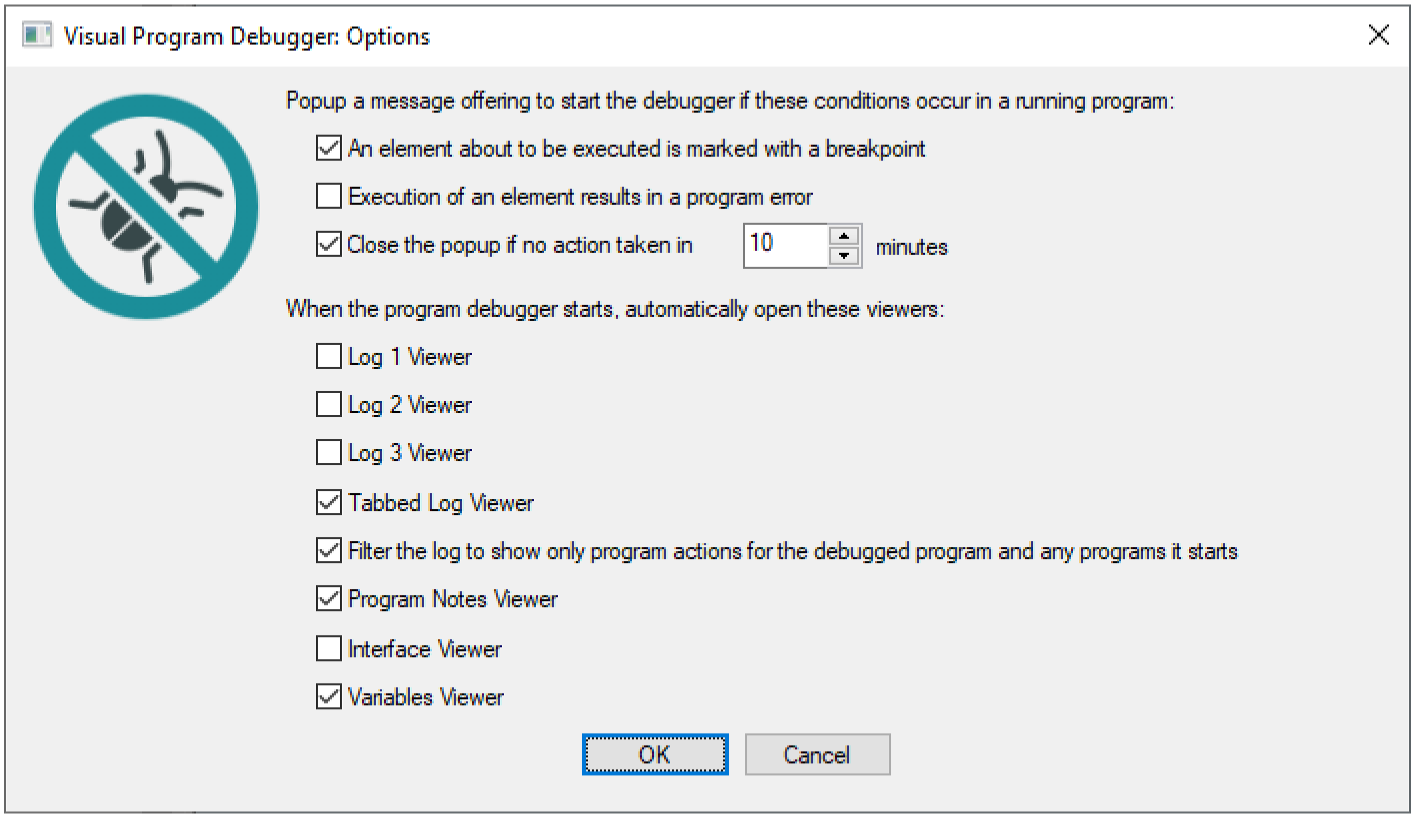
While debugging you can make changes to the program to correct problems. There are some limitations:

- You can only make changes to the program at the "stop of the stack".
- You can't change the Begin-Here element to modify program parameters.
- The changes must be saved if you want to continue execution in the debugger. The debugger prompts this when you begin execution with the *Start* or *Step* buttons.

Tip: It really isn't a good idea to make major changes while in the debugger. If you find you need to make major changes to the program, it is a good idea to leave the debugger and then use the normal HCA facilities to edit the program.

Configuring the debugger

From the "Tools" ribbon category select the "Options" button in the "Debugger" panel.


Visual Program Debugger: Options

Popup a message offering to start the debugger if these conditions occur in a running program:

- [✓] An element about to be executed is marked with a breakpoint
- [ ] Execution of an element results in a program error
- [✓] Close the popup if no action taken in 10 minutes

When the program debugger starts, automatically open these viewers:

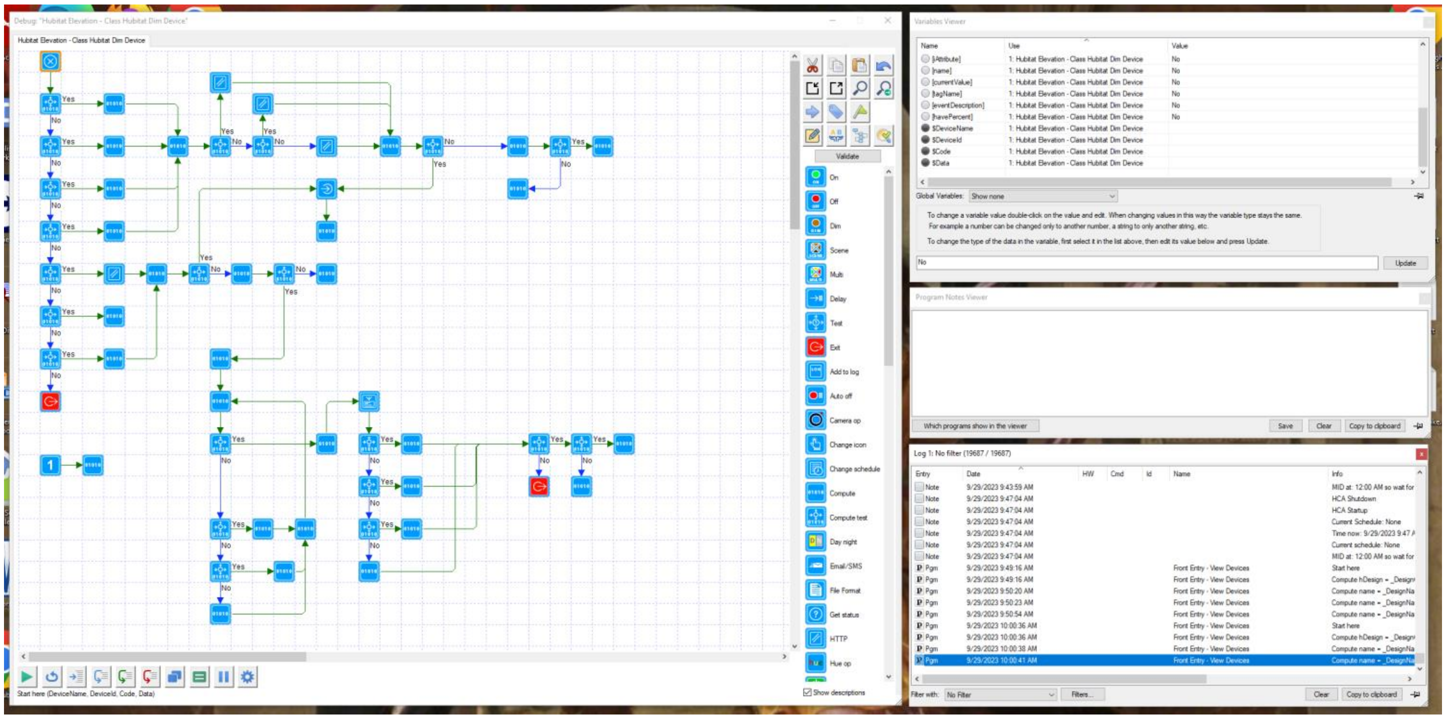
- [ ] Log 1 Viewer
- [ ] Log 2 Viewer
- [ ] Log 3 Viewer
- [✓] Tabbed Log Viewer
- [✓] Filter the log to show only program actions for the debugged program and any programs it starts
- [✓] Program Notes Viewer
- [ ] Interface Viewer
- [✓] Variables Viewer

The first set of options lets you control the action of the debugger if a program makes an error. The third option – "Close if no action taken…" is an easy way to make sure that if you are running HCA but not actively looking at it's display than the program that made the error can be simply terminated rather than having the debugger hang waiting for a response.

The next set of options allow you to configure the debugger when it starts. You can automatically open the log viewer(s), the programs notes viewer, the interface viewer, and the variables viewer. The 5th option has the debugger create a temporary filter so that the log shows only program elements from the currently being debugged program.

The advantage of this is that the debugger remembers the size and placement of these windows from when the debugger was last used. This way you can create a customized layout of what you need, with the viewer windows sized and positioned as you want them each time the debugger starts.

The screen image below, vastly reduced to include it here, shows the debugger open with the program in the large box on the left, and on the right the program variables, notes viewer, and log 1. Each time a program is debugged this same arrangement and placement of windows is automatically opened after you configured that in the debugger options.



Name	Use	Value
[Attribute]	1: Hubitat Elevation - Class Hubitat Dim Device	No
[name]	1: Hubitat Elevation - Class Hubitat Dim Device	No
[currentValue]	1: Hubitat Elevation - Class Hubitat Dim Device	No
[tagName]	1: Hubitat Elevation - Class Hubitat Dim Device	No
[eventDescription]	1: Hubitat Elevation - Class Hubitat Dim Device	No
[havePercent]	1: Hubitat Elevation - Class Hubitat Dim Device	No
$DeviceName	1: Hubitat Elevation - Class Hubitat Dim Device	
$DeviceId	1: Hubitat Elevation - Class Hubitat Dim Device	
$Code	1: Hubitat Elevation - Class Hubitat Dim Device	
$Data	1: Hubitat Elevation - Class Hubitat Dim Device	

Global Variables: Show none

To change a variable value double-click on the value and edit. When changing values in this way the variable type stays the same. For example a number can be changed only to another number, a string to only another string, etc.

To change the type of the data in the variable, first select it in the list above, then edit its value below and press Update.

No

Log 1: No filter (19687 / 19687)

Entry	Date	HW	Cmd	Id	Name	Info
Note	9/29/2023 9:43:59 AM					MID at: 12:00 AM so wait for
Note	9/29/2023 9:47:04 AM					HCA Shutdown
Note	9/29/2023 9:47:04 AM					HCA Startup
Note	9/29/2023 9:47:04 AM					Current Schedule: None
Note	9/29/2023 9:47:04 AM					Time now: 9/29/2023 9:47 A
Note	9/29/2023 9:47:04 AM					Current schedule: None
Note	9/29/2023 9:47:04 AM					MID at: 12:00 AM so wait for
Pgm	9/29/2023 9:49:16 AM				Front Entry - View Devices	Start here
Pgm	9/29/2023 9:49:16 AM				Front Entry - View Devices	Compute hDesign = _Design
Pgm	9/29/2023 9:50:20 AM				Front Entry - View Devices	Compute name = _DesignNa
Pgm	9/29/2023 9:50:23 AM				Front Entry - View Devices	Compute name = _DesignNa
Pgm	9/29/2023 9:50:54 AM				Front Entry - View Devices	Compute name = _DesignNa
Pgm	9/29/2023 10:00:36 AM				Front Entry - View Devices	Start here
Pgm	9/29/2023 10:00:36 AM				Front Entry - View Devices	Compute hDesign = _Design
Pgm	9/29/2023 10:00:38 AM				Front Entry - View Devices	Compute name = _DesignNa
Pgm	9/29/2023 10:00:41 AM				Front Entry - View Devices	Compute name = _DesignNa

##end##