Chapter 12
Visual Program Debugger

In the previous chapter on programs a section titled *Getting programs to do what you want* discussed using the log to trace how programs execute. That is a useful technique but it would be much better if you could execute each element of a program, one at a time, and follow the progress of the program that way.

HCA contains a tool to do just that called the *Program Debugger*.

If you are familiar with any program development environment that has a debugger then the facilities the HCA debugger provides will be familiar.

With the HCA debugger you can:

- Execute one element at a time
- Set breakpoints. A breakpoint set on an element causes the program to stop before that element executes and turns control over to the debugger
- Set the element about to be executed so you can change what the program does next
- Examine flag values and modify them

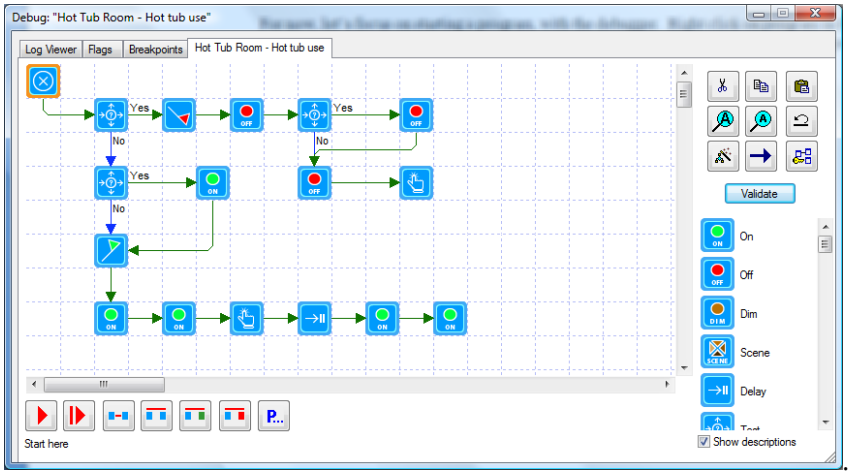
There are two different ways to interact with the debugger. A program can be started by the debugger so you have control right from the start. Or, in what is known as "just in time debugging", the program can start in the normal course of events by some trigger or schedule and then when it encounters an element with a breakpoint, control is turned over to the debugger.

The program debugger operates in either stand-alone or client-server mode.

Using the Debugger: The basics

Before describing in detail all the features of the Debugger, let's just jump into using it so you can see how it works.

For now, let's focus on starting a program with the debugger. Right click on program in the design pane, or on the icon for a program in the display pane and select *Debug* from the popup menu. The debugger opens.



The first thing to note about the debugger is that it looks a lot like the Visual Programmer. The debugger dialog has the same programming canvas, the same tool panel, and the same list of elements.

There are two differences you can immediately see: the seven buttons at the bottom and the fact that the Start Here element has an orange border.

The element with the orange border is what is called the *current element* and is the element that has not yet executed but when the debugger is told to proceed, it is the element first executed. At the bottom of the dialog the text image of the current element is shown. Unlike the Visual Programmer this text always matches the current element unless you place the mouse over another element and then it shows that element until you move the mouse off.

The debugger window

The next thing to note about the debugger window is that it is a separate window from the HCA application window. You can minimize it separately from the HCA window. While it is open you can still perform other actions with HCA: View and modify the properties for devices, groups, and programs, look at schedules, etc.

There is one main limitation with what you can do while the debugger is open and that is discussed in a later section.

Debugger Buttons

The buttons at the bottom of the debugger dialog control the actions of the debugger. They are from left to right:

Run until breakpoint. Runs the program as normal from the current element until the program either completes or the program encounters an element with a breakpoint. This is called the **Run** button.

Restart from Begin-Here. Changes the current element to be the Begin-Here element.

Execute current. As it says, executes the current element and then determines a new current element. If the element executed is a *Test* element then the next element is based upon the test outcome. This is also called the ***Step*** button.

Don't execute current. In this case the current element is not executed and the next element is determined as usual. This and the next two buttons are collectively known as the ***Skip Over*** buttons since they skip execution of an element.

Don't execute and take the YES path. Same as the *don't execute current* button except when the current element is a *Test* element. In that case the first element in the *Yes* path becomes the current element.

Don't execute and take the NO path. Same as the *don't execute current* button except when the current element is a *Test* element. In that case the first element in the *No* path becomes the current element.

Open program. Opens a dialog where you can select a program so you can set breakpoints in it. This can be useful if the program you are debugging uses the *Start-Program* element to execute another program and you want to set a breakpoint in that program before the *Start-Program* element starts it.

Debugger Tabs

At the top of the debugger dialog are a number of tabs. These are:

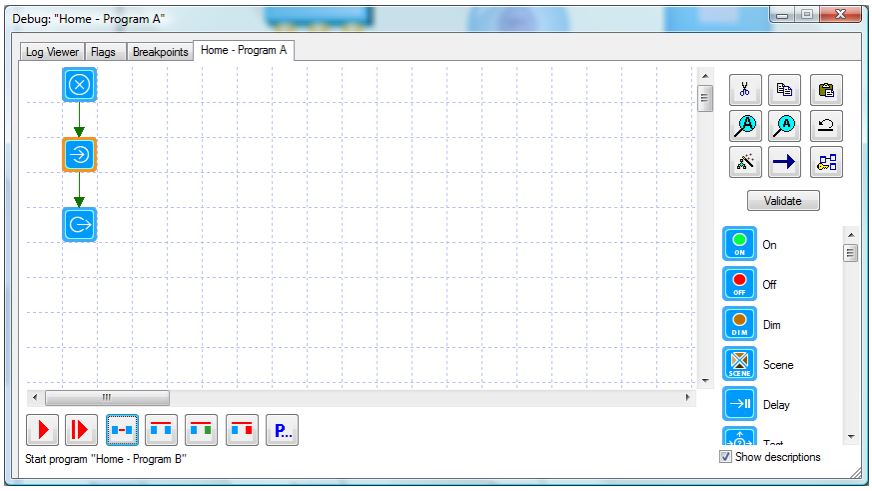
Log Viewer. A similar log viewer that opens from the ribbon *Control* category. Having it as a tab in the debugger may make managing multiple windows simpler.

Flag Viewer. A simplified viewer for flags with the ability to change their current value.

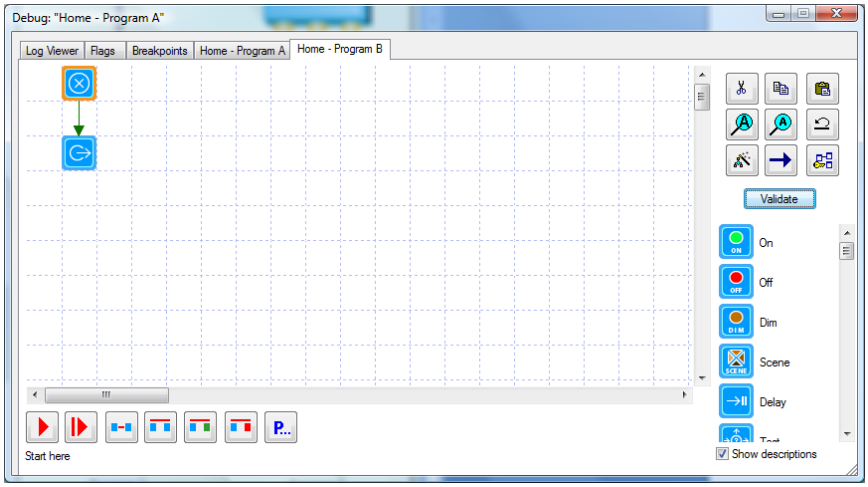
Breakpoints. A list of all breakpoints in all programs and on what elements they are set. From this list the breakpoints can be disabled and enabled.

The next tab is the program being debugged. Programs, as described in the previous chapter can start another program using the *Start-Program* element. In a traditional programming language it would be like a *subroutine* invocation.

If you use the *Start-Program* element then the program that is started appears in an additional tab and the current element is show in that tab. For example, suppose program A uses the Start-Program element to start program B. If program A is started in the debugger then the debugger window looks like this:



Press the *Step* button and the current element becomes the *Start-Program* element. Press the *Step* button again and the *Start-Program* element executes and a tab for program B appears.



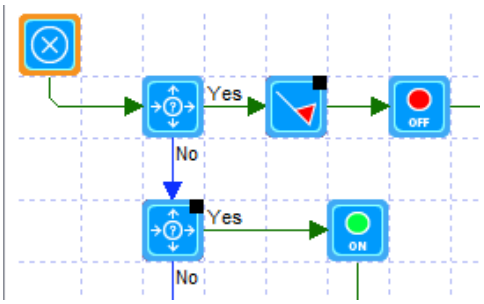
Note that the current element for program B is the *Begin-Here* element. If you switch back to the P*rogram A* tab, the current element is still at the Start-Program element since it has not yet completed.

Again, in a traditional programming language, the tabs at the top of the debugger window represent the execution *stack* with the program at the bottom of the stack showing in the left-most tab and the program at the top of the stack showing in the right-most tab.

If you continue to use the *Step* button, Program B finishes and the tab for it disappears.

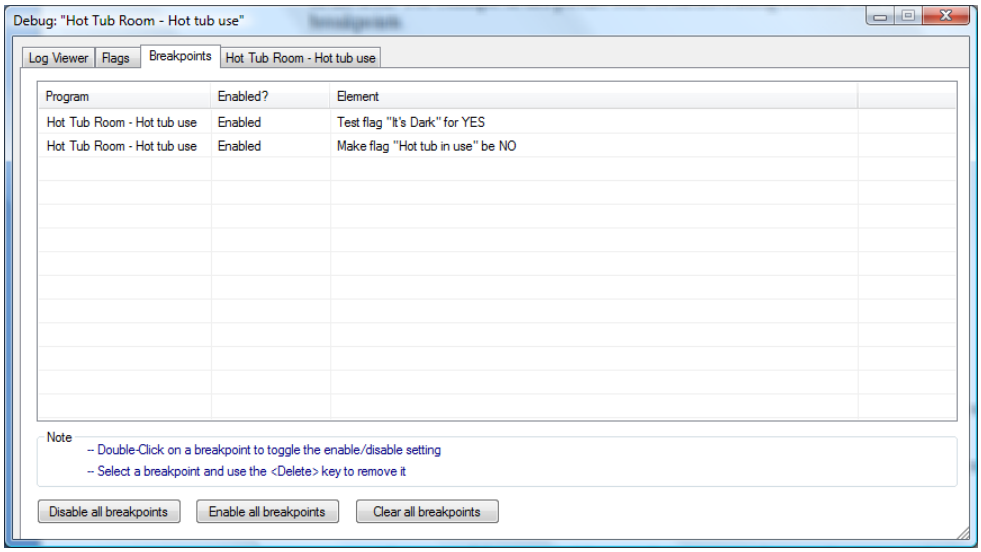
Breakpoints

When a program is about to be executed and a breakpoint has been set on that element then execution stops. You can tell an element that has a breakpoint by a black square in the upper right of the icon. For example in this picture both elements leading from the Test element have breakpoints.



To set a breakpoint on an element, right-click on it and choose *Set Breakpoint* from the popup menu. If the element already has a breakpoint the menu selection is *Clear Breakpoint* which removes the breakpoint.

If you want to see a list of all breakpoints, select the *Breakpoints* tab.

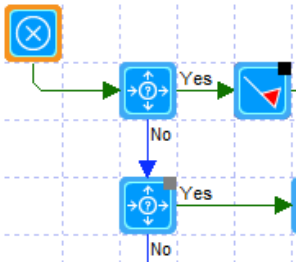


In addition to being set on an element, a breakpoint can be enabled or disabled. When you first add a breakpoint it is automatically enabled. If you no longer want execution to stop at that element you can disable or remove the breakpoint.

Why disable a breakpoint but and not remove it? Sometimes during debugging you may want to temporarily disable a breakpoint rather than remove it. By doing that you are saved the trouble of opening the program again and reestablishing the breakpoint. Also, enable and disable can be simply done form the breakpoint list.

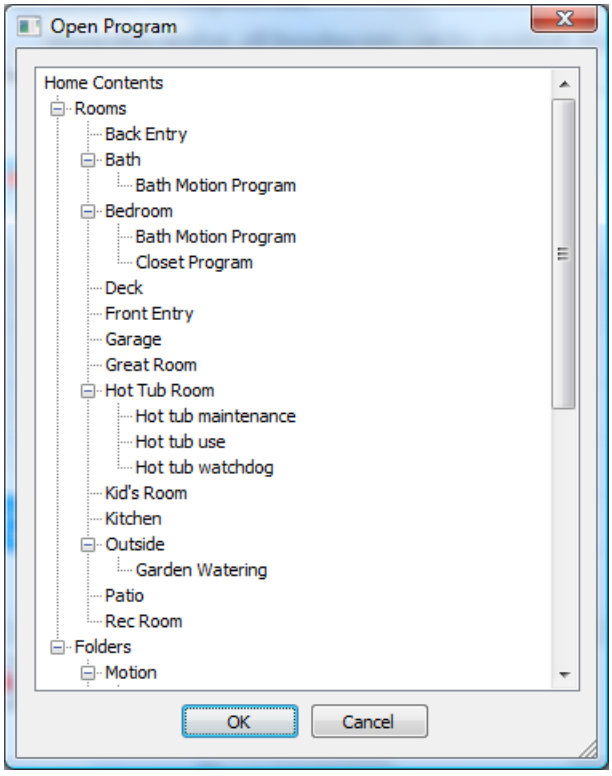
In addition to working with a single breakpoint, all breakpoints can be enabled, disabled, or cleared using the buttons at the bottom on the *Breakpoints* tab.

When a breakpoint is disabled the black square is shown in a lighter color.



The *No* path from the test element has a disabled breakpoint. The *Yes* path element has an enabled breakpoint.

If you want to set a breakpoint in a program that is not the program being debugged, press the *Programs* button – the right-most button with the "P" below the programming canvas. A list of all programs in the design opens.

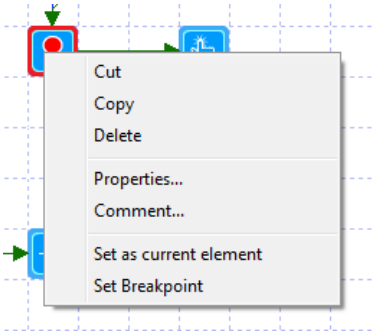


Choose a program and press OK. That program's properties dialog opens and you can locate and set a breakpoint on one or more elements. As before, right-click on the element and choose *Set Breakpoint* from the popup menu.

Changing the current element

Sometime during debugging you want to change the current element. This could be because you want to move to a different point in the program to test the elements there or to skip over elements that you don't want to test.

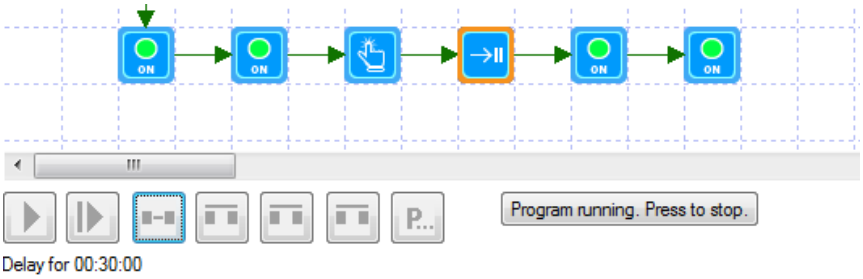
To change the current element, right-click on the element you want and select *Set as Current Element* from the popup menu.



Fast and Slow elements

Elements when executed in the debugger are classified into *slow* and *fast* elements. For example, an ON element is fast since it quickly queues for transmission an ON message to the device. A Delay element is a slow element since it can take minutes or hours to complete depending upon its settings.

When a fast element executes, the next element is selected usually so quickly that you probably will not notice a slight pause. But when a slow element executes then the debugger changes to show an extra button.



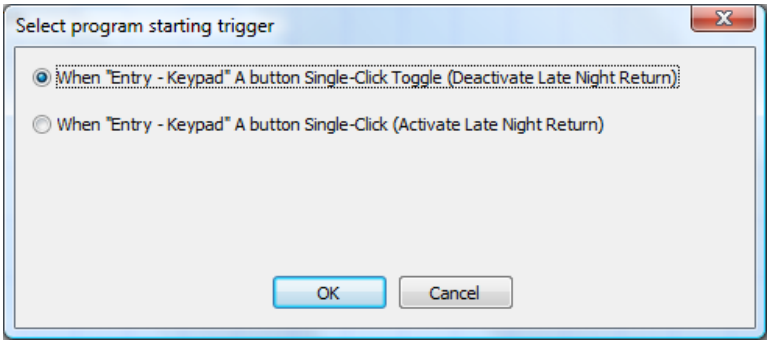
Note that the buttons to control the debugger have been disabled and an additional button has appeared. This button when pressed aborts the execution of the current element – in this example a *Delay* element. Once you stopped execution of the current element you can move to the next element by using the *Skip* button or by changing the current element as described above.

If you stop execution of the current element and then press the *Step* or *Run* button that element starts executing again.

Setting the starting trigger

As described in the previous chapter, programs are started by triggers and a program can have many different triggers to start it.

If a program with multiple triggers is run in the debugger, when the *Run* or *Step* button is pressed when *Begin-Here* is the current element then a popup appears to let you select the starting trigger.



This program has two triggers. If there were other triggers then additional options or a dropdown would be shown so you can select the trigger to use.

Selecting a starting trigger is important if the program tests to see which trigger started it. Using the debugger in this way you can quickly test all the various triggers even those that don't often occur.

Modifying programs

When in the debugger you can modify the program being debugged. You are free to delete elements, add elements, change the connecting lines between elements, and modify the element properties.

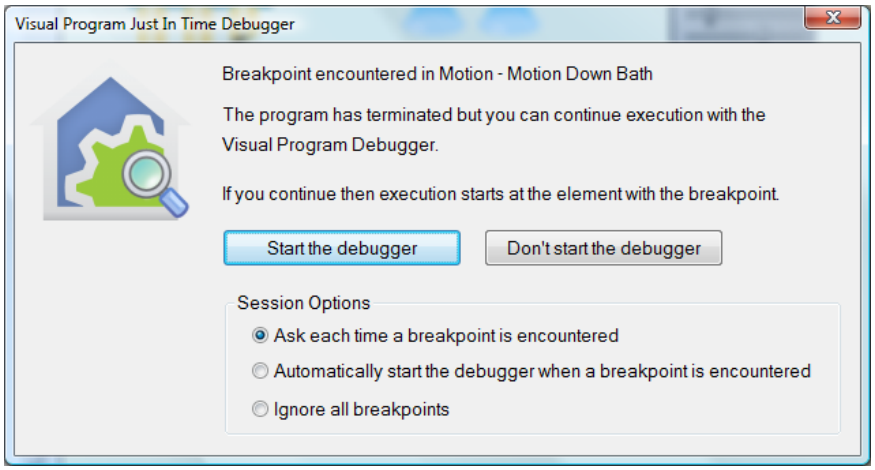
If you do make a change then when execution continues – you press the *Step* or *Run* buttons – a popup asks you to save the modified program.

Just in time debugging

In all the above discussion the debugger was started by selecting *Debug* from a right-click popup menu on a program name or icon. In addition to this method you can instead use what is called the *Just In Time* debugger.

The *Just In Time* debugger is the same debugger as described above but rather than starting the program in the debugger, you instead use the debugger to set breakpoints in one or more programs then close the debugger. Those breakpoints remain set. As your design is run by HCA, programs start and stop, schedules are monitored, etc. If a program encounters an element with one of the breakpoints you have previous set, then the debugger starts and you can debug from that point forward.

When a breakpoint is encountered in this manner then a popup dialog appears:

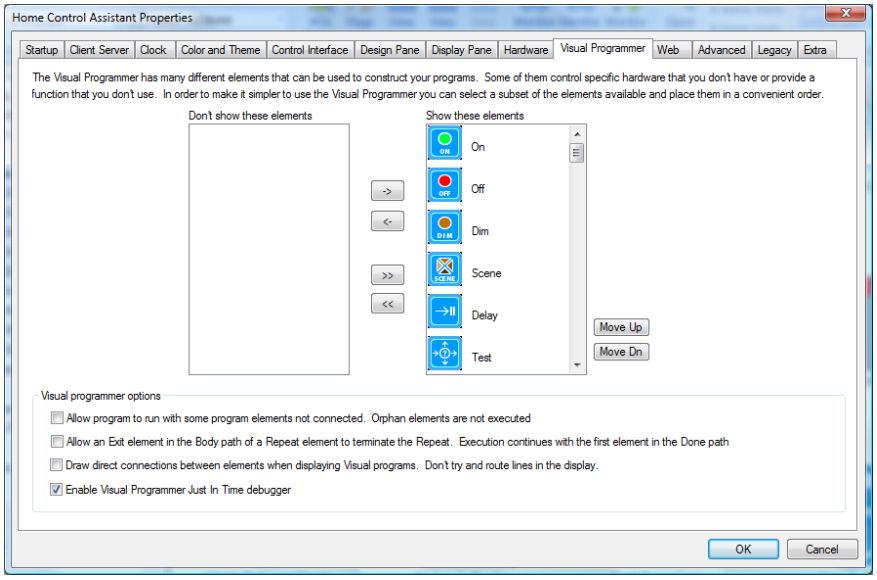


If you press the *Start the debugger* button then the debugger opens and the current element – the one with the breakpoint that has not yet been executed –is designated with the orange border showing it is the current element.

If you don't chose to start the debugger then the program is stopped - it doesn't continue.

The *Session Options* settings in the popup lets you configure the action of this popup if subsequent breakpoints are encountered. These stay in effect until HCA is restarted.

Before you can use *Just In Time* debugging it must be enabled in the HCA options on the *Visual Programmer* page. It is the last option in the *programmer options*.



Debugging Limitation

There is one major limitation of program debugging: Only one program can be debugged at a time. Normally this is not a problem but if you have several programs with breakpoints set then you should be aware of this limitation.

Suppose you have two programs – A and B – and both have breakpoints set. Program A is triggered and the debugger starts. Assume that program B is also triggered.

Because of the limitation of being able to debug only one program at a time, while program A is being debugged the breakpoint in program B is ignored and program B just continues as if it had no breakpoints.