Tile Builder Help
Version 1.3.0

Revised 6/10/23

Table of Contents

Section 1

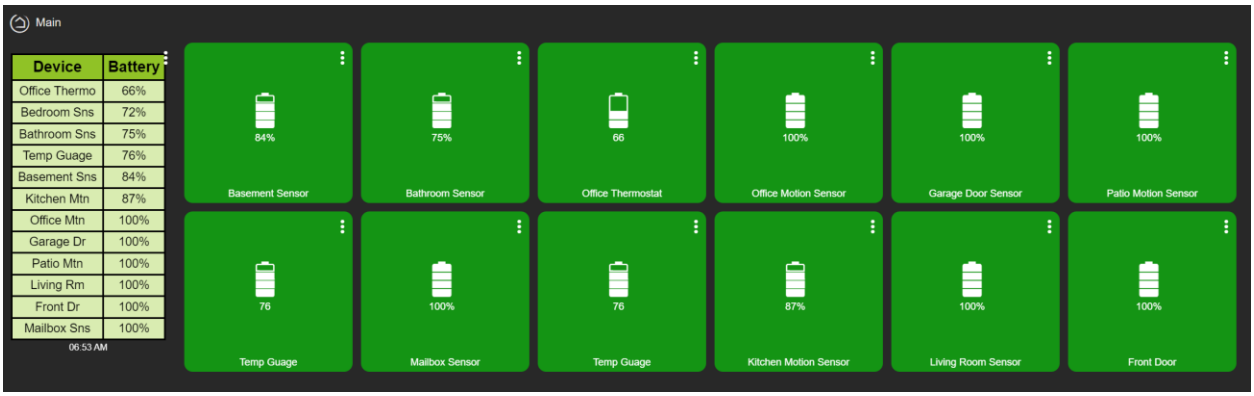
Tile Builder

Introduction and Installation

Introduction

Tile Builder is a novel way of presenting data on a Hubitat® Dashboard. Rather than each tile being a single unique device, **Tile Builder** allows data from multiple devices to be presented on the same tile in a highly customizable tabular format.

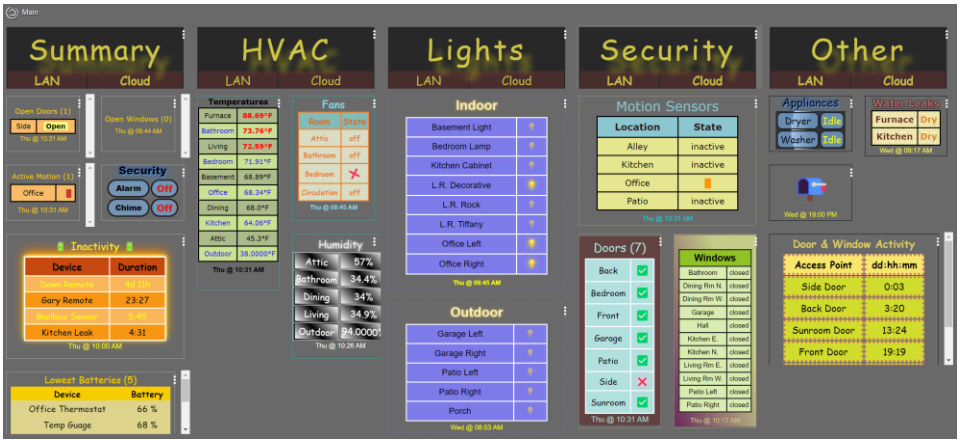
It is entirely native to Hubitat® and does not require any third-party elements or knowledge of CSS to achieve quite impressive results. The image below shows one of these tables vs the traditional method of using individual tiles.



Main Benefits of Tile Builder

- A native solution for Hubitat® dashboards that is easy to use and looks great.
- Full control over color, style, placement, effects of tiles etc.
- Data can be presented at a much higher density. Great for tablets and phones.
- Does not require maintenance of a separate external platform.
- Entirely local to the hub. No need to send events off to a remote system and no delays.

Devices that only present data without a control option are prime candidates. Good examples of this are temperature, motion, pressure, humidity, battery, contacts, presence, leaks etc. The example below is built entirely using **Tile Builder Advanced**. I place all the action items on separate dashboards which are only used when something does not seem right on the main page. The assorted styles show the level of control available in a **Tile Builder** tile.



How Tile Builder Works

There are four components to **Tile Builder.**

1) **Tile Builder Parent App** – The organizing parent app.
2) **Attribute Monitor** (child app) – Generates device tables using a single attribute such as the first battery example.
3) **Activity Monitor** (child app) – Generates device tables using the **Last Activity** attribute and can be used to monitor both active and inactive groups of devices.
4) **Tile Builder Storage Driver** – Device driver used for storing **Tile Builder** data.

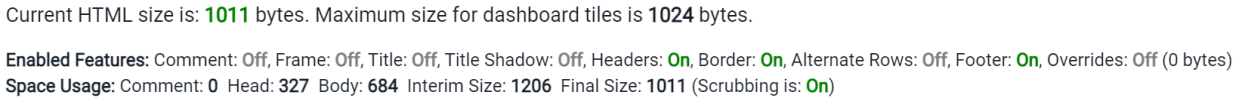
The **Tile Builder** parent app is the primary organizing app under which all others are created.

Tiles are generated by one of the two child apps and organized under the parent app. When tiles are generated, the results are stored in the **Tile Builder Storage Device** in a named tile attribute (tile1 – tile25). This attribute is then placed onto the dashboard as shown previously.

Attribute Monitor uses an event subscription model to monitor a list of devices. If the value of one of the monitored attributes changes the table is automatically regenerated and is updated on the dashboard. This happens within fractions of a second without any perceptible delay.

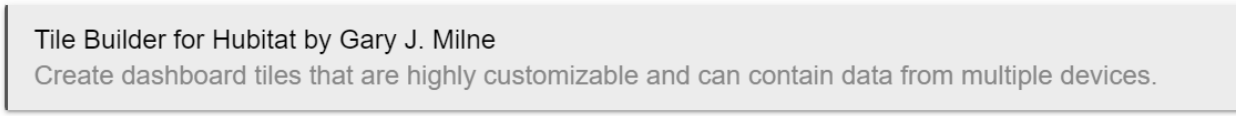
Activity Monitor only relates to the **Last Activity** time on a device and uses a timed model. A query is constructed via the child app such as **Inactive Battery Devices** to list only those battery devices that have had no activity in over 24 hours for example. A refresh interval is assigned from minutes to hours to days. When that interval is reached the table is regenerated, the results are stored in the **Tile Builder Storage Device** and immediately updated on the dashboard.

The Hubitat® dashboard has a limit of 1,024 bytes for any **attributes** that are displayed. Whenever a tile exceeds 1,024 bytes **Tile Builder** publishes an HTML file to the **File Manager** containing the tile contents and then updates the storage device attribute with a link to the new file. **Tile Builder** offers a great deal of optimization and guidance to help you minimize the tile size. Whenever you build a tile, the size of the tile and which components are active is displayed along the bottom.

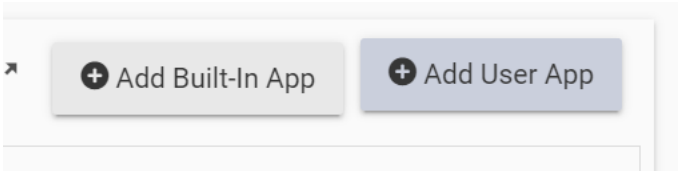


Tile Builder Installation

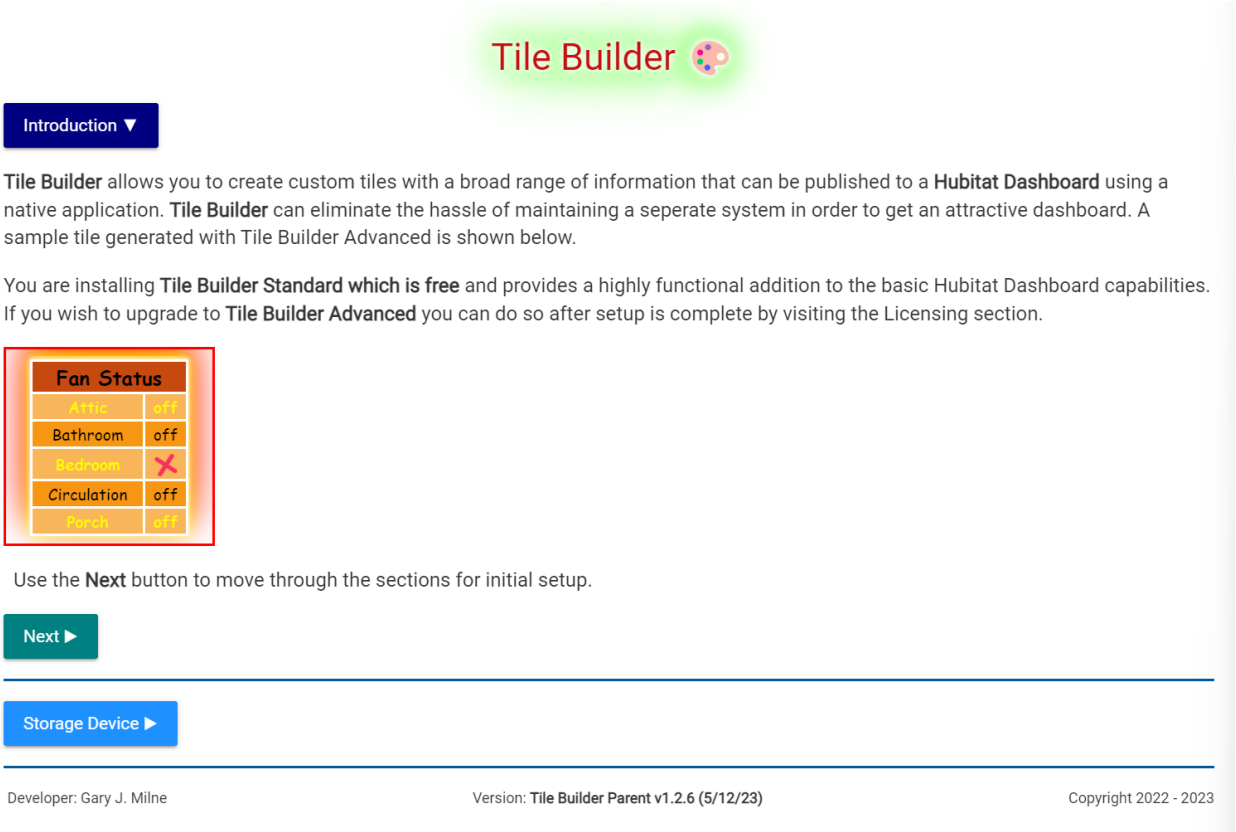
Tile Builder is listed in Hubitat® Package manager. Choose to install by tags and select the **Dashboards** tag. Select **Tile Builder for Hubitat®** and complete the installation process. This will place the code on your hub and then there are a few steps to complete the installation.



1. Go to the Apps tab and click on **Add User App**

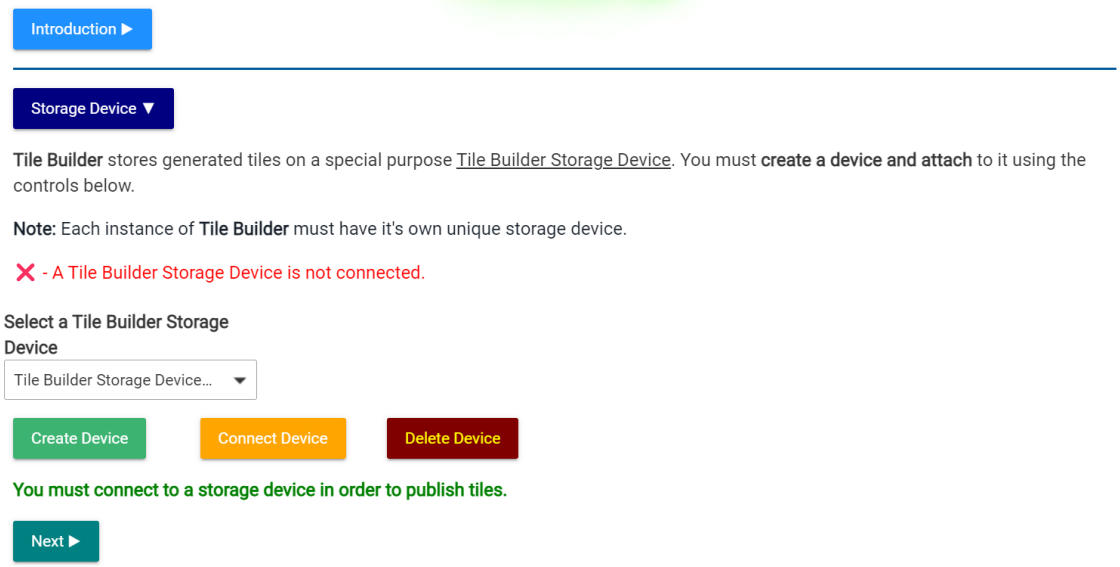


2. Select **Tile Builder** from the list of available apps.
3. **Tile Builder** will install and bring you to the parent screen.

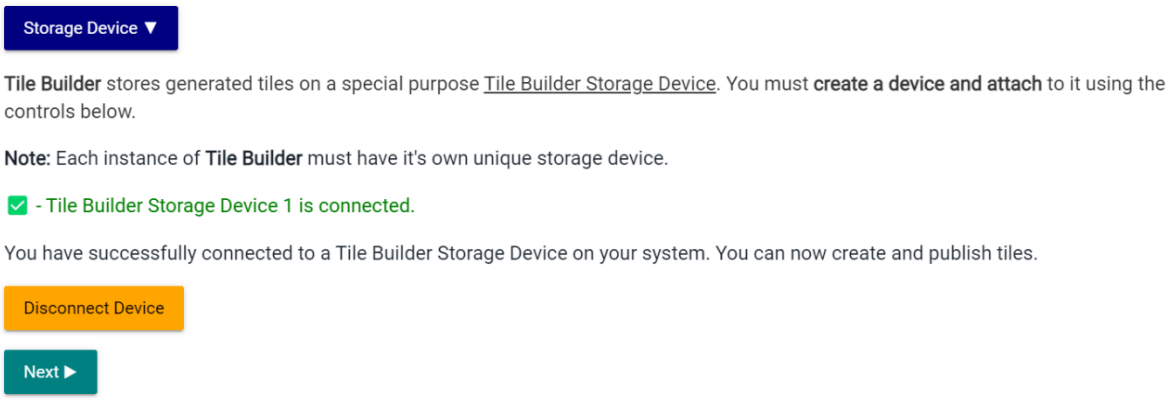


Click Next.

4. Create and Connect a Storage Device

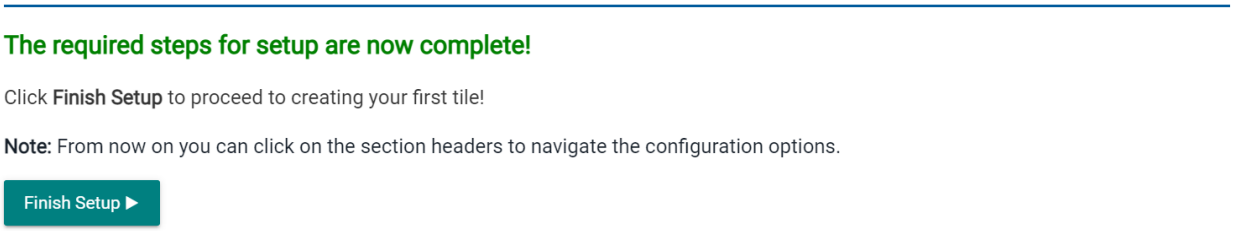


Once the device is created and connected it will look like this.



Click Next.

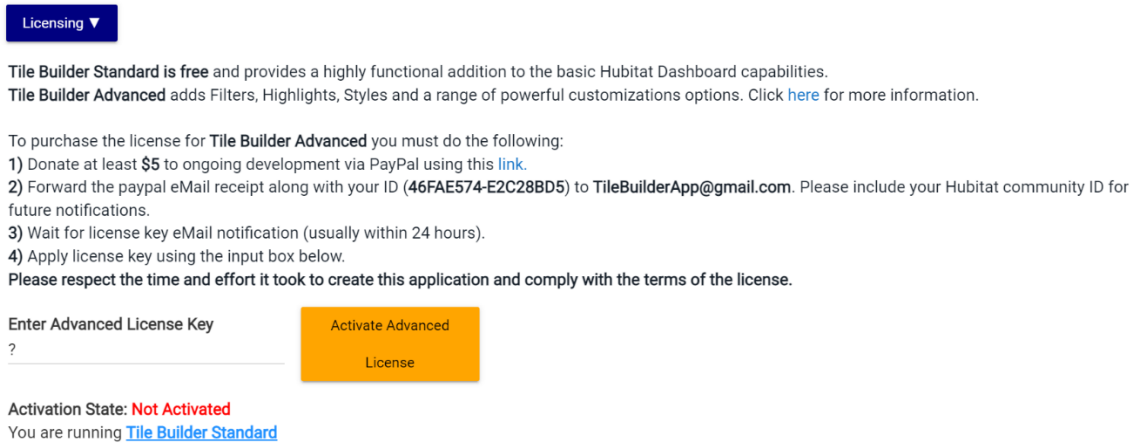
5. Finish Setup



We can now create our first tile.

Licensing

Tile Builder has **Standard** and **Advanced** versions. The **Standard** version has a great degree of functionality and is entirely free to use as much as you wish. With **Tile Builder Standard** you can create tables and fully customize those elements available within the parameters of the UI including Title, Headers, Borders, Rows and Footer. **Tile Builder Advanced** adds powerful features such as filtering, highlighting, thresholds, styles and overrides. The latter allows the customization of elements with values not configurable via the UI.



A lifetime hub license for **Tile Builder Advanced** is currently $5. Just follow the on-screen instructions.

The top 5 reasons to be a registered user are as follows:

1) The extra money means I can justify the time to develop new Tile Builder modules.
2) The next module will be a multi-device X multi-attribute table which should be especially useful for creating a combined view of key elements.
3) New modules will only be available to registered users.
4) Registered users will get priority for any questions posted on the forum or via DM.
5) Help establish a viable market in the Hubitat® community for high quality software solutions.

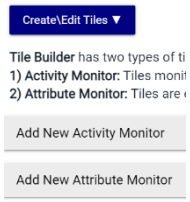
If you don't want to pay $5 you are still free to use **Tile Builder Standard** as much as you wish.

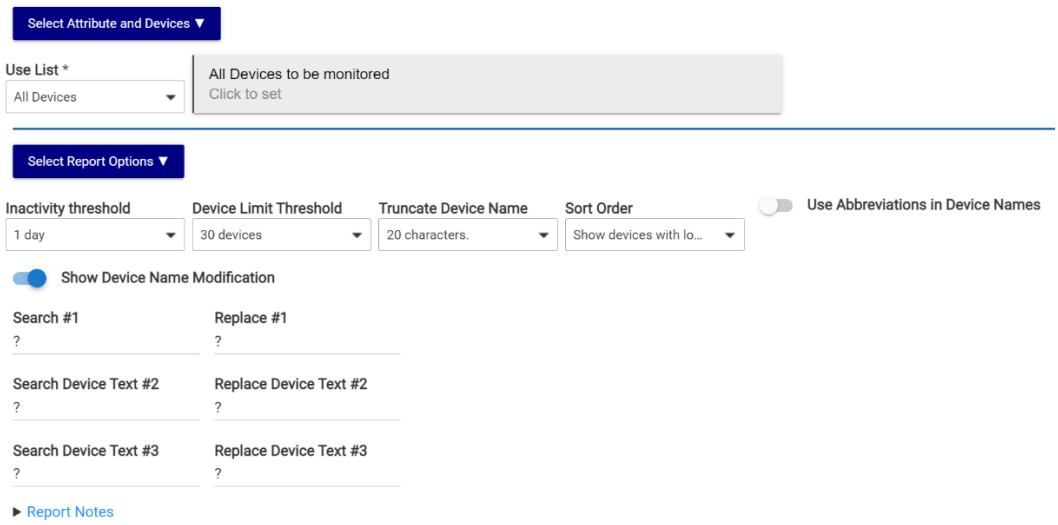
Section 2

Tile Builder Standard

Creating a Tile Using Attribute Monitor

Within the **Tile Builder** parent app go to the section called **Create Tile** and select **Add New Attribute Monitor**. The **Attribute Monitor** main screen will be displayed. The top part of the display controls which devices will be monitored and how that information will be displayed.

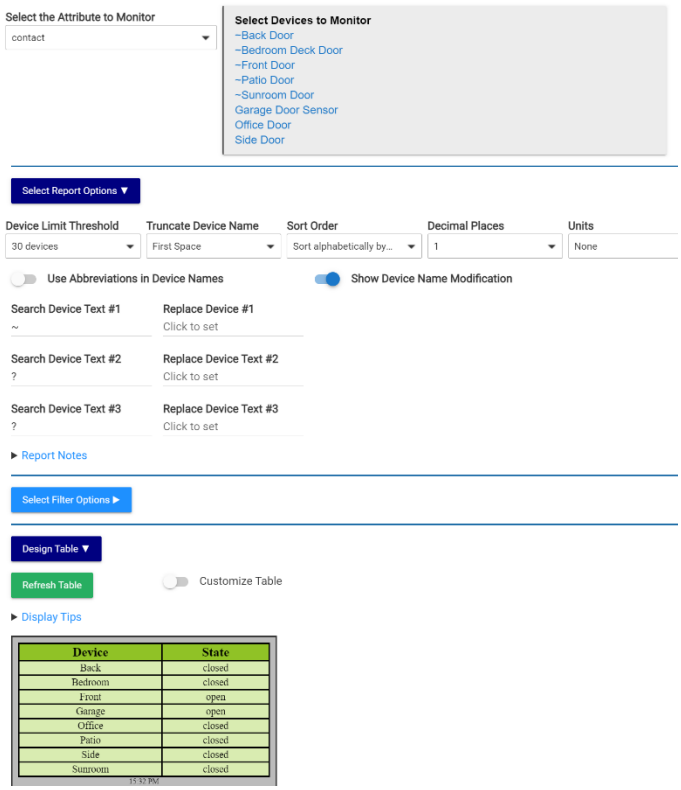




We will create a sample table of **doors** using the **contact sensor** attribute.

1. Using **Select the Attribute to Monitor** list dropdown select '**contact'**.
2. Click on '**Select Devices to Monitor**'. This is a filtered list of all your contact devices. Select all the doors that you have.
3. The **Device Limit Threshold** limits the display to no more than this number, but it may be less. I will change this number to 7 to match the number of doors I have. An example of where this is useful is reporting on the battery state. You could sort the list by lowest battery % first and limit the list to 5 devices. Only those 5 lowest devices would ever be on the list.
4. The **Truncate Device Name** allows you to shorten the device name and is useful in shrinking the amount of data space required. It makes devices names more consistent in length and can reduce text wrapping. In this example I chose to truncate at the first space because they were all named something like 'Front Door', 'Back Door' etc. so the word door was redundant.
5. The **Sort Order** allows you to present the results in a different order. I'm using the default alphabetical sort because it's a relatively short list.
6. **Show Device Name Modification** reveals three search and replace options that you can use to modify device names. I sometimes use ASCII characters such as ! or ~ to group like devices together in the device table and device picker. I can strip those characters from the final display name.
7. The **Use Abbreviations in Device Names** is an option for reducing the size of the data in the result set. With this enabled the word " Room" becomes " Rm", " Sensor" becomes "Sns" etc. When trying to cram a lot of data into a table < 1,024 this can be a useful option. We will leave this turned off here as we have ample free space.

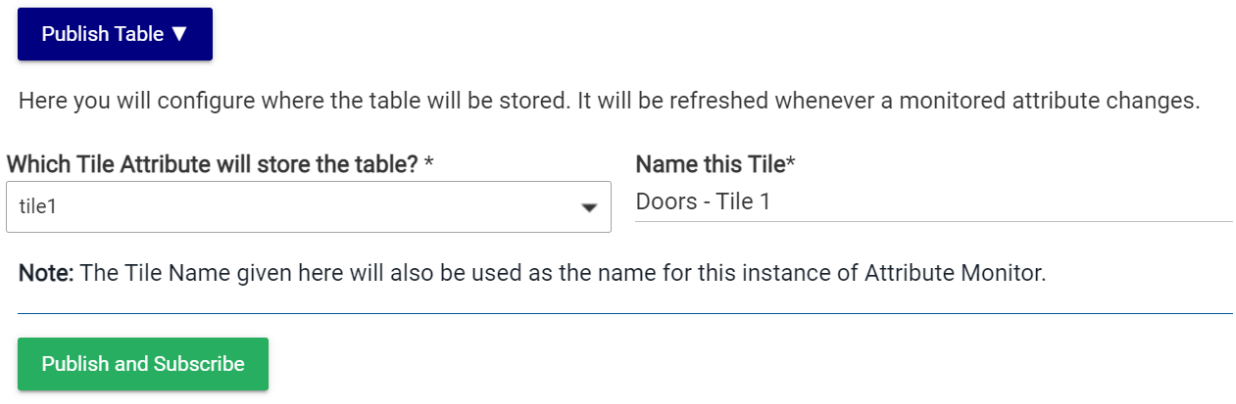
Based on these parameters my table looks like this. A total of 7 doors, one of which is open.



All the table customizations are hidden unless the **Customize Table** option is checked. We will come back to that, but at this point there is nothing else that is absolutely required, and we can publish the table as it currently is.

Publishing a Tile

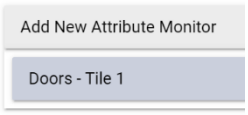
The publishing options are shown below.



To Publish a tile, follow these steps:

1. Select the **Tile Attribute** to store the table in. Tile attributes are tile1 – tile 25. We will use tile1 in this case.

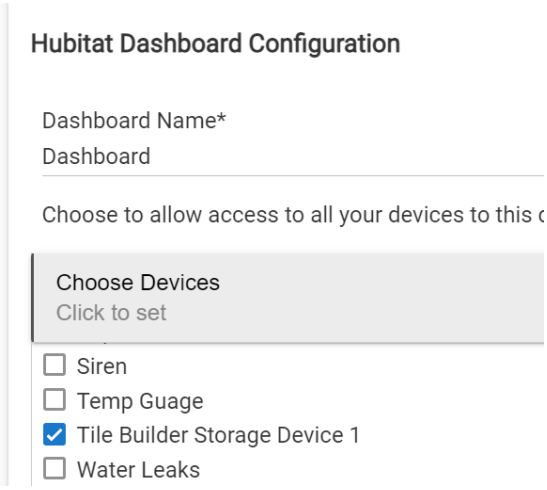
2. **Name the Tile**. I'm going to call it **Doors – Tile 1**. This is also the name that will be visible when looking at the **Tile Builder** parent app. I recommend you append the tile name with the tile number so you can see it on the parent screen.
3. With those values set, click on **Publish and Subscribe**.
4. Click on **Done** to close the **Attribute Monitor** app.
5. You should see your new tile listed under the **Activity Monitor** child app like this.



You can go back and edit this tile any time by clicking on this button.

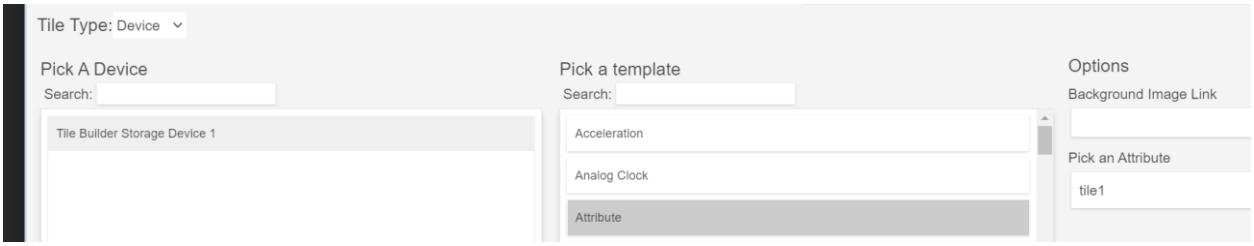
Dashboard Integration

We are done with **Tile Builder** for the moment. Now we can add the newly generated tile to the dashboard of our choice. Before we can do that, we must first grant the dashboard access to the **Tile Builder Storage Device** as shown below.

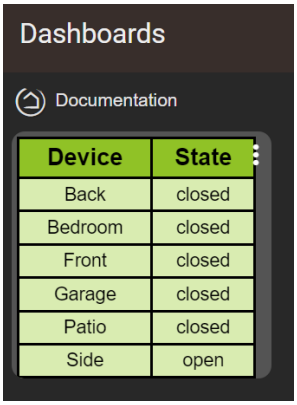


Note: Dashboard does not need access to the underlying devices that provide data for the table.

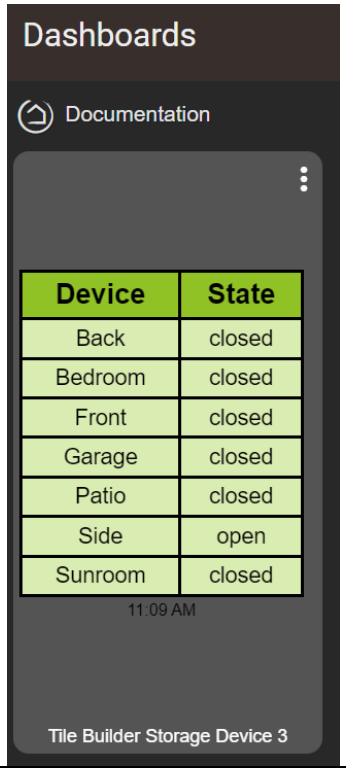
Next, we must add the newly generated tile to the Dashboard in the same way we would add any other device as shown below.



The tile will initially look like this. Notice it overflows the space of the dashboard grid so that only 6 of the doors are visible.



By changing the dimensions of this tile to 1 wide x 2 high it will look like this.

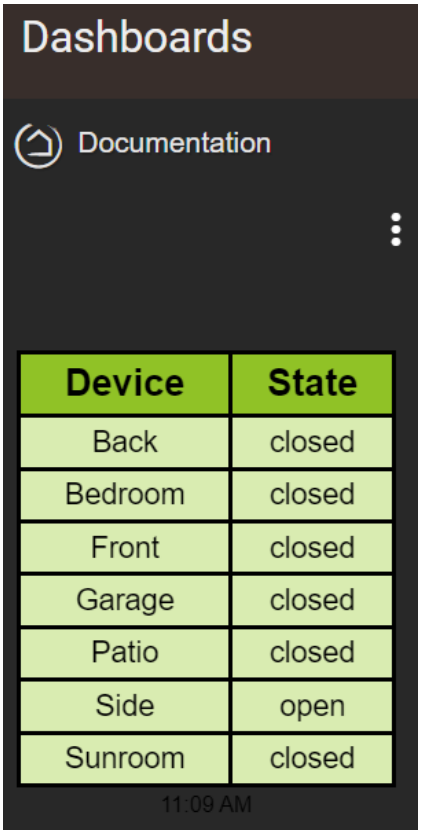


If you wish, you can improve the look of the tile by adding these lines to your dashboard CSS.

- Make the tile background transparent.
  #tile-0 {background-color: rgba(128,128,128,0.0) !important;}
- Hide the name of the device.
  #tile-0 .tile-title {visibility: hidden; display: none;}

Note: The tile number referred to in the above CSS code has no bearing on the tile numbers assigned in the **Tile Builder** app.

With the CSS code applied it is cleaned up a little bit.



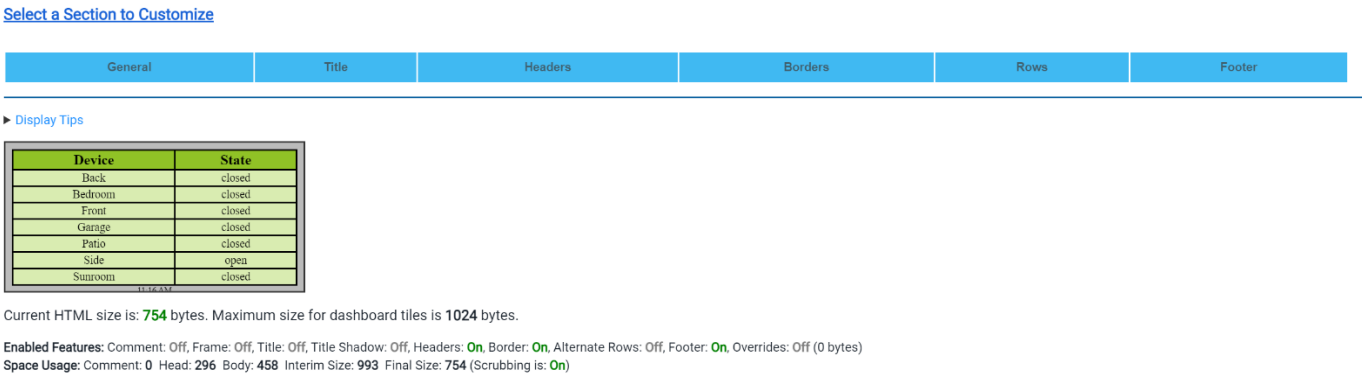
Not bad, it has the basic information but it's not going to win any design awards. Here are a few things I don't like about it so far.

1. The font is rather plain.
2. The purpose of the table could be clearer.
3. It is hard to the read timestamp at the bottom of the time. (Last time it was generated.)
4. The table border blends into the background making it almost invisible.
5. Green is not my favorite color.

Let us go back and fix these and make it a little more pleasing.

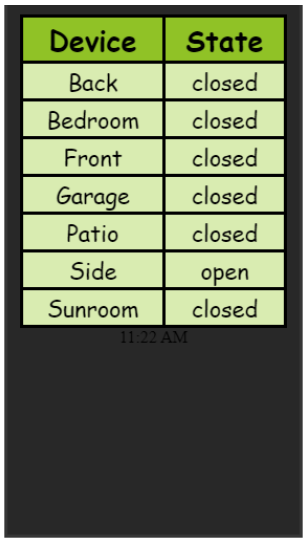
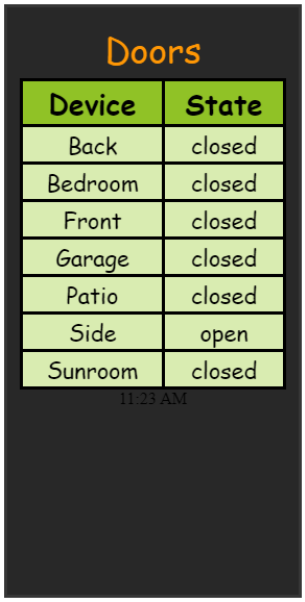
Editing Attribute Monitor Tiles

Open the recently created '**Doors**' tile. This time we will click on **Customize Table** and the UI controls will appear. In the standard version it looks like this:



Before we start, let's go to the **General** tab and change the **Select Tile Preview Size** to match our desired size. In this case **1 x 2**. Then click on **Dashboard Color** and use the eye dropper tool to grab the color from **your** dashboard. This makes the preview more accurate, especially when selecting colors.

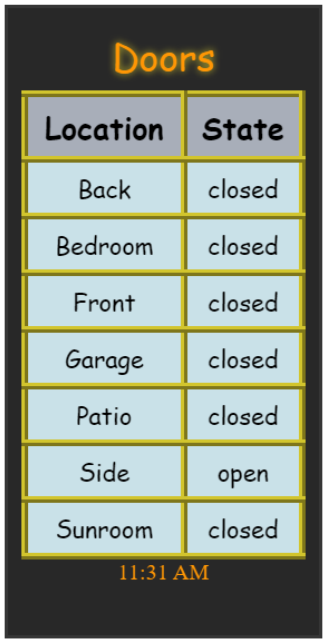
The following table shows the progression of the table as the different elements are changed.

General Tab Select font Comic Sans.	**Title Tab - Add Title** Size 110%, new color.	**Footer Tab** Size 75%, new color.
		

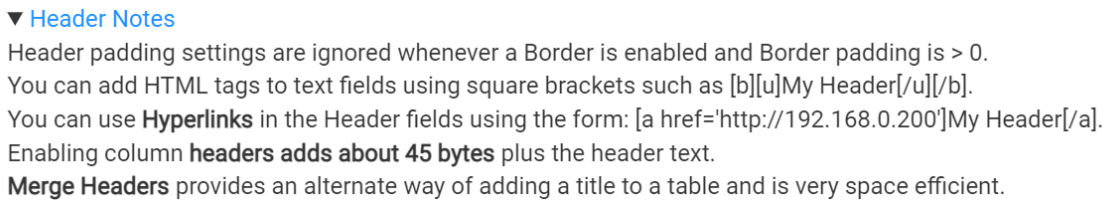
Borders Tab – Change Border
Type: Ridge, Width: 4, new color.

Header Tab & Row Tab
Change background colors and opacity.

Title Tab
Change border and title padding. Add title shadow.



As you can see it is easy to change the table and the number of variations is limitless. The purpose of most controls is obvious from the name. Where explanation is needed it is included under a notes section that exists under each tab. The following example is from the **Header** tab.



Tile Size Discussion

Tile are published in 2 different ways.

Direct: When a tile is less than 1,024 bytes all the HTML\CSS is saved to the **Tile Builder Storage Device** *tileX* attribute. The dashboard reads the contents of the attribute directly from the database. The 1,024 byte limit is enforced by the Dashboard.

< 1,024 Advantages:

1. All data is stored within the Tile Builder Storage Device.
2. These tiles will render within the Hubitat App without the need for a VPN.
3. This HTML can display directly and offers the best and cleanest rendition of the layout. When an update comes through to a tile the only indication that the tile refreshed is the change in state of the device. It's a very smooth visual experience.

< 1,024 Disadvantages:

1. This restriction makes large tables impossible. Realistically you are limited to 10-15 rows with minimal styling.
2. This restriction also reduces the potential for more sophisticated tiles without employing other space saving techniques.

Indirect: When tile size exceeds 1,024 bytes Tile Builder generates a file with the contents. This file is uploaded to the File Manager in the form **TBSD*x*_Tile_*y*.HTML** where x is the number of the Storage Device and y is the number of the tile. Tile Builder then updates the tile attribute telling it to load the updated filename within an iFrame. This is how it gets around the 1,024 size limit.

>1,024 Advantages:

1. The number of rows of data can be much larger, easily 50+.
2. The extra size leaves room for all kinds of embellishments such as animations, transformations, backgrounds, classes etc. Tiles can be quite fancy.

>1,024 Disadvantages:

1. These tiles load through an iFrame and the refresh is a noticeable event, although it can be partially masked with a refresh animation in **Tile Builder Advanced**.
2. The dashboard loads these files from an internal address (your Hub IP) and this address is only available when you are at home or have an active VPN connection. This applies also to the Hubitat app.

Size Management Strategies

You will have to find the right mix for your own tastes but here are some ideas.

With **Standard** use limited lists and sort by the interesting event. For example, instead of displaying all windows and doors, limit the list to 5 and sort by the status so open contacts will show first.

With **Advanced** use a filter to only show those devices in the state you are interested in.

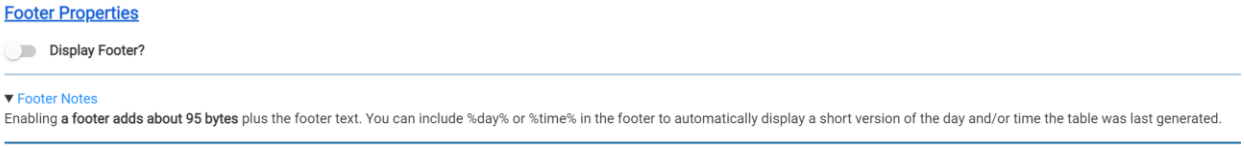
Consider having a limited device list (<1,024) on your main dash such as 5 lowest batteries, with a full device list (>1,024) of all battery levels on a separate dash. This would allow your main dash to work without a LAN connection and still provide very useful status.

Consider breaking tables into smaller groups. Instead of “Lights” you may have “Indoor Lights” and “Outdoor Lights”.

Managing Tile Size

Tile Builder has capabilities built in to help optimize the tables as well as analyze the usage.

1) Under the **Notes** for each section the added payload from enabling a given option is displayed as shown in this example.



2) The area below the table preview shows the current size of the table, the compressed size of the table and which options are currently enabled. For example:

Current HTML size is: **731** bytes. Maximum size for dashboard tiles is **1024** bytes.

Enabled Features: Comment: Off, Frame: On, Title: Off, Title Shadow: Off, Headers: On, Border: On, Alternate Rows: Off, Footer: Off, Overrides: On (176 bytes)
Space Usage: Comment: **0** Head: **493** Body: **238** Interim Size: **920** Final Size: **731** (Scrubbing is: On)

3) **Scrubbing** removes all excess characters from the HTML making it extremely dense and is automatically turned on.
4) When the 1,024 byte limit is exceeded the size will be displayed in red and the **Publish** button will be greyed out (not shown).

Current HTML size is: **1122** bytes. Maximum size for dashboard tiles is **1024** bytes.

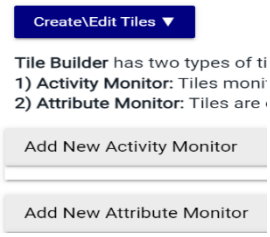
Enabled Features: Comment: Off, Frame: On, Title: Off, Title Shadow: Off, Headers: On, Border: On, Alternate Rows: Off, Footer: Off, Overrides: On (176 bytes)
Space Usage: Comment: **0** Head: **493** Body: **629** Interim Size: **1311** Final Size: **1122** (Scrubbing is: On)

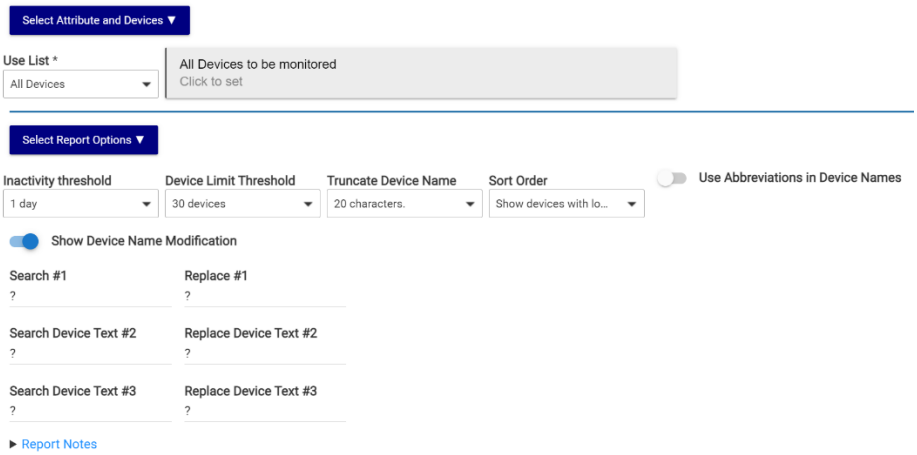
5) If a tile grows to exceed the 1,024-byte limit after initial publication, then a warning message is displayed on the dashboard in place of the tile.

Note: If you have **Tile Builder Advanced** you can load the style called **Everything Off** which uses all the CSS and HTML default values to render the minimal tile size. You can then enable only the components that you deem necessary.

Creating a Tile Using Activity Monitor

Within the **Tile Builder** parent app go to the section called **Create Tile** and select **Add New Activity Monitor**. The **Activity Monitor** main screen will be displayed. The top part of the display controls which devices will be monitored and how that information will be displayed.

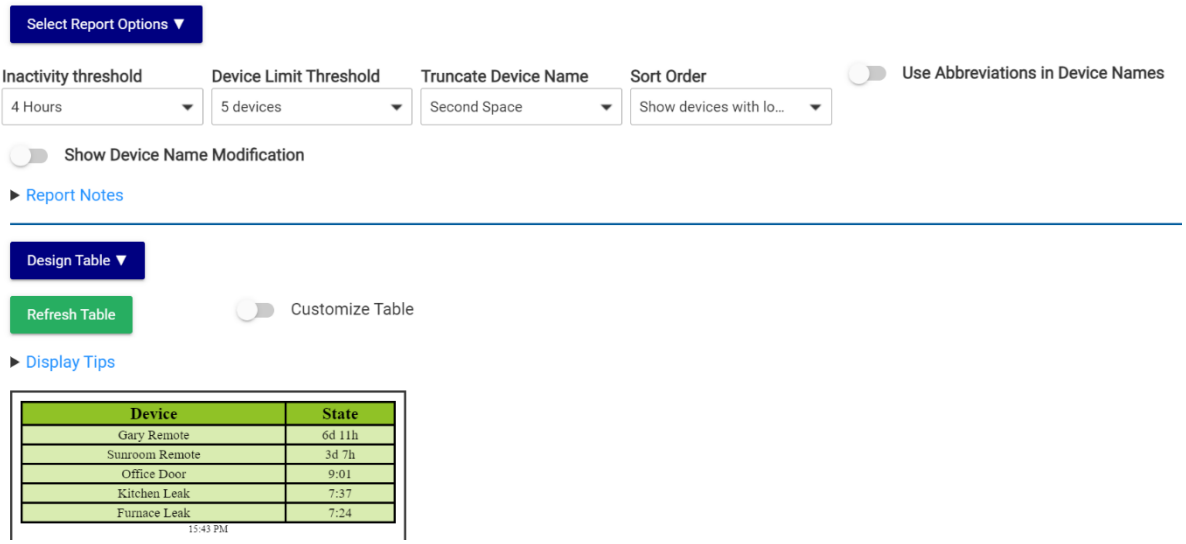




We will create a sample looking for inactive battery devices. Note: I have contact sensor devices that fail while the battery still claims to be 100%, so using the percentage alone is not a guarantee of catching a failed battery.

1. On the **Use List** dropdown select **Battery Devices**.
2. Click on the device selector. This is a filtered list of all your devices that have a battery capability. Select all the devices in the list.
3. The **Inactivity Threshold** will limit the results to only those devices whose inactivity is greater than the threshold value. The default is 1 day, let's change that to 4 hours.
4. The **Device Limit Threshold** limits the display to no more than this number, but it may be less. We will leave that at the default of 5. The 5 displayed devices is determined by the sort order.
5. The **Truncate Device Name** allows you to shorten the device name and is useful in shrinking the amount of data space required, making device names more consistent in length and to reduce text wrapping. In this example I chose to truncate at the second space.
6. The **Sort Order** allows you to sort the table by oldest or youngest. Change the sort order to **Show devices with longest inactivity period** first.
7. **Show Device Name Modification** reveals three search and replace options that you can use to modify device names. I sometimes use ASCII characters such as ! or ~ to group like devices together in the device table and device picker. I can strip those characters from the final display name.
8. The **Use Abbreviations in Device Names** is an option for reducing the size of the data in the result set. With this enabled the word " Room" becomes " Rm", " Sensor" becomes "Sns" etc. When trying to cram a lot of data into a table this can be a very useful option but is mostly of use in **Attribute Monitor** where the result sets are longer. We will leave this turned off here.

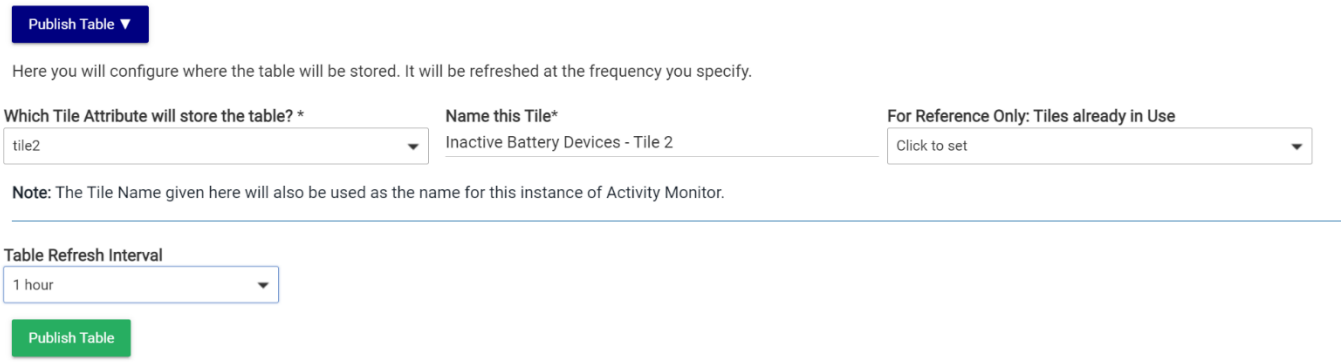
Based on these parameters my report looks like this. Only 5 battery devices have not been active in the last 4 hours as shown below.



That is all we are required to do, now we can go ahead and publish the table.

Publishing a Tile

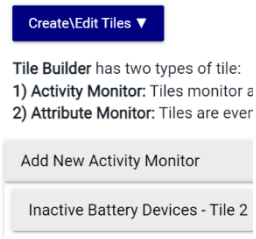
The publishing options are shown below.



To Publish a tile, follow these steps:

1. Select the **Tile Attribute** to store the table in. Tile attributes are tile1 – tile 25. We will use tile2 in this case.
2. **Name the Tile**. I'm going to call it **Inactive Battery Devices – Tile 2**. This is also the name that will be used when looking at the **Tile Builder** parent app. I recommend you append the name with the tile number so you can see it on the parent screen.
3. Select a **Table refresh interval**. No need to make the hub work too hard. In this case I think hourly would be reasonable.
4. With those values set click on **Publish Table**.
5. Click on **Done** to close the **Attribute Monitor** app.

6. You should see your new tile listed under the **Attribute Monitor** child app like this.



You can go back and edit this tile any time by clicking on this button.

Tile Size Revisited

In the first example the final tile size was 1,008 bytes after scrubbing and close to the upper limit of 1,024 bytes. As a **closed** door uses 2 bytes more than an **open** door, we need to leave some buffer space to handle the resulting variations in tile size. In this case the less frequent state (open) is shorter than the more frequent state (closed) so it won't be an issue in this case.

Current HTML size is: **1001** bytes. Maximum size for dashboard tiles is **1024** bytes.

Enabled Features: Comment: Off, Frame: Off, Title: On, Title Shadow: On, Headers: On, Border: On, Alternate Rows: Off, Footer: On, Overrides: Off (0 bytes)
Space Usage: Comment: **0** Head: **541** Body: **460** Interim Size: **1201** Final Size: **1001** (Scrubbing is: On)

If we wished to add more lines of data, we would have to cut some of the features that were turned on. In this case I would probably turn off the Title, merge the Headers and use that as a title. I'd add some border padding and change the font size a little bit to fill in the extra space. If I do that, the tile now looks like this:

If we look at the HTML info, we can see the tile size has shrunk from 1,001 bytes to 846. That is a substantial difference and would be sufficient to add at least 5 more rows of data if desired.

Current HTML size is: **846** bytes. Maximum size for dashboard tiles is **1024** bytes.

Enabled Features: Comment: Off, Frame: Off, Title: Off, Title Shadow: On, Headers: On, Border: On, Alternate Rows: Off, Footer: On, Overrides: Off (0 bytes)
Space Usage: Comment: **0** Head: **393** Body: **453** Interim Size: **1008** Final Size: **846** (Scrubbing is: On)

Embedded HTML Tags

Anywhere you can enter text you can wrap it inside HTML tags. But rather than using **<>** you must use **[]** as normal HTML tags are rejected by the Hubitat® interface. For example, you could enter **[u]Door Status[/u]** in the title field and the title would be displayed in underline, **[b]Door Status[/b]** and it would display in bold. Multiple HTML tags can be used.

Note: I have observed that it is not necessary to close HTML tags such as [/b] or [/u] located in text fields such as Title\Header\Keyword substitution. This can save a little space, especially in Keyword substitution where the values are repeated on each row.

Macros

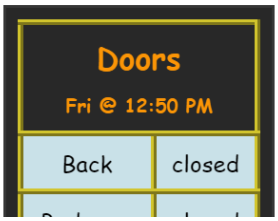
The following values are macros that will be expanded in the final HTML.

- **%day%** will be replaced by a short version of the day name.
- **%time%** will be replaced by a 24-hr. time including AM\PM.
- **%units%** will be expanded into the Units chosen, if any.
- **%count%** will be expanded into the number of data rows in the table.

In the above example the **Doors** header text was replaced with the text:

Doors [br][font size=2]%day% @ %time%[/font]

With the result looking like this.



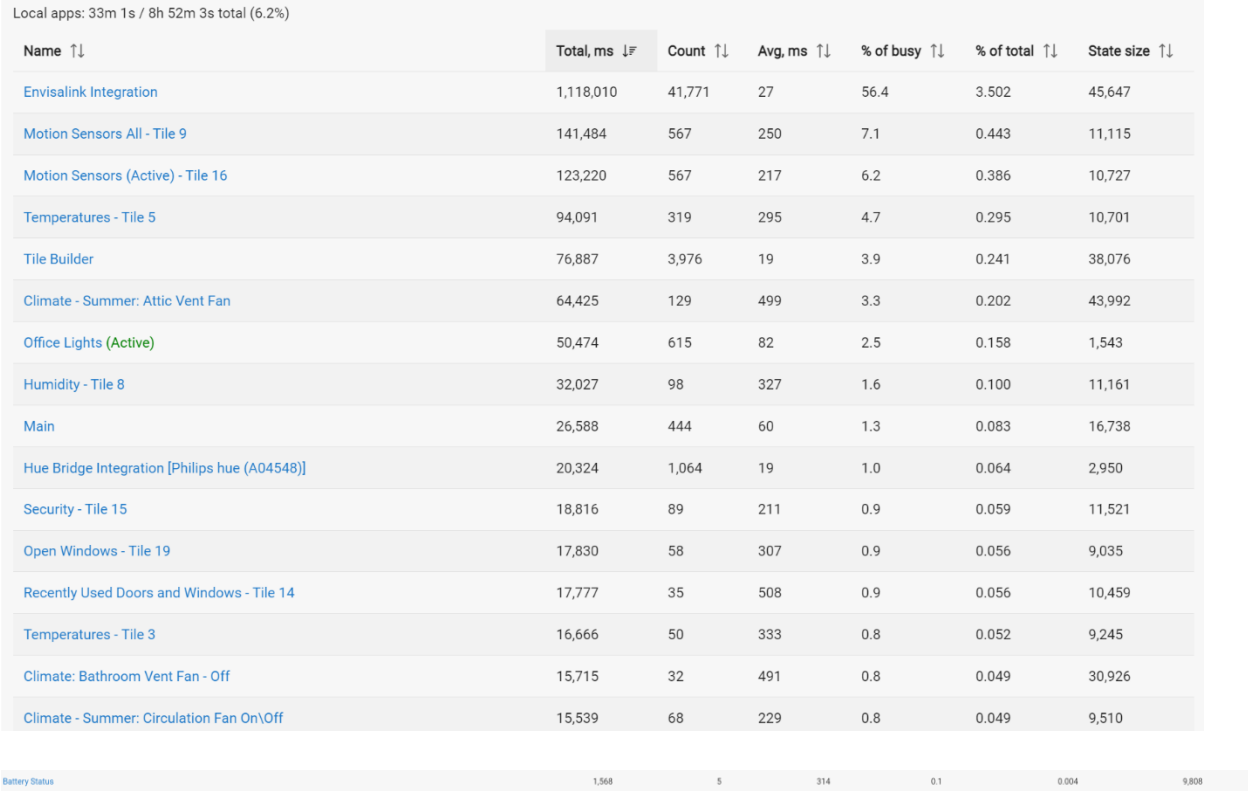
You can use these macros in any text field combined with HTML tags.

Publish and Subscribe Model

When you click on the **Publish and Subscribe** button, **Tile Builder** creates an event subscription to each of the selected devices and chosen attribute. The **Attribute Monitor** child app then remains dormant until such time as one of the monitored attributes changes, in the above case if a door opens or closes, at which point the table is immediately regenerated and published. This is a highly efficient model, and **Tile Builder** tiles will only regenerate when the underlying data has changed. Tile updates will appear on the dashboard at the same speed as a typical device tile.

Hubitat® CPU Utilization

You can view the performance of **Tile Builder** apps under **Logs\App Stats**. The screenshot below is from my production system where I have about 25 active tiles. In the below example you can see that the busiest tiles are my two motion sensor tiles because I have a motion sensor right next to me in the office where I work and it's firing constantly most days. Even under those extreme conditions those two tiles combined account for less than 1% of the total time. My Temperatures tile which updates any time 1 of 12 temperatures change, only uses 0.052% of the available CPU time.



Local apps: 33m 1s / 8h 52m 3s total (6.2%)

Name	Total, ms	Count	Avg, ms	% of busy	% of total	State size
Envisalink Integration	1,118,010	41,771	27	56.4	3.502	45,647
Motion Sensors All - Tile 9	141,484	567	250	7.1	0.443	11,115
Motion Sensors (Active) - Tile 16	123,220	567	217	6.2	0.386	10,727
Temperatures - Tile 5	94,091	319	295	4.7	0.295	10,701
Tile Builder	76,887	3,976	19	3.9	0.241	38,076
Climate - Summer: Attic Vent Fan	64,425	129	499	3.3	0.202	43,992
Office Lights (Active)	50,474	615	82	2.5	0.158	1,543
Humidity - Tile 8	32,027	98	327	1.6	0.100	11,161
Main	26,588	444	60	1.3	0.083	16,738
Hue Bridge Integration [Philips hue (A04548)]	20,324	1,064	19	1.0	0.064	2,950
Security - Tile 15	18,816	89	211	0.9	0.059	11,521
Open Windows - Tile 19	17,830	58	307	0.9	0.056	9,035
Recently Used Doors and Windows - Tile 14	17,777	35	508	0.9	0.056	10,459
Temperatures - Tile 3	16,666	50	333	0.8	0.052	9,245
Climate: Bathroom Vent Fan - Off	15,715	32	491	0.8	0.049	30,926
Climate - Summer: Circulation Fan On\Off	15,539	68	229	0.8	0.049	9,510
Battery Status	1,568	5	314	0.1	0.004	9,808

In addition to the **Tile Builder** apps utilizing the CPU the **Tile Builder Storage Device** is also a consumer. You can track this in Logs\Device Stats as shown below. For the instance shown below there are 20+ published tiles, and the **Tile Builder Storage Device** is using about 0.2% of the total.

Devices: 23m 14s

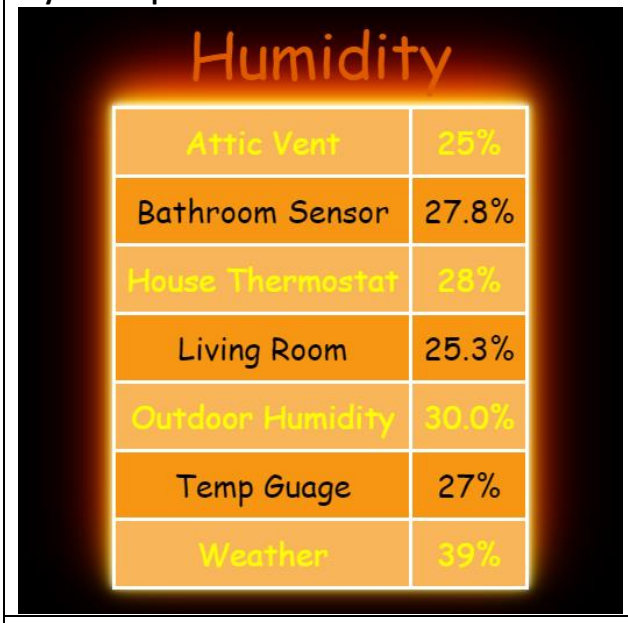
Name	Total, ms	Avg, ms	% of busy	% of total
Envisalink	1,249,632	252	89.6	3.911
Tile Builder Storage Device 1	56,345	30	4.0	0.176
Temp Guage	14,267	11	1.0	0.045
Dining Room Thermostat	13,645	126	1.0	0.043

Final CPU utilization in any given case will be a function of the number of tiles published and especially how frequently those tiles are updated. For this reason, publishing rapidly changing data such as power consumption needs to be done thoughtfully.

Summary

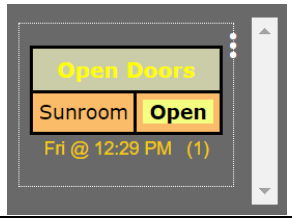
As you can see, **Tile Builder** offers a great deal of flexibility in how to display data onto a Hubitat® dashboard. What you have learned so far should keep you busy for a while. While **Tile Builder** has a lot to offer, **Tile Builder Advanced** is a big step up in terms of flexibility and control. The tiles in the table shown below were created with **Tile Builder Advanced**.

Style Example - Halloween

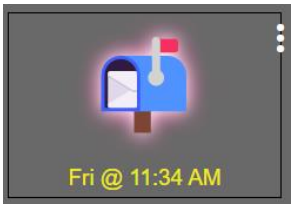


Highlighting Example – Keyword substitution

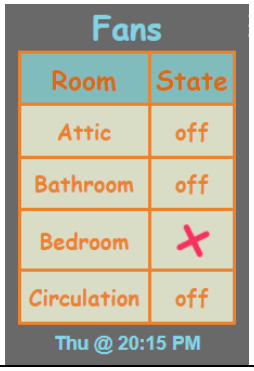
Filtered Lists



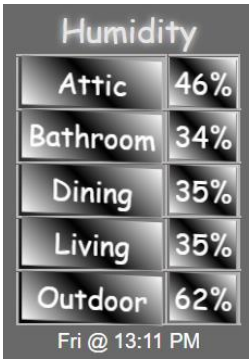
Icons



Animation



Effects



In the next section we will explore all the capabilities of the **Tile Builder Advanced** version.

Section 3

Tile Builder Advanced

Overview of Advanced Features

Tile Builder Advanced adds significant capabilities to the base version. In summary these are:

Filtering: Filtering can be used to publish a subset of results that meets certain criteria. For example, doors that are open, temperatures => 80 for example.

Highlighting: Highlighting is a way to draw attention to specific values that are of greater importance by changing how those values look and making them more prominent.

> **Keywords:** These are used to match a string value and enhance those with color, size or completely replace the value. For example, rather than display the word **closed** for a contact sensor you could specify ✔ or O.K. be used instead.
>
> **Thresholds:** These allow numeric values that meet >=, ==, or <= conditions to be highlighted or replaced with a text value. Batteries <= 50% could all display as Low instead.

Styles: Styles are a collection of settings for quickly and consistently modifying how data is displayed.

> **Built-In Styles:** Tile Builder has over a dozen Styles built into the app. These are there primarily to spark your imagination and demonstrate the power of Styles. These are not polished color guides, but I expect to add more of those later with help from the community.
>
> **Apply Styles:** Apply a built-in style or your own style to a table. This only affects those properties associated with how the table looks and feels. It does not affect the contents of fields such as Title, Headers, or Footer.
>
> **Save Styles:** Create your own styles and save them for future use.
>
> **Delete Style:** Remove a style you no longer wish to keep.
>
> **Import\Export:** Share style strings with others via Hubitat® Community forums or save an imported style string as a new Style.

Advanced\Overrides: Overrides are a powerful tool akin to a programming model. With overrides you can achieve all kinds of visual effects on a table by overriding or adding to the existing contents of a field. This allows for the creation of new classes and have those classes act on various parts of the table such as the header or row data only.

> **Show Effective Settings:** A diagnostic tool for showing the combined result of normal and override settings.
>
> **Show Pseudo HTML:** A diagnostic tool for showing the final HTML but using [] instead of <>.
>
> **Overrides Helper:** A collection of 40+ examples of overrides used to achieve all kinds of style effects not possible through the interface.

We will go through each of these sections in more detail in the remainder of this section.

Filtering Results

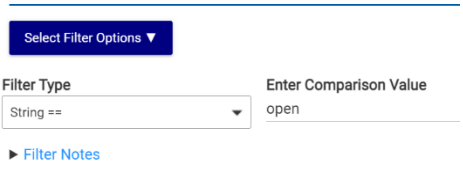
Filtering allows the table to contain a subset of the selected devices based upon specific criteria. There are multiple advantages to filtering.

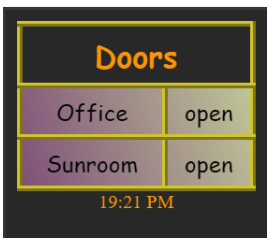
1. In many cases we only care about exceptions, such as open windows or doors, active motion sensors, lights or switches that are on etc.
2. A filtered result set is naturally smaller and takes up less space on the dashboard. Great for the master dashboard where space can be tight.
3. A filtered result set generates a smaller file size and therefore leaves more room for enhancements like titles\footers\animations etc.
4. A filtered list focusses attention on those items that are important, not the mundane.

An example: In my house I have 18 contact sensors covering ground floor doors and windows. In the traditional model that would be 18 1x1 tiles on a dashboard. With an unfiltered list I can do the same thing with a 1x3 tile for windows and a 1x2 tile for doors. That is a 72% space reduction.

With a filtered list I could use a single 1x2 tile that could display 8-10 open contacts, more than would ever be open at one time in my house. That is an 88% reduction in space over the traditional 1 tile per device model. I also have full control over the look of the tile and what is presented in lieu of the standard “open” value.

To filter a result set, select a **Filter Type** and enter a comparison value. In the example below we are using a string comparison for the word ‘**open**’. Filters can be text or numeric comparisons.

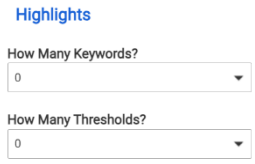




Only results that match this string will now qualify to be added to the table. In the above example we are using **Highlighting** to substitute “Open” for “open” which is a little more visually appealing.

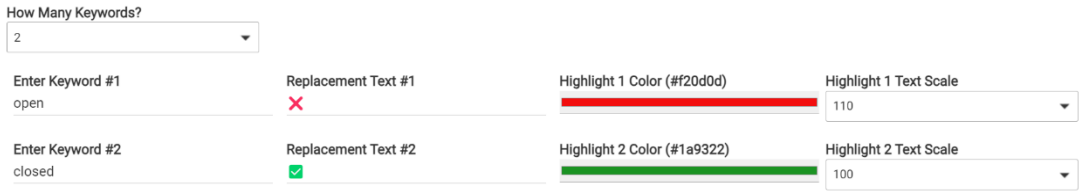
Highlighting

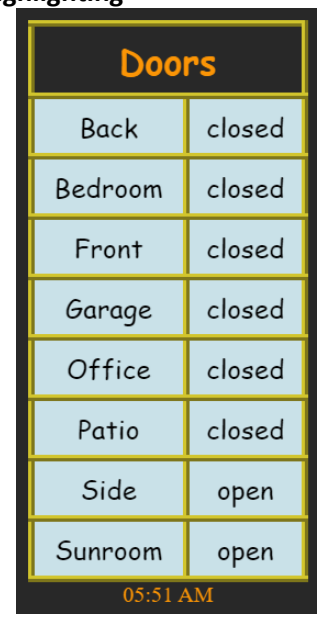
The Highlights interface looks like this and is only available in **Attribute Monitor**.



Keywords

This is a remarkably simple concept, simply stated you can replace one string with another to produce a more attractive result. The replacement string\character can be animated, see the **Classes** section later for details on how to do this. **Note:** When increasing a highlight scale, choosing a scale greater than 100% will cause the height of the row to change slightly to accommodate it.



Without Highlighting	**With Highlighting**
	
Size: 896 bytes	**Size: 1017 bytes**

Thresholds

Thresholds are like keywords but act on numerical data vs. strings. Available comparisons are <=, ==, and >= . You can use up to 5 of these thresholds as shown in the example below.

%value% Macro

You can use the %value% macro anywhere in the **Keywords** or **Highlights** replacement text field and it will be replaced by the actual value, regardless of it's data type.

Battery Example:

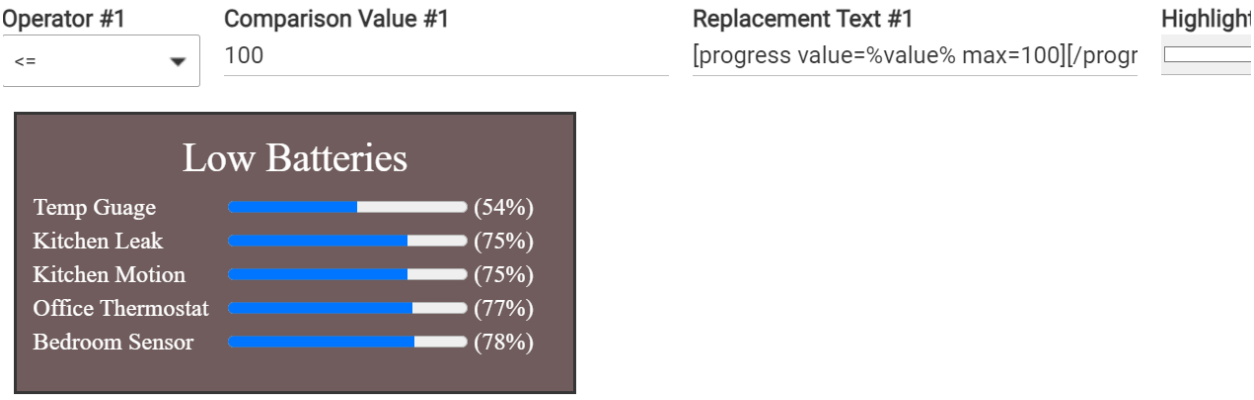
This example uses the HTML "Meter" tag.

The full text of the replacement field is:

[meter low=70 high=80 max=100 optimum=100 value=%value%][/meter] (%value%%)

Progress Example:

This example uses the HTML “Progress” tag.



The full text of the replacement field is:

[progress value=%value% max=100][/progress] (%value%%)

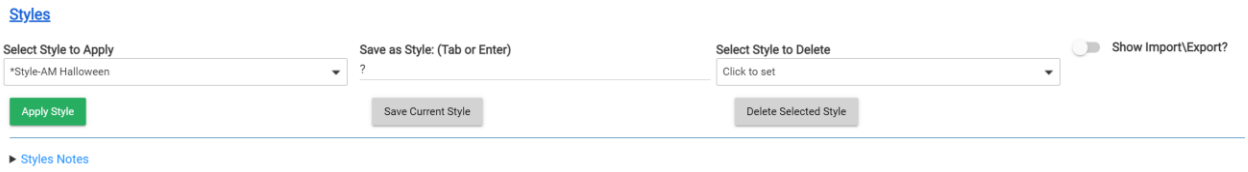
HTML Tags

You can use embedded HTML within the text replacement fields by using **[]** braces. For example, replacing **closed** with ‘**[b]O.K.[/b]**’ would make a closed-door show as **O.K.**, which has a bit more punch.

You can combine HTML tags with the %value% macro. A value of 74 could be formatted with: [b][u]%value% °F[/u][/b] to produce **74 °F**

Styles

The options under the Style tab look like this.



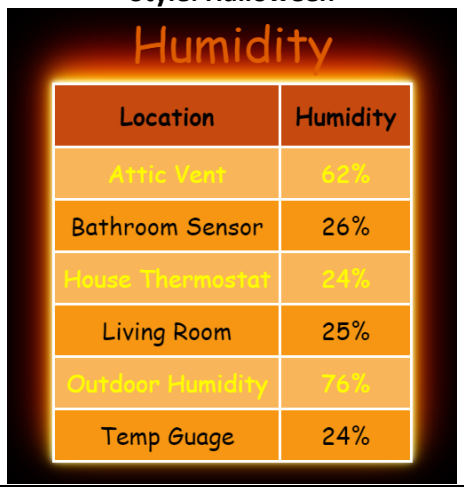
Styles are a quick way of grouping all table settings (not data) together in a convenient package. You may generate an initial table that looks like this.

Location	Humidity
Attic Vent	62%
Bathroom Sensor	26%
House Thermostat	24%
Living Room	25%
Outdoor Humidity	76%
Temp Guage	24%

Apply Style

By applying a style, you can effortlessly make a tile look entirely different.

Style: Halloween



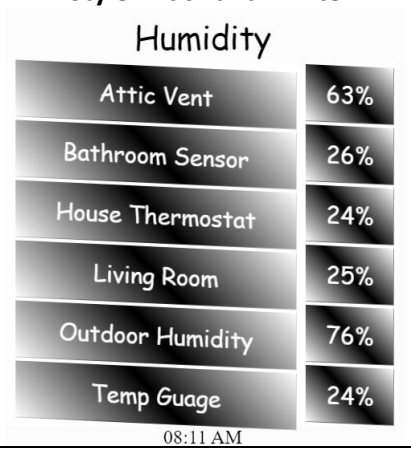
Style: Wood

Style: Pastel Swirl

Location	Humidity
Attic Vent	62%
Bathroom Sensor	26%
House Thermostat	24%
Living Room	25%
Outdoor Humidity	76%
Temp Guage	24%

08:10 AM

Style: Black and White

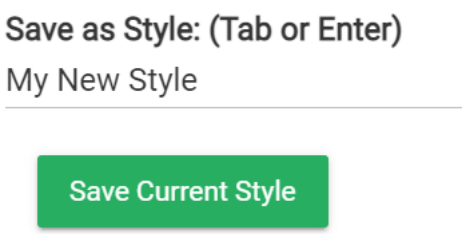


Size is always a consideration given the 1,024-byte limit. The more elements in a style, the larger it will be. All these examples are in the 900-950 byte range including and could easily be published as shown.

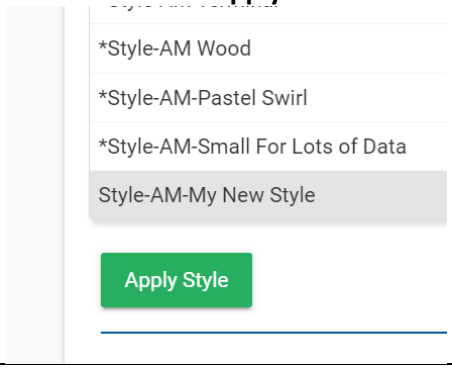
Save Style

You can create your own styles so that you can easily make numerous tables consistent without having to manually apply all the various settings over and over. Once you have the table looking as you wish, simply type the new style name into the text box. Use Tab or Enter to submit that change and the **Save Current Style** button will no longer be greyed out and can be clicked.

Style: Save New Style



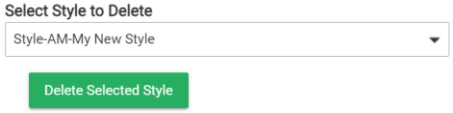
Style: Now available to apply.



Built-In styles are preceded with an * so they sort to the top of the list. User created styles will appear at the bottom of the list in alphabetical order. Styles are stored under the parent app and are shareable between **Activity Monitor** and **Attribute Monitor**. Future modules will share these styles where the color scheme is compatible.

Delete Style

To delete a style simply select it from the list and click **Deleted Selected Style**.

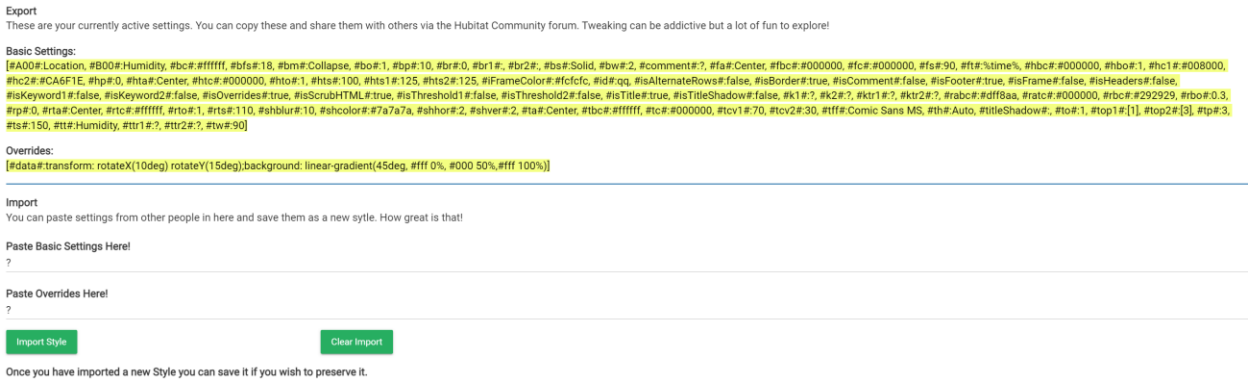


Import\Export

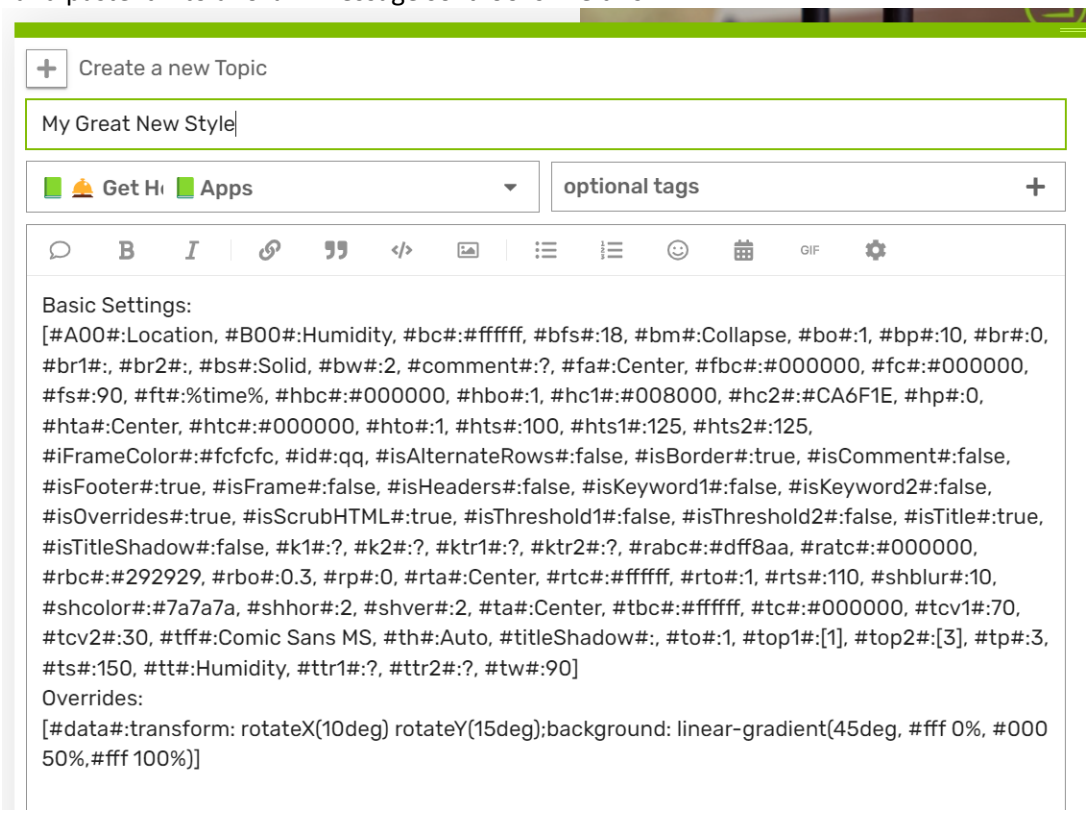
Hubitat® has a vibrant community forum and it is a great asset, so why not use it. When **Tile Builder Advanced** users produce an attractive or innovative style, they can easily share it with others via the forum using the **Import\Export** functions.

Export a Style

The screen below shows the **Export** settings for the currently active configuration split into **Basic Settings** and **Overrides**. Don't worry if these strings do not make any sense yet, they will after you understand **Overrides** which are covered later.



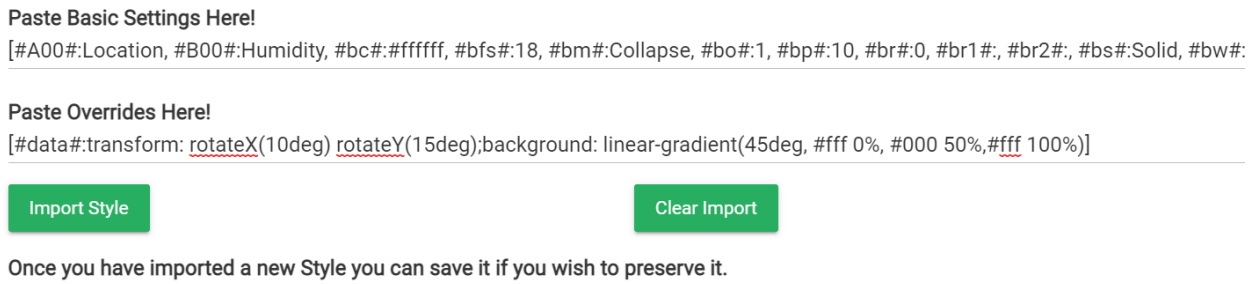
To share a style with others simply copy the text from **Basic Settings: down to the end of the Overrides section** and paste it into a forum message so it looks like this.



That's it, that is all you need to do to share a style with other people.

Import Style

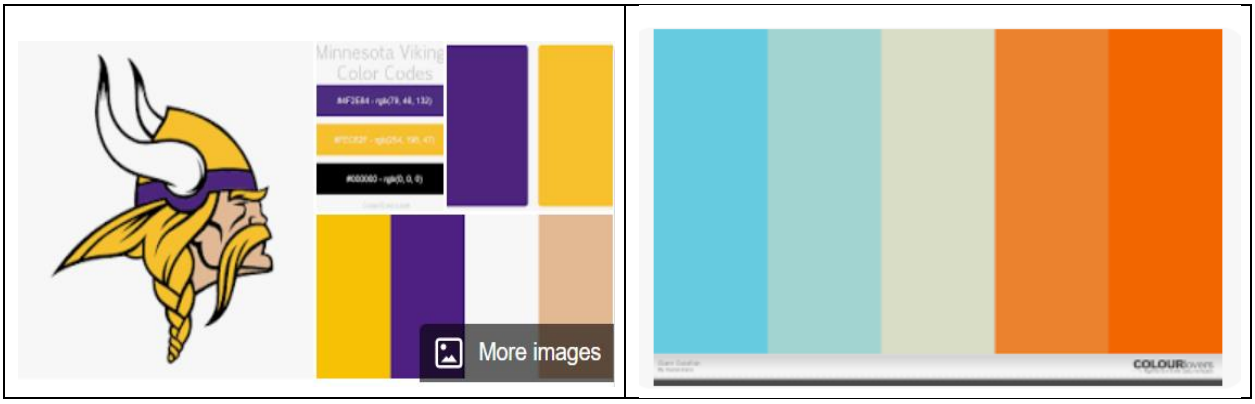
To import a style, you simply reverse the process described above. Copy the string values from a forum into the **Basic Settings** and **Overrides** fields and click **Import Style**.



The new settings will be immediately applied. If you like the new style, you can save it to your style list for future use.

A Note on Styles

Color coordination is not a talent that I possess, and the built-in styles reflect that. I'm hoping that others in the community with a better eye for color and design will share their creations for the benefit of the community and we can incorporate those as built in styles in future releases. What the built-in styles do offer, is an example of what is possible with **Tile Builder Advanced** and hopefully spark your imagination to explore and make your own creations. It's quite a fun way to spend 15 minutes and you can use the eye-dropper tool to select colors from images on the internet such as these to create your personalized color scheme. These two swatches were used as the basis for the **Vikings** and **Palm Springs** styles.



Advanced\Overrides

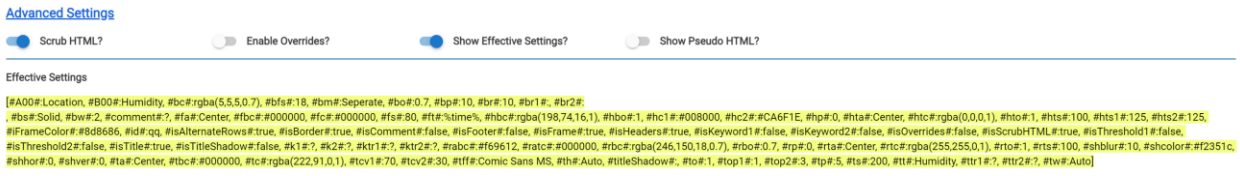
The contents of the Advanced tab look like this.

Scrub HTML

When this is enabled a scrubbing routine is employed to remove all excess characters from the HTML. For example, the default text alignment is left so any references explicitly setting this value can be removed. To see the default values for various HTML elements, load the style **Everything Off**, this will set all values to their default setting for the smallest table size.

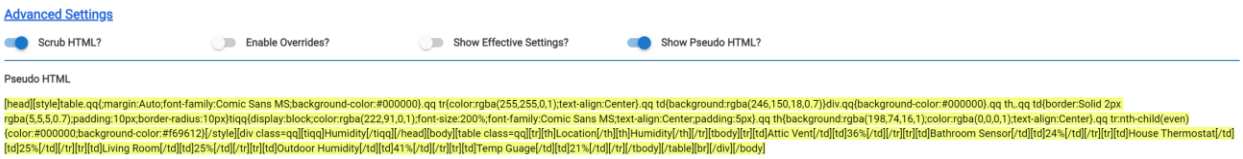
Show Effective Settings

This is a diagnostic tool that provides an easy way to view the effective settings which are a combination of the UI selected settings with the Overrides applied. Normally this can be left off.



Show Pseudo HTML

This is also a diagnostic tool used for troubleshooting. When enabled, the HTML is displayed in text form but using **[]** instead of **<>** so it can be viewed in text. You can copy this to an HTML editor and replace the **[]** with **<>** and it will render. Normally this can be left off.



Overrides

Overrides are a way of replacing content within the HTML with something else. To better understand how overrides work, consider this line of code from the **Tile Builder** application.

String HTMLTITLESTYLE = "<style>ti#id#{display:block;color:#tc#;font-size:#ts#%;font-family:#tff#;text-align:#ta#;#titleShadow#;padding:#tp#px;#title#}</style>"

Each instance you see **#???#** is a field that will get replaced in the final HTML. For example **font-size:#ts#**. If we set the font size to 90% using the UI it will become **font-size:90%**. This occurs many times over for each of the different settings. All these field codes are laid out in **Appendix A**.

Simple Use Case

You won't use **overrides** if the setting that you need is available within the menu system, but the menu system has a finite number of options so there are times when it makes sense.

- You can set row text to 70% or 75%, but what about the values in between? **#rts#=72**
- Use a font family not available in the **Tile Builder** UI. **#tff#=Blackadder ITC** (on Windows)

But these are edge cases and not super important compared to the Class tags described next.

Class Tags

Looking back at that line of code for the **Title block** you will also see an entry **#title#**. This is a placeholder that we can use to extend the properties of the title because it's part of the title class that begins <style>ti#id# and ends </style>. These entries require a property and a value, for example:

#Title#= background-color: #ff00ff;

In this case **#title#** block will be replaced with **background-color: #ff00ff;** and the result looks like this.

Even though the **Title** field does not have a background property defined in the **Tile Builder** UI we can still add one using this method. We are not limited to a single command. In the next example we set a gradient behind the title and add a border radius to make an oval.

#Title#=background-image: repeating-conic-gradient(red 10%, yellow 15%);border-radius: 55%

It produces this ugly result but you get the idea:

Each of the sections of the HTML table has its own class tag as follows. The **#???#=outline: 2px ….** is changed in each example to show which area is affected.

Class Tag Examples

#Table#=outline: 2px solid red;

#Border#=outline: 2px solid red;
Or #Row#=outline: 2px solid red;

#Title#=outline: 2px solid red;

#Header#=outline: 2px solid red;

Note: Header is merged here.

#Data#=outline: 2px solid red;

#Footer#=outline: 2px solid red;

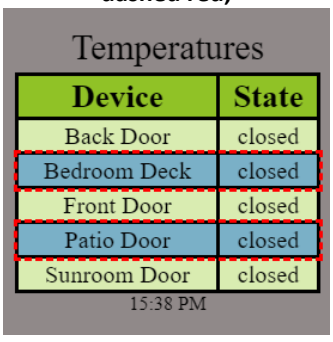
In the above example we have used the outline property, but we could use any valid CSS property to change the text color, bold, italic, size, background color etc. See W3 Schools CSS reference here.

The above examples are obvious except for **#Data#**. This tag simply refers to the table data (<td> for the HTML savvy), excluding any table header present.

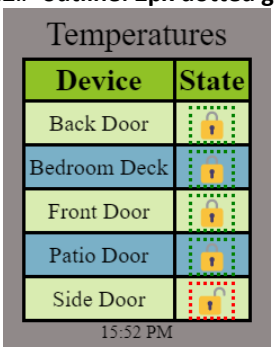
Additional Formatting Tags

In addition to the above-named tags there are four others that are not obvious which are shown below.

#Alternaterow#=outline: 2px dashed red;

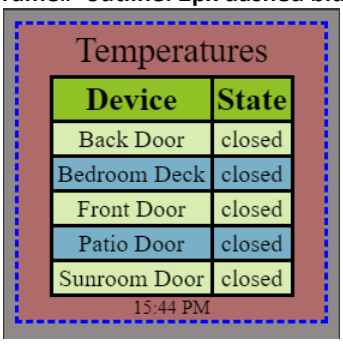


#high1#=outline: 2px dotted red; |
#high2#=outline: 2px dotted green;



Note: Alternate Rows not required.

With Frame enabled.
#Frame#=outline: 2px dashed blue;



Note: Alternate Rows not required.

- **#Alternaterow#:** When Alternate Row colors are enabled, these tags can be used to influence the content of alternate rows in ways not exposed through the UI.
- **#High1# and #High2#:** When highlighting is enabled and replacement text is specified, these tags will be present and can be used to influence the appearance of all affected rows.
- **#Frame#:** When a frame is enabled the #Frame# tag will be present and can be used to influence the properties of the frame.

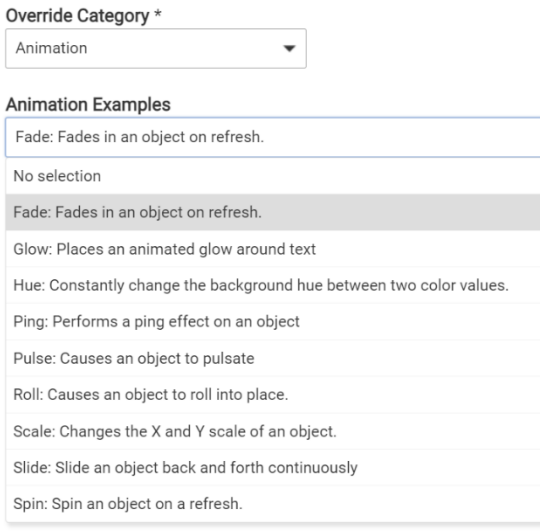
The #Class# Tag

This is the most powerful of all the tags as it allows new instructions to be introduced and applied to the table elements such as Title, Table, Border etc. With classes you can introduce interesting effects and animations to your dashboard. These will be covered after doing it the easy way which is to use the various #Class# examples available through the Overrides Helper which we will discuss next.

Creating Effects

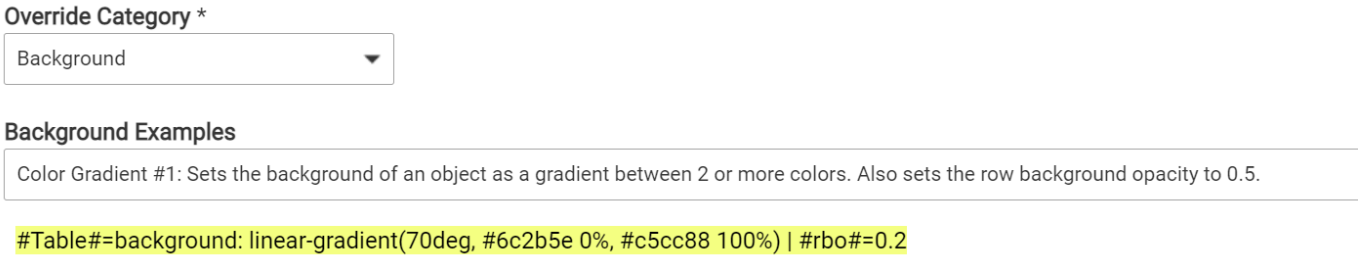
Where do these command strings come from? How do I know what I can put in here?

To get you started with effects, the **Overrides Helper** has over 50 built in examples that you can use to experiment with.

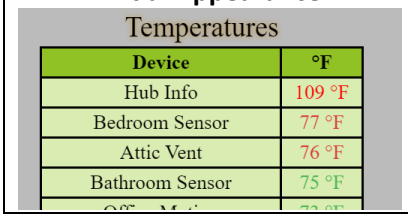
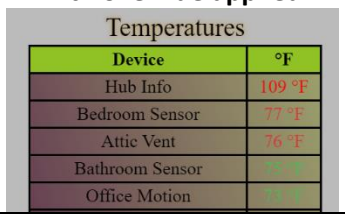
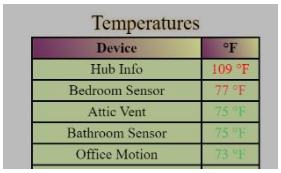


Simply **select the Category and Example** you wish to try, click on **Copy to Overrides** and then click **Refresh**. The displayed table will change to reflect the new settings. All the changes are created by the content of the overrides string applied to the present configuration.

Let us try this one which creates a gradient between two colors.



Copy to overrides and apply.

Initial Appearance	With override applied.	Edit Override, change #Table# to #Header#.
		

Why does the gradient not appear the same between the two examples? If you look closely at the override string it includes #rbo#=0.2 which sets the row background opacity to 20%. This affects the gradient in the first example, but not the second.

I pieced together most of these examples by using online CSS\HTML generators. One of the better ones I used can be found here: https://webcode.tools/generators/CSS . Looks complicated but it's quite easy if you follow the example.

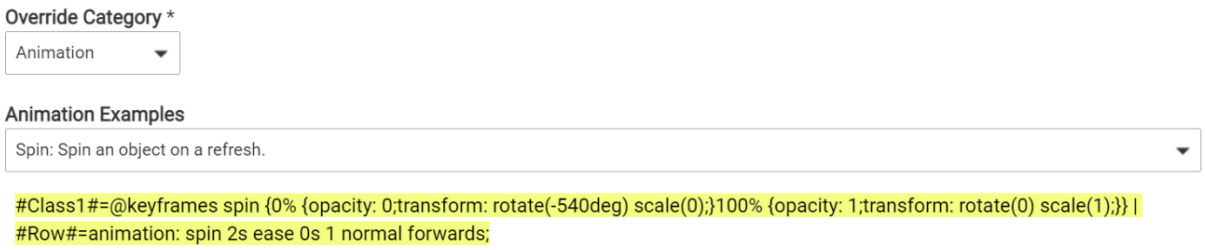
Now try some of the other examples. Try replacing #Table# with #Title#, or #Row# with #Header# etc. and watch what happens. This will help you get comfortable with some of the capabilities without having to type anything from scratch.

Once you feel comfortable with these you are ready to take a closer look at classes.

Classes

Classes are small pieces of code\style that can be called upon to change an object. These are common in CSS and can be utilized in **Tile Builder Advanced**. Even if you have never programmed don't worry, there are plenty of examples to edit. In **Tile Builder** there are 5 unique **#Class#** placeholders, #Class1# … #Class5#. You can use any one, but each one can only be used once in a tile.

Let us take one of the sample overrides and break it down.



Class Sample Part 1

The first part is the class and what is shown in red will be substituted into the HTML code in place of the **#Class1#** placeholder.

> **#Class1#**=@keyframes myAnim {0% {opacity: 0;transform: rotate(-540deg) scale(0);}100% {opacity: 1;transform: rotate(0) scale(1);}}

@keyframes indicates that it going to be a transition between 2 or more points enclosed in {}

myAnim is the name of the animation class and is used to activate it.

The first point is: **0% {opacity: 0;transform: rotate(-540deg) scale(0)}**

- 0% marks it as the starting point of the transformation.
- Transform means it's a transformation of the object with the following properties.
- Rotate (-540deg) means it will begin pre-rotated 540 degrees to the left.
- Scale (0) means that it will not change in size at its starting point.

The second point is: **100% {opacity: 1;transform: rotate(0) scale(1)}**

- 100% marks it as the ending point of the transformation.
- Transform means it's a transformation of the object with the following properties.
- Rotate (0deg) means it will end in a normal position (0 degrees rotated).
- Scale (0) means that it will not change in size at its ending point.

This animation will spin the object from -540 degrees to 0 degrees when implemented. The browser will make the animation smooth between the starting and ending points. It is possible to add other points in the animation, at say 50% with different properties if so desired, but they take up more precious space.

Class Sample Part 2

The '**|**' is just a required separator that **Tile Builder** uses to separate multiple instructions on a single line. It will be stripped out and discarded.

Class Sample Part 3

The last part is used to associate the class with some part of the HTML table, in this case the **#Row#** element.

#Row#=animation: myAnim 2s ease 0s 1 normal forwards;

This will cause **Tile Builder** to look for the **#Row#** placeholder and replace it with **animation: myAnim 2s ease 0s 1 normal forwards;**

animation: is a required element for HTML\CSS indicating the presence of the animation.

myAnim: Is a name and it must match the name of a class, in this case the name given earlier.

2s ease 0s 1 normal forwards;: These are parameters passed to the animation that control things like animation speed, duration, direction, reversal nature etc.

That is how to implement and call a class. The class will get activated at refresh.

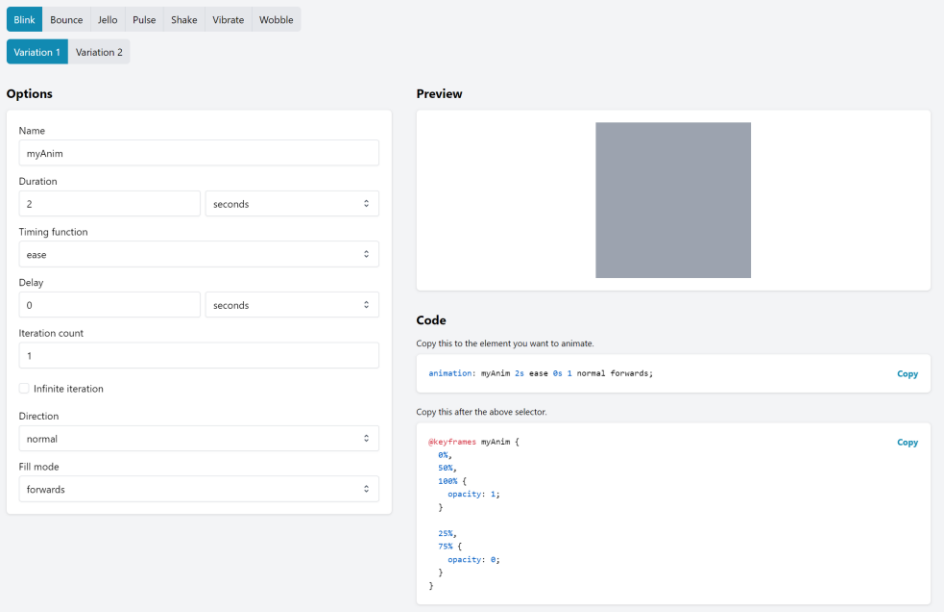
Very Important: The class and animation names must be unique across all tiles unless they are intentionally re-used. In the above examples instead of using **myAnim** you might use **spin** or **rotate**.

Online Resources for Classes

You don't need to write these commands from scratch. You can use a generator like this.

https://webcode.tools/generators/CSS/keyframe-animation

Enter the parameters that you want, and it will generate the command strings as shown below.



Turn the above strings into:

#Class1#=@keyframes myAnim {0%,50%,100% {opacity: 1;}25%,75% {opacity: 0;}}

#Title#=animation: myAnim 2s ease 0s 1 normal forwards;

Put them together with the '|' separator and paste the string into the **overrides** field like this.

#Class1#=@keyframes myAnim {0%,50%,100% {opacity: 1;}25%,75% {opacity: 0;}} |
#Title#=animation: myAnim 2s ease 0s 1 normal forwards;

You now have a title that blinks twice whenever the table is refreshed. This happens automatically on the dashboard when the table gets new data.

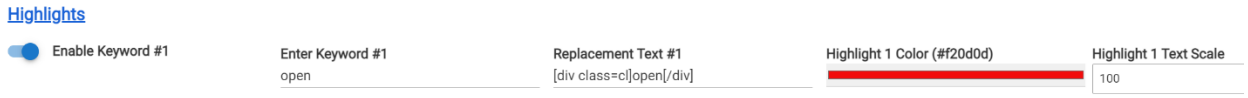
Important: When generating an event to test an animation it will take effect immediately on the dashboard due to the event subscriptions. This is not true within the Tile Builder UI so you must do a refresh manually to see event changes.

Static Class Example

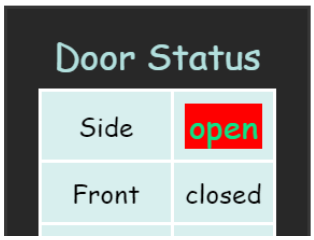
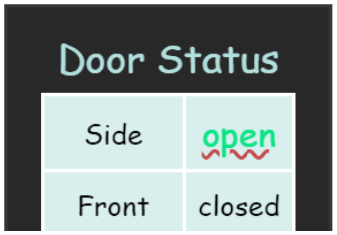
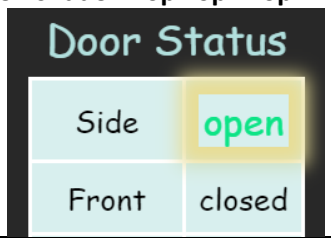
Classes can do all kinds of things and are not just limited to animation. In this example we create a class called "**cl**" which turns the background color to red. We use a leading period in the declaration, so it's "**.cl**" but referred to by using "**cl**". This name can be anything using a-z. This documentation was originally written when pace was at a premium so I opted for shorter class names.

#Class1#=.cl{background-color: #ff0000;}

We can call this class when we get a matching condition in **Highlights** by doing the following. Whenever the word **open** is detected, we call the class to change the background color.



The above example result is shown in the first cell. The required class definition is shown for other effects. The results looks like this:

#Class1#=.cl{background-color: #ff0000;} 	**#Class1#=.cl{text-decoration: underline wavy #C34E4E;}** 
#Class1#=.cl{outline: 2px solid red;} 	**#Class1#=.cl{box-shadow: 0px 0px 10px 10px #E8DD95;}** 

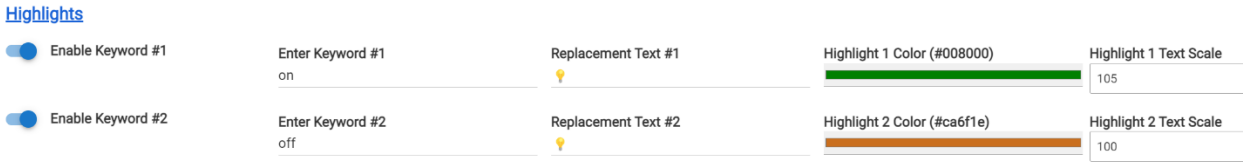
Remember: We used **#Class1#** in these examples but it could have been **1, 2, 3, 4 or 5**. Having 5 unique class placeholders just makes it easier to format and make changes when including multiple classes in a single tile.

If we were to look inside the HTML we would see the code **[div class=cl]open[/div]** repeated on each line where a contact is open. Naturally this takes up more space than just **open**. When you are considering these enhancements, it is best to apply these effects to a small number of results where space is a concern.

The Highlights Class Example

Whenever you have highlights activated the table will now contain classes with the **#high1#** … **#high5#** tags. These can be used to add formatting to the data column to create various effects. In the following table we use these tags to create an effect of lights either being on or off.

First, we must use **Highlights\Keywords** to swap a light bulb emoji in for the normal text which would be on or off as shown below.



Then we must add information to the highlight classes using the **#high1#** and **#high2#** tags. To do this go to **Advanced**, **Enable Overrides** and then paste the string below into the **Settings Overrides**.

#high1#=text-shadow: 0px 0px 10px #ffff00; | #high2#=opacity:0.5

When the table is generated **text-shadow: 0px 0px 10px #ffff00;** will be added to the class that formats **Keyword #1** and **opacity:0.5** will be to the class that formats **Keyword #2**. Bulbs that are on will appear at full brightness and glow because of the text gradient. Bulbs that are off will appear dim because the opacity has been set to 0.5. The net result is a pleasing differential between bulbs that are on and those which are off.

Animated Icon Examples

In this section we will cover how you can animate icons to give your dashboard a more appealing look. This is achieved by applying an animation class to a particular device state such as 'open'.

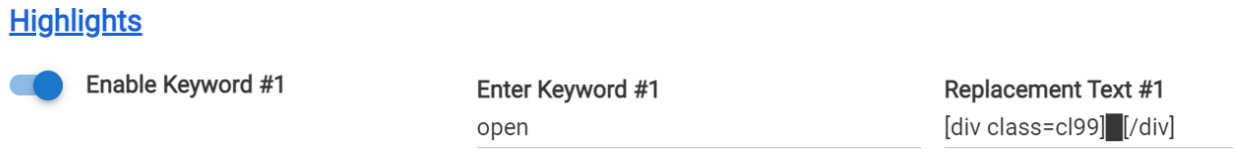
Slide Example (Classes Example 6)

The following line of code is a CSS animation that will move an element 20 pixels to the left at the start and end at 20 pixels to the right of its normal position. This is example 5 in the classes group.

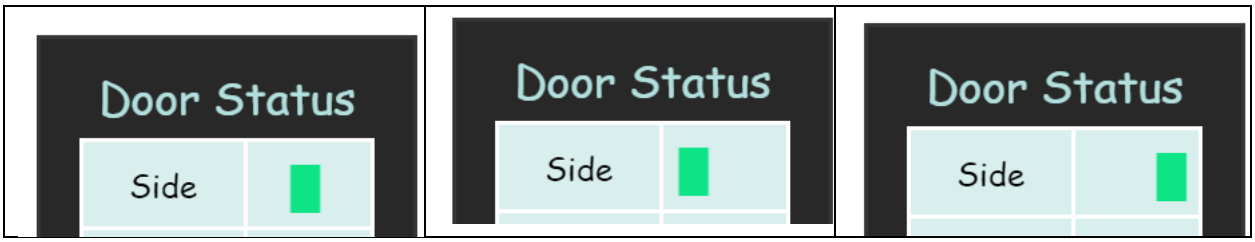
#Class1#=.cl99{animation: myAnim 1s linear 0s infinite alternate-reverse;}

@keyframes myAnim {0% {transform: translateX(-20px);}100% {transform: translateX(20px);}}

Using the same trick as the prior example we can add this animation for any open door.



Using an ASCII block character combined with the animation class we get a pleasing blob that moves back and forth smoothly and continuously until the door is closed.



You don't have to write these statements by hand. Use the **Overrides Helper** examples or an HTML\CSS generator such as https://webcode.tools/generators/CSS/keyframe-animation.

Spinner Example (Classes Example 5)

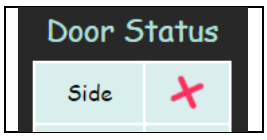
Spinners are commonly used to indicate activity and a variety of characters are suitable. I chose the emoji character ❌ but extended ASCII characters such as ☼ , ֍ , ✳ are well suited to indicating a spinning motion. With Unicode characters you also get the option of specifying the character color and size. With emojis you can only specify size. Try this out.

Copy this line to your overrides field, this is example 6 in the classes group.

#Class#=.cl99{animation: myAnim 1s linear 0s infinite normal forwards;} @keyframes myAnim {0% {transform: rotate(0deg);}100% {transform: rotate(360deg);}}

Change the replacement text in **Highlights** to **[div class=cl99] ❌ [/div]**

The result is a red X that will spin continuously when the door is open. Spinners are a good option for motion, contacts, switches, fans etc.



A Word About Classes

Before you start creating classes there are a couple of things you must know.

Scope

The scope of a class is global. If you set up a class via tile A you can reference it using tile B. The good news is that can save space, the bad news is it can have unintended consequences if you are not careful about naming your classes.

Naming

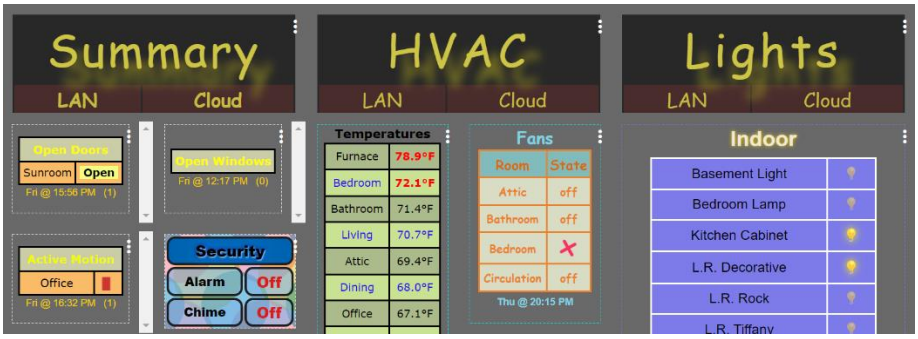
Each unique class or animation must have a unique name. Because of space constraints I'd recommend you name classes cl1, cl2, cl3 etc. and animations an1, an2, an3… etc. This should avoid naming collisions. If you have two classes by the same name anywhere in the dashboard only one of them will work. If something is spinning when it should be sliding etc, your class names are the problem.

Advanced Techniques

This section covers some techniques I have discovered. Even though I wrote **Tile Builder** I'm still finding new ways of doing things which may be of interest to others.

Create a Menu Block

I like to group topics on the dashboard. In the picture below I've chosen to do vertical groupings and put a title block at the top.



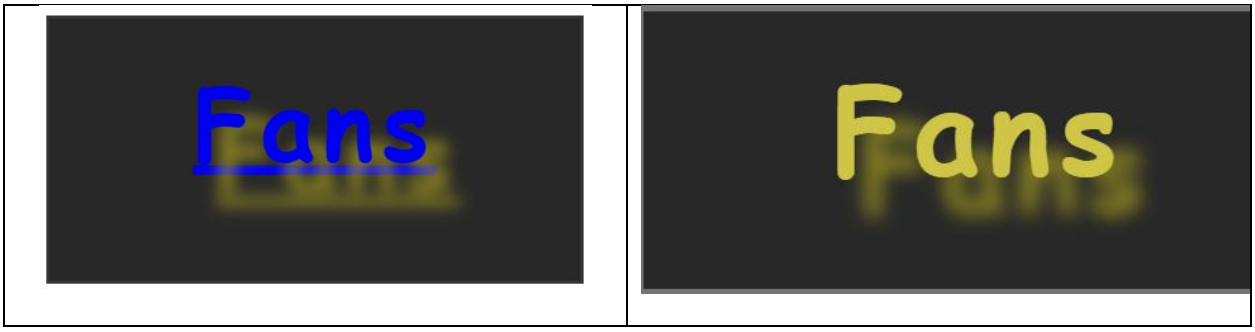
Here's how to do this.

1. Add a new **Activity Monitor** but don't select any attributes or devices.
2. Set the **Device Limit Threshold** to 0
3. Go to **Styles** and select the style **Menu Bar** and apply it**.**
4. Got to the **Title** tab and change your text. I chose fans and it looks like this.

5. Give the tile a name and select a tile number to publish it on, set the **Table Refresh Period** to **Never** and then add it to your dashboard.
6. If you want to use it as a dashboard link you can by doing this.
    a. Edit the tile and go to the Title tab.
    b. Get the link to the dashboard and enter it like this:

    [a href='http://192.168.0.200/apps/api/3204/dashboard/3204?access_token=aaa-bbb-ccc&local=true']Fans[/a]

Your tile will now look like this in the preview window and on the dashboard.



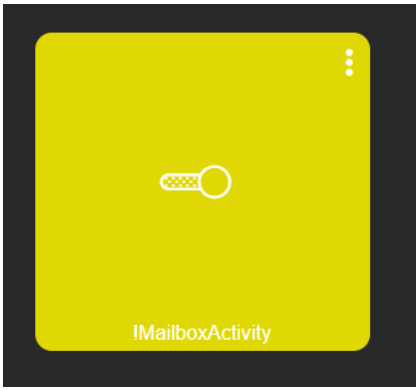
You can use the **Menu Bar Dual Links** style to add two links style by replacing the two header values with the appropriate links you can have easy access to either the **Local** or **Cloud** dashboard. It will look something like this but obviously you can style it to your own needs.

Creating an Animated Icon

One of the benefits of **Tile Builder** is the density we get with tabular results. But it is possible to generate tiles that use icons (which can be animated) to reflect the state of an individual device when that is desired. Creating animated icons requires **Tile Builder Advanced**.

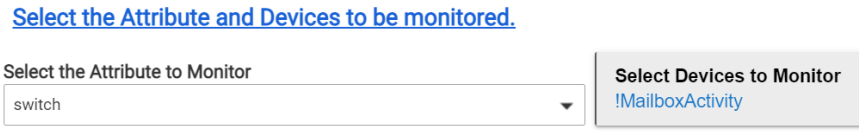
This approach is a highly functional workaround and may be replaced in the future by something more specific to the task that would support a font Icon pack of some sort.

In my house I use a virtual switch to indicate whether the mail has been delivered or not, driven by a contact sensor in the mailbox and a few rules. If I add that virtual switch to the dashboard it looks like this.

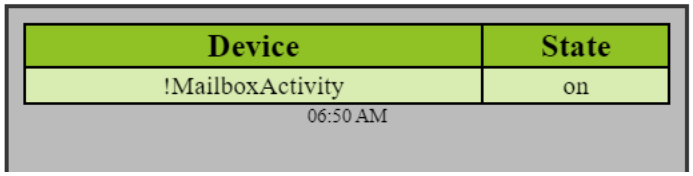
Default appearance:	When activated:
	

To make an icon version of this use **Tile Builder** and perform the following steps:

1. Create a new **Attribute Monitor**
2. Select and attribute and **single device** to monitor.



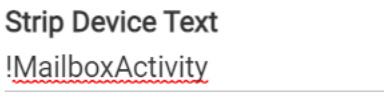
3. You should see something like this.



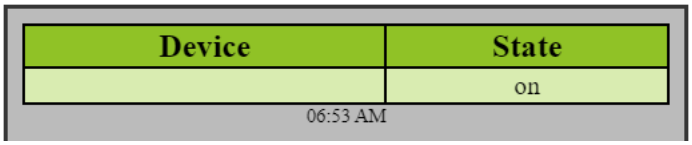
Device	State
!MailboxActivity	on

06:50 AM

4. Here is the trick. Add the name of the device to the **Strip Device Text** field like this:

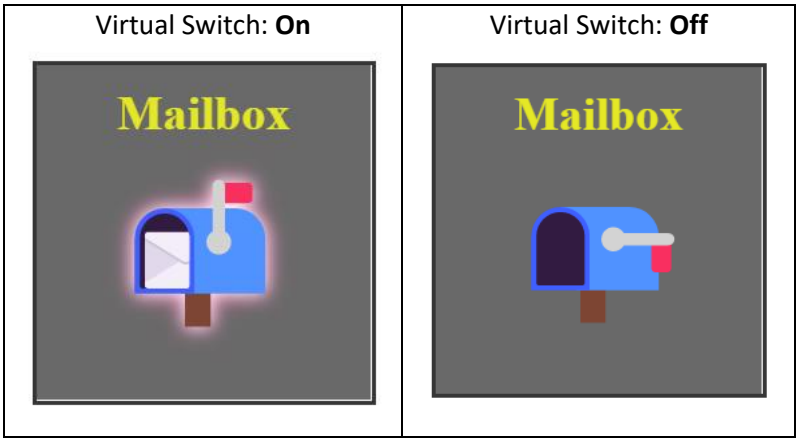


And now it looks like this:



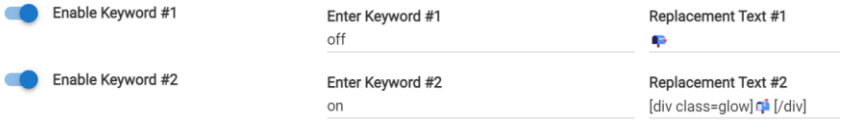
5. Select **Customize Table** and add a title.
6. Go to the **Styles** tab and apply style **Style-AM-Mailbox**.
7. On the **General** tab change the **Tile Preview** size to **1 x 1**

Depending on the state of your switch\contact it will look like one of these:



How to Change Icons\Emojis

Go to the highlights tab, it should look like this.



Simply replace the two mailbox characters with whatever values you would like to use instead.

How to Change Animation Effects

First let's review where the animation effect comes from. Under the **Advanced** tab your **Settings Overrides** will look like this as they are part of the Style.

```
#Class#= .glow {animation: glow 5s ease-in-out infinite alternate;} @keyframes glow {from {text-shadow: 0 0 5px #fff, 0 0 10px #fff, 0 0 15px #e60073;}to {text-shadow: 0 0 10px #fff, 0 0 15px #ff4da6;}} | #row#=text-align: center |#rp#=0 | #bp#=-5
```

The statement **#Class#= .glow……** specifies the parameters for this animation. You can change these parameters or introduce a totally different animation using examples in the **Overrides Helper** or from the Animated Icon examples described previously.

It's a Wrap

Well, if you made it this far you are ready to exploit most of the power of **Tile Builder** to build beautiful, functional, and even animated tiles to make your **Hubitat®** dashboard a lot more fun and engaging. I look forward to seeing some of the designs that people come up with and share on the community forums.

I have several ideas for other **Tile Builder** modules of the same quality as **Activity Monitor** and **Attribute Builder**. I will almost certainly write them for my own use, but their public release will depend on the **Hubitat®** communities' willingness to acknowledge value in quality software and donate towards the ongoing development of the project.

This is the day the Lord has made, let us rejoice and be glad in it.

Appendix A

Tile Builder Field Code Reference

General	Code	Units	Example	Notes
Table Width	#tw#	%	#tw#=85	Auto or a % of the container width.
Table Height	#th#	%	#th#=90	Auto or a % of the container height.
Border Mode	#bm#	String	#bm#=Separate	Options are Collapse and Separate
Background Color	#tbc#	#Hex	#tbc#=#e88c8c	This background color is visible when border mode is separate.
Font Family	#tff#	String	#tff#=Arial	Text Font Family for all text

Header	Code	Units	Example	Notes
Text	#A0# for Column 1 & #B0# for Column 2		#A0#=My Title	Any text string. You can include Emoji or HTML codes using square brackets [b]Bold[/b].
Text Size	#hts#	%	#hts#=125	Any integer value.
Text Color	#htc#	#Hex	#htc#=#000000	Any Hex color.
Background Color	#hbc#	#Hex	#hbc#=#232323	Any Hex color
Background Opacity	#hbo#	Float	#hbo#=0.3	An opacity value in the range 0 – 1. These are combined with the Hex colors to form RGBA values in the HTML.
Text Alignment	#hta#	String	#hta#=Left	Options are: Left, Center, Right and Justify
Text Opacity	#hto#	Float	#hto#=0.7	An opacity value in the range 0 – 1. These are combined with the Hex colors to form RGBA values in the HTML.
Padding	#hp#	Px	#htp#=10	An integer, typically in the range of 0 – 10. N/A if border padding is active.
Spacer	#br1#	String	#br1#=[br]	Adds string before the header. For example, a line break may look better when frame is enabled.

Title	Code	Units	Example	Notes
Text	#tt#	String	#tt#=My Title	The Title Text
Size	#ts#	%	#ts#=150	Expressed as a % of the base font size.
Color	#tc#	#Hex	#tc#=#00FF00	Color expressed as a 6 digit Hexadecimal
Alignment	#ta#	String	#ta#=Right	Options are: Left, Center, Right and Justify
Padding	#tp#	Integer	#tp#=3	Change the space around Title.

Border	Code	Units	Example	Notes
Style	#bs#	String	#bs#=Solid	The border style
Width	#bw#	Pixels	#bw#=5	The border width in pixels
Color	#bc#	#Hex	#bc#=#00FF00	The border color
Radius	#br#	Integer	#br#=20	The border radius in pixels
Padding	#bp#	Integer	#bp#=3	The border padding in pixels

Rows	Code	Units	Example	Notes
Text Size	#rts#	%	#rts#=100	Expressed as a % of the base font size.
Text Color	#rtc#	#Hex	#rtc#=#000000	Any Hex color.
Text Alignment	#rta#	String	#rta#=Center	Options are: Left, Center, Right and Justify
Text Opacity	#rto#	Float	#rto#=0.8	An opacity value in the range 0 – 1. These are combined with the Hex colors to form RGBA values in the HTML.
Padding	#rp#	Px	#rp#=5	An integer, typically in the range of 2 – 10. Not applicable if border padding is active.
Background Color	#rbc#	#Hex	#rbc#=#FF00FF	Background color of the row area.
Background Opacity	#rbo#	Float	#rbo#=0.5	Background Opacity.
Alternate Text Color	#ratc#	#Hex	#ratc=#0000FF	The alternate text color when alternate table rows are configured.
Alternate Background Color	#rabc#	#Hex	#rabc#=#D3D3D3	The alternate background color when alternate rows are configured. Change background opacity to make visible.

Footer	**Code**	**Units**	**Example**	**Notes**
Text	#ft#	String	#ft#=My Footer	The footer string. You can use %day%, %time% as a macro.
Size	#fs#	%	#fs#=75	Percentage of the base font size.
Color	#fc#	#Hex	#fc#=#5c290d	The color of the footer text
Alignment	#fa#	String	#fa#=Right	The alignment of the footer text.
Spacer	#br2#	String	#br2#=[br]	Adds string after the footer. For example, a line break may look better when frame is enabled.

Appendix B

Hubitat® Dashboard CSS

Here are some common statements used in CSS to improve the overall appearance of the Dashboard that are useful with Tile Builder.

- **Make a Tile Transparent:** #tile-1 {background-color: rgba(128,128,128,0.0) !important;}
- **Hide the Tile Device Name:** #tile-1 .tile-title {visibility: hidden; display: none !important}
- **Enable Scroll Bars:** #tile-1 {overflow-x: hidden !important; overflow-y: scroll !important;}
- **Change Tile Vertical Alignment:** Align Top: #tile-1 .tile-primary {vertical-align: top;}
- **Outline a Tile with a border:** #tile-1 .tile-primary {outline: 1px dotted white;}
- **Change a Tile margin or padding:** #tile-1 .tile-contents {margin:0px ; padding: 10px;}

Sample Classes
These classes can be used within **Tile Builder** or off-loaded to the dashboard CSS. These are just examples, you can add your own.

```
.slide{animation: slide 1s linear 0s infinite alternate-reverse;} @keyframes slide {0% {transform: translateX(-20px);}100%
{transform: translateX(20px);}}

.ping{animation: ping 2s ease 0s infinite normal forwards;} @keyframes ping {0% {opacity: 0.8;transform: scale(0.2);} 80%
{opacity: 0;transform: scale(1.5);} 100% {opacity: 0;transform: scale(2.2);}}

.blink{animation: blink 2s ease 0s infinite normal forwards;} @keyframes blink {0%,50%,100% {opacity: 1;}25%,75% {opacity:
0;}}

.wobble{animation: wobble 2s linear 0s infinite normal forwards;} @keyframes wobble {0%,100% {transform:
translateX(0%);transform-origin: 50% 50%;}30% {transform: translateX(15px) rotate(6deg);}45% {transform: translateX(-15px)
rotate(-3.6deg);}60% {transform: translateX(9px) rotate(2.4deg);}}

.roll {animation: roll 1s linear 0s infinite alternate forwards;} @keyframes roll {0% {opacity: 10;transform: translateX(-
25px) rotate(-200deg);}100% {opacity: 1;transform: translateX(25px) rotate(200deg);}}

.pulse {animation: pulse 2s linear 0s infinite normal forwards;}  @keyframes pulse {0% {transform: scale(.75);}50% {transform:
scale(1.25);}100% {transform: scale(0.75);}}

.fade {animation: fade 1s linear 0s infinite alternate forwards; }  @keyframes fade {0% {opacity: 0.5;}100% {opacity: 1;}}

.swirl {animation: swirl 2s ease 0s infinite normal forwards;} @keyframes swirl {0% {opacity: 0.25;transform: rotate(-360deg)
scale(1);}100% {opacity: 1;transform: rotate(0) scale(1);}}

.spin{animation: spin 1s linear 0s infinite normal forwards;} @keyframes spin {0% {transform: rotate(0deg);}100% {transform:
rotate(360deg);}}

.glow {animation: glow 5s ease-in-out infinite alternate;} @keyframes glow {from {text-shadow: 0 0 5px #fff, 0 0 10px #fff, 0
0 15px #e60073;}to {text-shadow: 0 0 10px #fff, 0 0 15px #ff4da6;}}

.glow2 {background: radial-gradient( #F6E105 30%, #7b7bea 80%);}

.backred{background-color: #ff0000;}
.wavy{text-decoration: underline wavy #C34E4E;}
.outline{outline: 2px solid red;}
.bs{box-shadow: 0px 0px 5px 5px #E8DD95;}
```

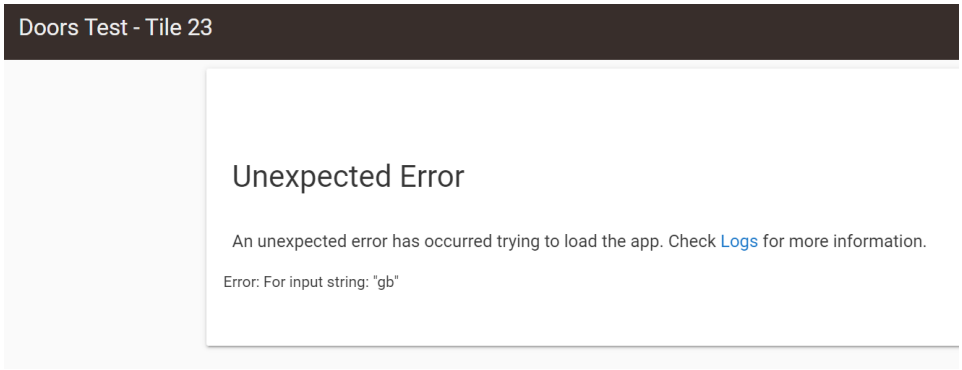
These can be called in Tile Builder using one of these two methods.
Highlights: Use - **[div class=slide] ↔[/div]**
Overrides: Use - **#Row#=animation: slide 2s ease 0s 1 normal forwards;**

Important: Off-loaded classes will not work within **Tile Builder UI** because the class only exists on the Hubitat® Hub. However, re-publishing a tile will cause an update to occur on the dashboard and you can observe your changes this way.

Appendix C

Recovering Failed Tiles

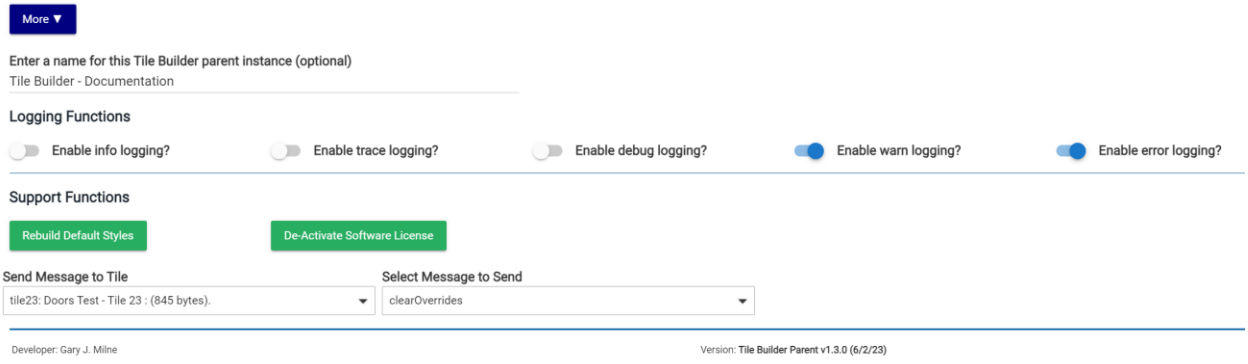
Eventually you will run into a condition where you make a change to a tile and you get an error condition like this.



This is probably the result of a recent change creating an error condition that the program does not catch. A common cause is the use of Overrides which are incorrectly constructed. Other issues can pertain to certain devices returning null values or Keywords\Thresholds with badly formatted values.

Prior to version 1.3.0 your only option was to delete the tile and start over. With 1.3.0 you now have a good chance of recovering the tile. Here is how you do it.

Go to the Parent App and open the **More** section which will look like this.



Under **Send Message to Tile** select the problematic tile in the dropdown list.
Under **Select Message to Send** select the option that relates to the last change you made. In this case I chose **clearOverrides**.
Now go to the problematic child app and do a refresh of the browser. In this case the Overrides were cleared and now the app loaded successfully.
If the app failed to load you can try the other recovery options and repeat the process.

Once the child app has been recovered you must go to the parent app and reset the recovery options to prevent them running each time the app is refreshed.