HCA Tech Note 116

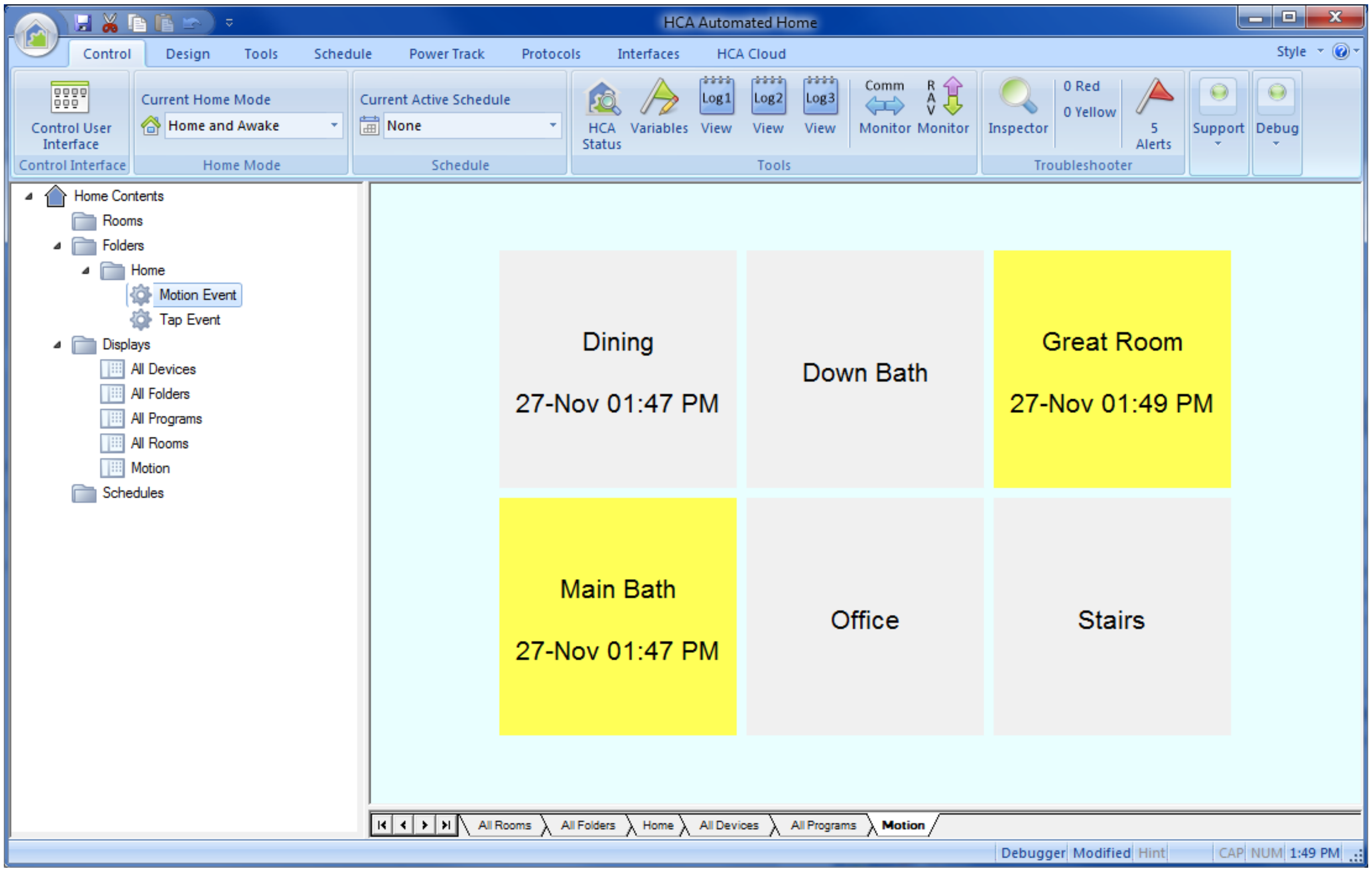
Parameterized Triggers

Other technical notes discuss using parameters to supply arguments when one program uses the "Start-Program" element to start another program. But you can also provide arguments when a program is started by a trigger. Here is a short example that shows how this feature can make it easy to do a complex operation in an easily extensible manner

The problem to solve:

The problem is to create a tiled display with a tile for each motion sensor in a home. The tile should have the name of the motion sensor – the room it is in – and show if it is ON (room occupied) by drawing the tile in yellow or OFF (room unoccupied) in grey. Also, it should show the date/time of the last reception from the sensor.

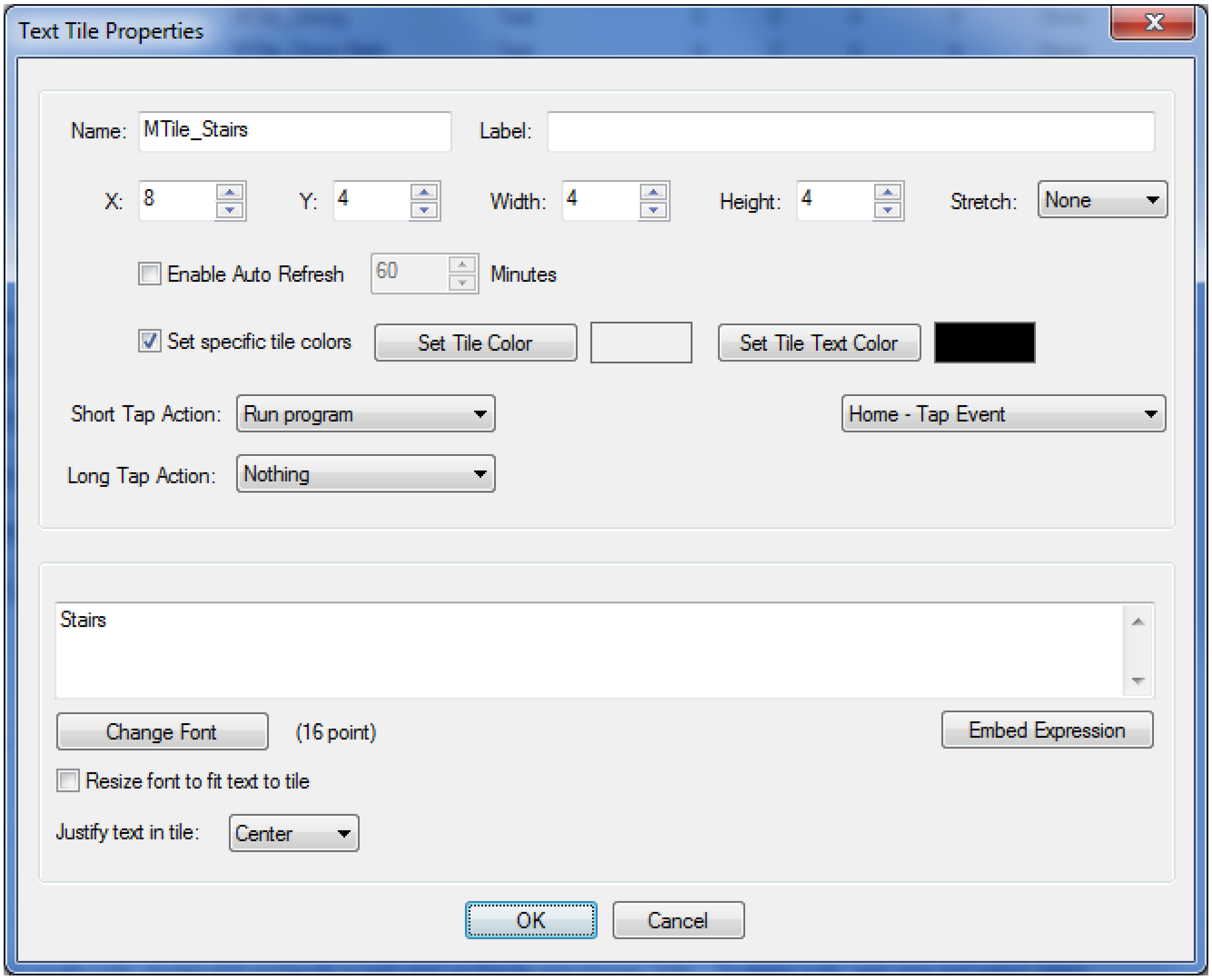
Like this:



The Solution:

The first part of the solution is to create the tiled display and add the six text tiles to it. There really isn't much new here. Each tile is given a name that starts with "MTile_" followed by the text that is shown in the tile. Giving a tile a name allows us to refer to it in a program.

Each tile has a configuration like this:



Name: MTile_Stairs
Label:
X: 8 Y: 4 Width: 4 Height: 4 Stretch: None
☐ Enable Auto Refresh 60 Minutes
☑ Set specific tile colors

Short Tap Action: Run program — Home - Tap Event
Long Tap Action: Nothing

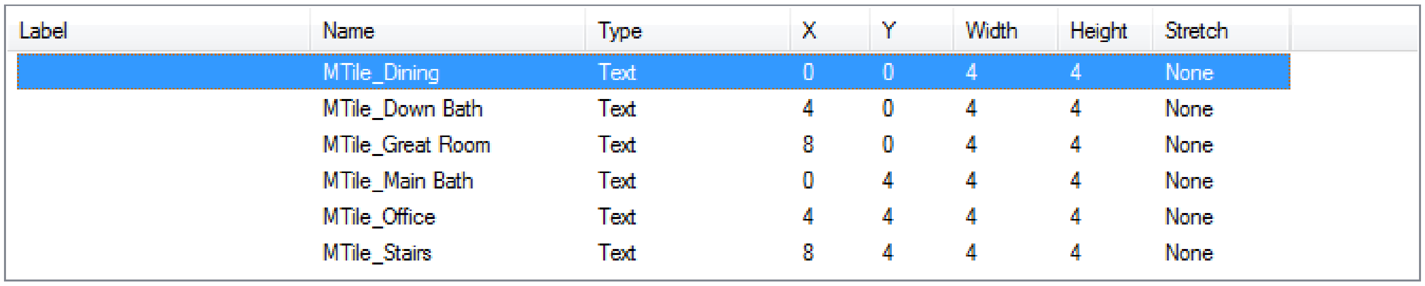
Stairs

(16 point)

☐ Resize font to fit text to tile
Justify text in tile: Center

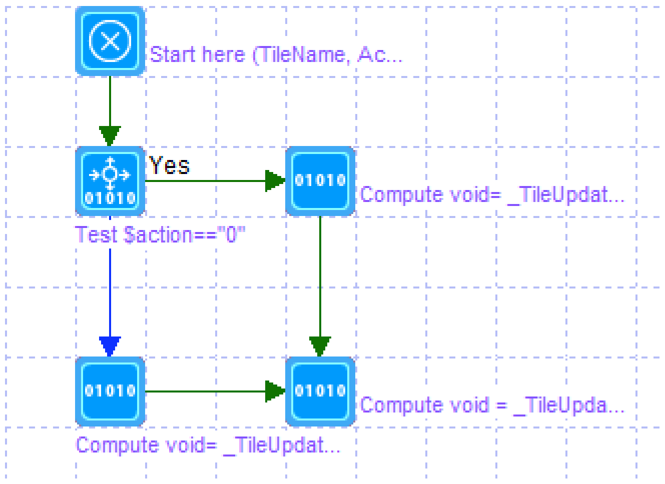
We will come back to the program that is run when the tile is tapped later. For now, the important parts in the configuration are that the tile color and is set to the gray we want to show when OFF and the text color is set to black. The tile has no label.

When all six tiles are created we have this:

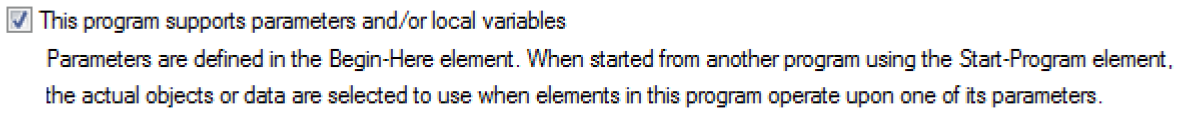


Label	Name	Type	X	Y	Width	Height	Stretch
	MTile_Dining	Text	0	0	4	4	None
	MTile_Down Bath	Text	4	0	4	4	None
	MTile_Great Room	Text	8	0	4	4	None
	MTile_Main Bath	Text	0	4	4	4	None
	MTile_Office	Text	4	4	4	4	None
	MTile_Stairs	Text	8	4	4	4	None

But as I said there really isn't anything all the interesting here so let's turn to the program called "Motion Event". Here is the program and it surprisingly has only a few elements.


Start here (TileName, Ac...
Yes
Compute void= _TileUpdat...
Test $action=="0"
Compute void = _TileUpda...
Compute void= _TileUpdat...

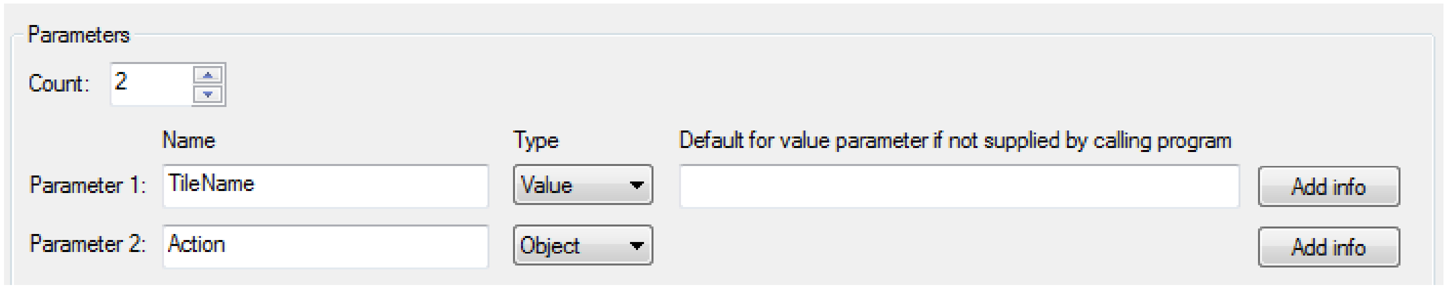
Before we look at how the program works let's look at the Advanced tab, then the properties of the "Start Here" element, and the triggers tab. The Advanced tab enables the option that says this program has parameters:



[✓] This program supports parameters and/or local variables

Parameters are defined in the Begin-Here element. When started from another program using the Start-Program element, the actual objects or data are selected to use when elements in this program operate upon one of its parameters.

The Begin-Here element defines the number and names of what those parameters are.



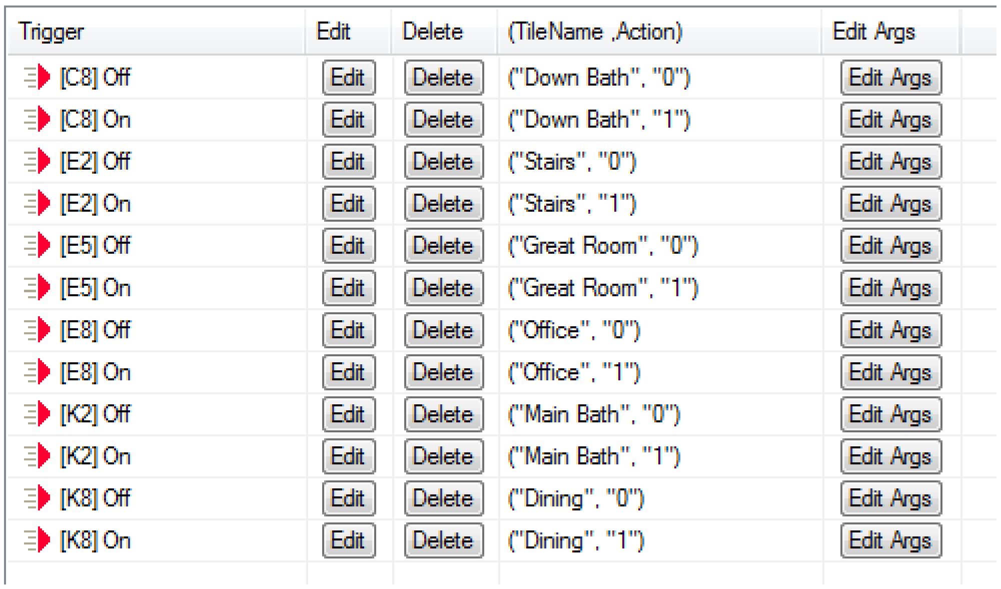
Parameters

Count: 2

	Name	Type	Default for value parameter if not supplied by calling program	
Parameter 1:	TileName	Value		Add info
Parameter 2:	Action	Object		Add info

There are two parameters, and each is a “value” not an “object”.

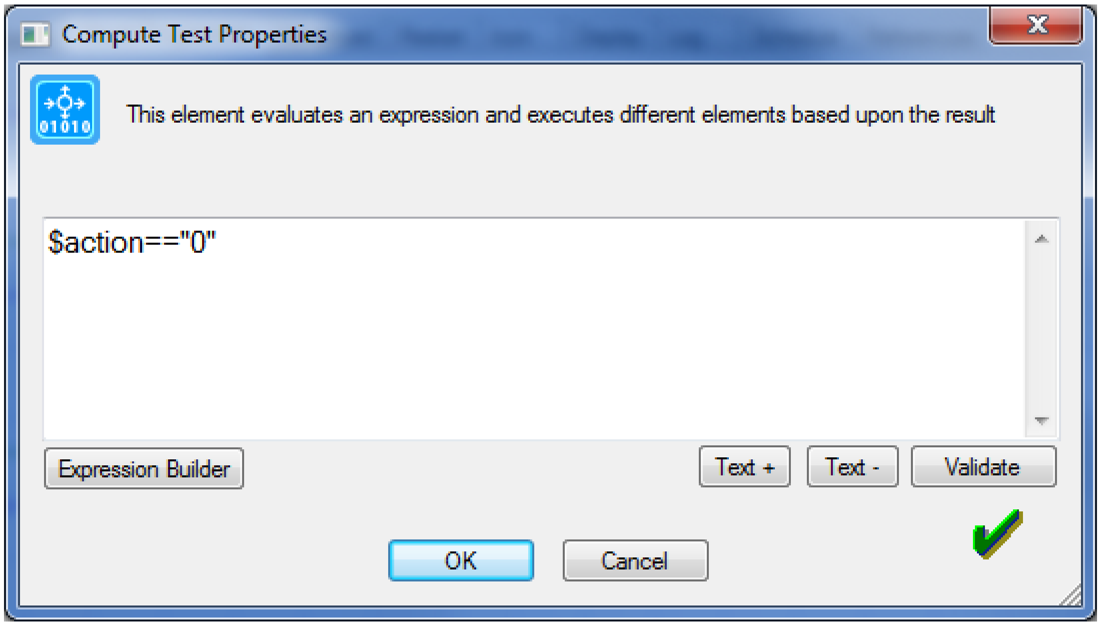
Next, we turn to the Triggers tab where the real work is:



Trigger	Edit	Delete	(TileName ,Action)	Edit Args
[C8] Off	Edit	Delete	("Down Bath", "0")	Edit Args
[C8] On	Edit	Delete	("Down Bath", "1")	Edit Args
[E2] Off	Edit	Delete	("Stairs", "0")	Edit Args
[E2] On	Edit	Delete	("Stairs", "1")	Edit Args
[E5] Off	Edit	Delete	("Great Room", "0")	Edit Args
[E5] On	Edit	Delete	("Great Room", "1")	Edit Args
[E8] Off	Edit	Delete	("Office", "0")	Edit Args
[E8] On	Edit	Delete	("Office", "1")	Edit Args
[K2] Off	Edit	Delete	("Main Bath", "0")	Edit Args
[K2] On	Edit	Delete	("Main Bath", "1")	Edit Args
[K8] Off	Edit	Delete	("Dining", "0")	Edit Args
[K8] On	Edit	Delete	("Dining", "1")	Edit Args

For this example, I used inexpensive X10 motion sensors, but this same technique will work for any protocol devices. The key here is that for each trigger you can specify what the values are that the parameters take on when the program is run. For each trigger we provide two values: The name of the tile used for that motion sensor and the action – “0” for OFF, and “1” for ON.

The first element in the program after “Begin-Here” is a Compute-Test element like this:



This element evaluates an expression and executes different elements based upon the result

```
$action=="0"
```

What this does is to see if we are handling the case for when a motion sensor sends an ON or OFF. In an expression, whenever you refer to a parameter you must proceed its name with a “$” so the HCA knows that it is a parameter and not a variable.

Let’s follow the “Yes” path – when the action is “0” - when the motion sensor goes off. The first element on that path is a Compute element containing this expression:

```
void= _TileUpdate("MTile_" + $tileName, 1, _RGB(240, 240, 240), _RGB(0, 0, 0));
```

Lots going on here! First, we are using the _TileUpdate function which, from the documentation, we see that it takes 3 or 4 arguments. These are:

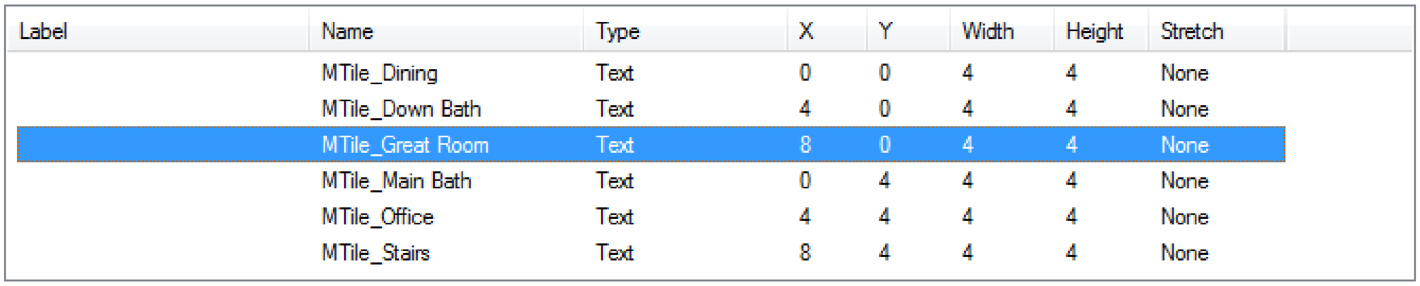
1. The name of the tile to update
2. A code that says what to do. In this case, code “1” is to change the tile colors
3. Because it is code 1, the 3rd argument is the tile color
4. Because it is code 1, the 4th argument is the tile text color

Rather than specify the colors in some hex number the _RGB function is used to compose the colors wanted. In the case of the background color it is a gray and the text color is black.

The name of the tile is composed from the argument - $tileName – prefixed with “MTile_”. Let’s take an example and you should refer to the triggers list on the previous page. Suppose that an E5 OFF arrives, the program starts with these values:

- TileName = “Great Room”
- Action = “0”

The first argument to the TileUpdate function is “MTile_Great Room” after the string is assembled, which matches the name of a tile in the display:



Label	Name	Type	X	Y	Width	Height	Stretch
	MTile_Dining	Text	0	0	4	4	None
	MTile_Down Bath	Text	4	0	4	4	None
	MTile_Great Room	Text	8	0	4	4	None
	MTile_Main Bath	Text	0	4	4	4	None
	MTile_Office	Text	4	4	4	4	None
	MTile_Stairs	Text	8	4	4	4	None

Yu may wonder why we didn’t just name the tiles the same text as we wanted to display and not have this “MTile_” stuff. The reason is that tile names are global to your whole design. So, if you have many tiled displays it may become confusing as to what tile is on what display. The way to handle this is to come up with and use a “Naming Convention”. In this case I decided that the times on the “Motion” display would have names that start with “MTile_”.

When this element executes the tile color changes to gray. The next element in this path is another Compute element:

```
void = _TileUpdate("MTile_" + $TileName, 3, $TileName + _chr(13) + _chr(10) + _chr(13) +
_chr(10) + _FormatTime(_now(), "$d-$b $I:$M $p"));
```

Again, this one is a bit complex and uses the UpdateTile function but in this case, from the documentation, a code of “3” is used to change the text in the tile.

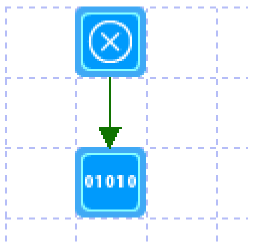
The name of the tile is assembled in the same manner as before. The text to display in the tile is also composed from the tile name from the trigger and the current date/time formatted into a short string. The Update-Tile element when changing the text will break the text across multiple lines if each part is separated from the other by a carriage return - _chr(13) – and newline - _chr(10). (Not as well documented as it should be!)

When this element executes, the text in the tile changes to show the tile name – unchanged – and the date-tike of the last reception.

In the program, the "Action == "1" path is similar, the only different being that the tile color changes to yellow.

The nice part about this comes from the parameterized trigger. One program is all that is needed for keeping any number of tiles updated. And because of the parameterized triggers, ***the program does not have to change*** if an additional motion sensor is added or one removed. Only a new trigger need be added and, of course, a new tile on this display.

The second program in this example uses many of the same techniques. When the tiles were created as part of the configuration it was specified that when the tile was tapped the program "Tap Event" would execute. This program does this:

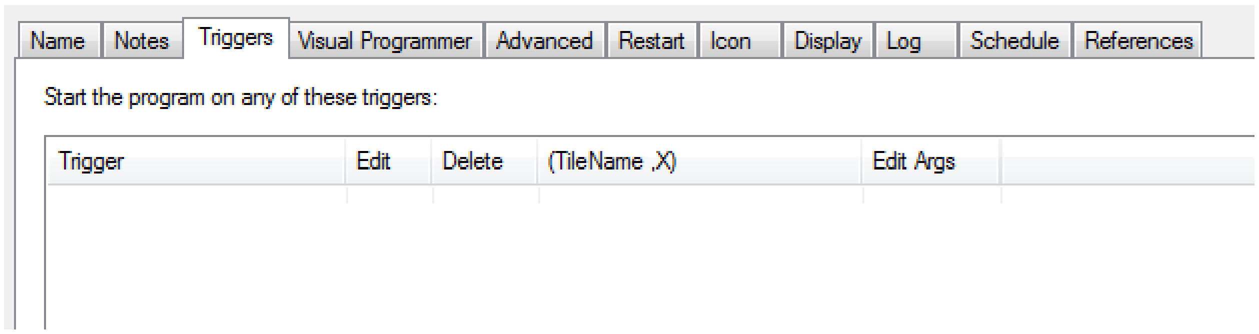


Only one element besides Start-Here! The Compute element contains this expression:

void = _TileUpdate($TileName, 3, _Right($TileName, _Len($TileName)-6));

Another Tile-Update, and this time it is also changing the text in the tile. What it does is to display the tile name and not show the last reception time – so you can clear that it you wanted. There is a $TileName parameter like before and it uses the string functions to set the text to the tile name without the "MTile_" prefix.

But where does the value assigned to $TileName come from? Unlike the other program not from a trigger. Look at the triggers list and nothing is there!



Start the program on any of these triggers:

Trigger	Edit	Delete	(TileName ,X)	Edit Args

When a program is started from a tap on a tile, it automatically gets two values passed to it: The tile name and the tile label. Why? It just is a feature of how HCA works. In this case since it gets the tile name it can use that to update the appropriate tile – that is, the tile just taped on.

Finishing Up…

The key to this example is knowing that you can provide parameters to triggers and in doing so really cut down the work load of the program. An alternative implementation could have used a series of TEST elements looking at the starting triggers and then using Tile-Update elements. But that would make the program much larger and more difficult to maintain.

Hope this helps!

##end##