Theory of Operation

This is a short document outlining the basic theory of operation of the Node Red flow for those more interested in its workings either for better understanding or desires to modify. As a fair warning this document will not be updated with each update to the node red flow. It's purpose is to give a general overview of how the flow works, not document every detail. I will update it when major features are added.

WLED + Lamp Overview

WLED is simply an LED controller, it has no knowledge of the weather or other processing that is going on inside node red. There are limited outputs from WLED, but some robust inputs, which we make use of.

For outputs, WLED will report status over MQTT. If you're not familiar with MQTT, there is a basic overview here. The short version is that WLED will output messages that report things like color and brightness when it changes the LED effects. We're using this as the trigger for different events in the Node Red flow. The three save states that are setup during the initial configuration set 3 specific colors that the node red flow then looks for.

These three colors are ones that you are very unlikely to ever see during normal use, so it shouldn't false trigger. These colors are #010000, #000100, and #000001, which is basically off, but with red, blue, and green on at their lowest possible setting for each one respectively. The Lamp is set to load save state 1, which is #010000 at boot, which then triggers the node red flow to kick off and start. Short pressing the button loads save state 2, #000100, which then triggers the forecast mode, and long pressing loads save state 3, #000001, which triggers the triple color band mode.

The reason we don't save the weather animations in WLED is twofold. First, there aren't enough save states for all the needed animations. Second, WLED won't save multiple segments in any of the save states except for 16. So instead we save them in the node red flow and then tell the WLED controller what to display every time we need to change. Commanding WLED was made really easy by allowing it to receive json data that sets it's state. So we save the json strings for each animation and send the desired one when called for.

Node Red Flow Overview

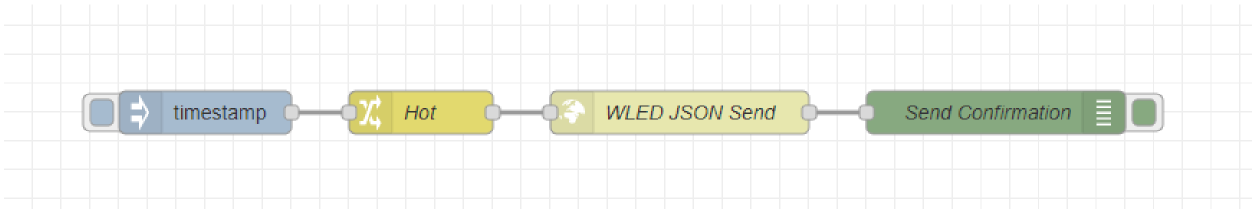
Node Red Basics

There are a million node red tutorials and overviews out there, I will not go deep into it's theory of operation. My intent here is to explain just enough that what I outline below makes sense. The most basic explanation is that node red is a python based visual coding environment. You have a multitude of different nodes you can link together to build flows. These nodes pass and modify messages. The messages are the heart of how node red works, each node is either triggered by, passes, modifies, or generates messages which then move through the flow. These messages can be very complicated with multiple parts, headers, topics, etc, or they can be simple minimal just used for triggers. Again, this is not a Node Red tutorial, I just want the concept of messages to be introduced.

Basic WLED Control from Node Red

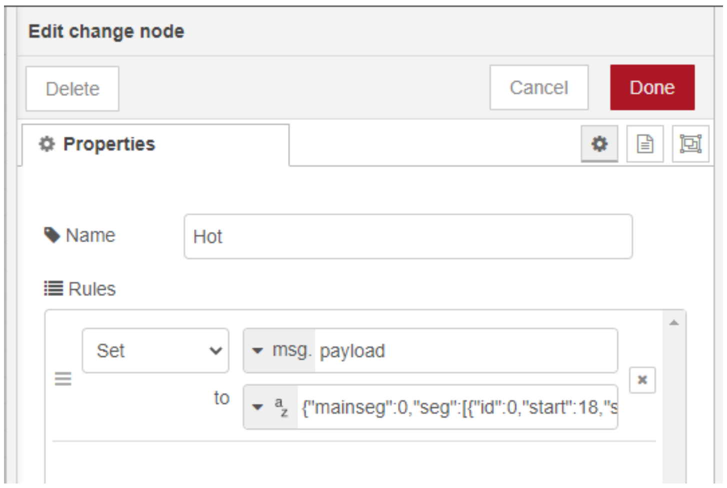
Before I dive into the full flow, I want to quickly go through the basic way we control WLED from Node Red. There are 3 basic steps to sending data to the WLED controller.

1. Something triggers the actual need to send data to WLED
2. We set the message payload to be the desired json string
3. We then actually pass this string to WLED over the local network



In this above super simplified flow, our trigger is a manual trigger that's just injecting a time stamp to the flow. This inject mode requires human interaction to press the button. In the actual flow the triggers come from various automated sources.

Setting the message payload to the json string happens in the node labeled "Hot". This is a change node where we are changing the payload from the time stamp coming from the trigger, and replacing it with the long json string that defines the "Hot" animation. This node then automatically passes the new payload long.



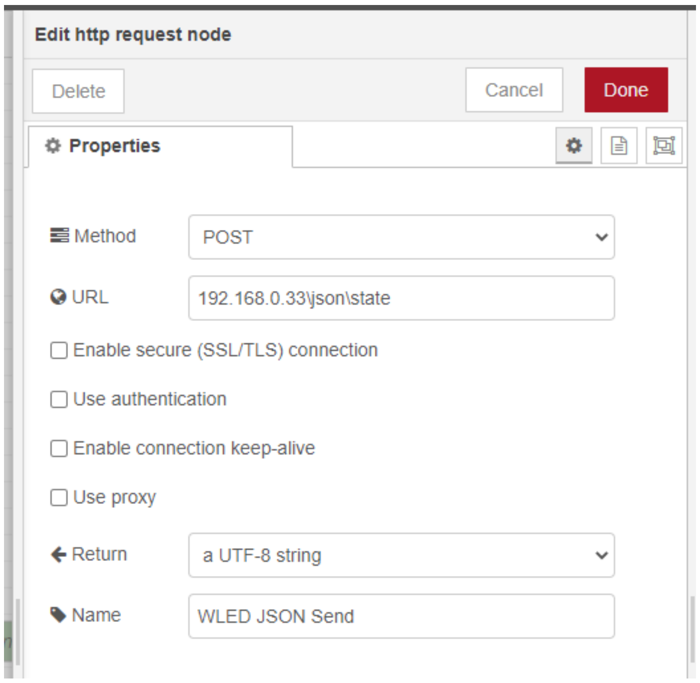
Name: Hot

Rules

Set msg. payload

to {"mainseg":0,"seg":[{"id":0,"start":18,"s

Then we pass the message to WLED using the HTTP request node. This node can be used in a couple ways, but we're using it to post data to a URL. In this case we are posting the message payload (the json string) to the IP of the WLED controller.



Edit http request node

Method: POST

URL: 192.168.0.33\json\state

☐ Enable secure (SSL/TLS) connection

☐ Use authentication

☐ Enable connection keep-alive

☐ Use proxy

Return: a UTF-8 string

Name: WLED JSON Send

The output from this HTTP request node will depend on how we're using it. If we're requesting data like outlined later in this document, it will output the received data. However, since we're using this as a POST, all we will get back is a confirmation of success, or notice of failure. This message we are just passing to the debug window, which I used during development.

User Defined Variables

These nodes are the ones that get setup during initial configuration. If I were building this flow just for my own use this section wouldn't exist and these nodes would be located in the various sections lower in the flow, but since I wanted everyone to be able to use this I located them all in a single place and used link in/out nodes to route the flow.

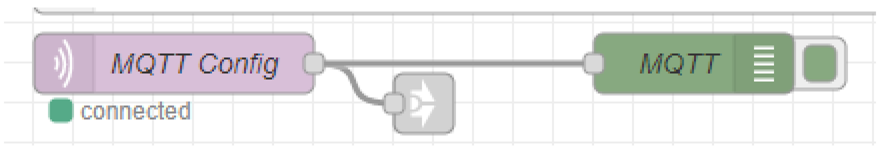
I won't go too into detail here since most get covered below or in the setup instructions. But the basic structure is as follows. The first bunch of nodes are the ones that MUST be configured by the user for the flow to run. MQTT, weather API, sun-events and the WLED send nodes all must be configured to your specific setup or the flow won't work.

The update trigger and delay node can be tweaked, but don't need to be other than enabling the trigger after setup.

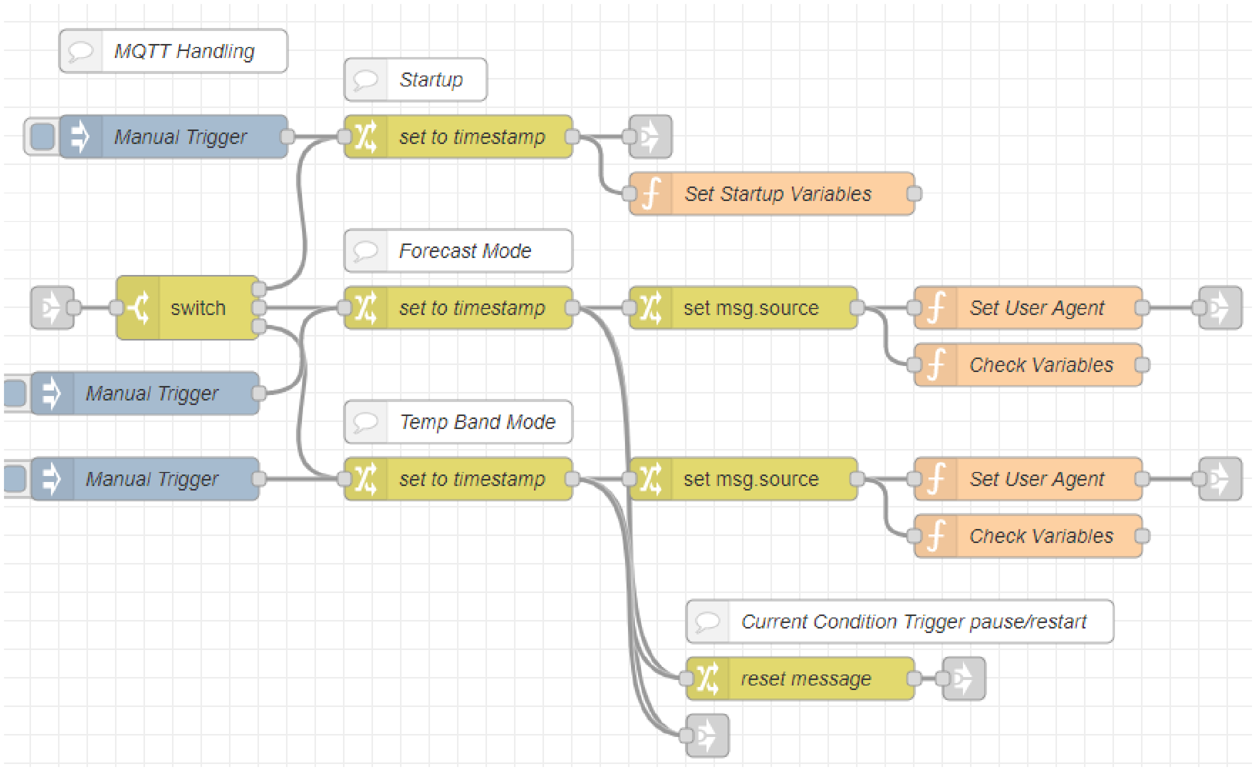
The save state setup nodes could be deleted after initial setup if you wanted to, they are there just to help make setup easy.

MQTT Handling

Starting in the User Defined Variables section, we first configure the flow to listen to your MQTT broker.



This setup node very simply connects to the MQTT broker and listens for messages on the specific wled/weather/c topic, which is where the color of segment 0 is defined. From there it kicks the received messages both to the debug window and down to the MQTT handling section.



This section is basically a 3 branch switch. The manual triggers are just there for debugging, and may no longer exist depending on how many times I've updated the flow. The switch node reads the message received by the MQTT node, then then passes it to one of the three branches if it is one of the three trigger colors (#010000, #000100, #000001).

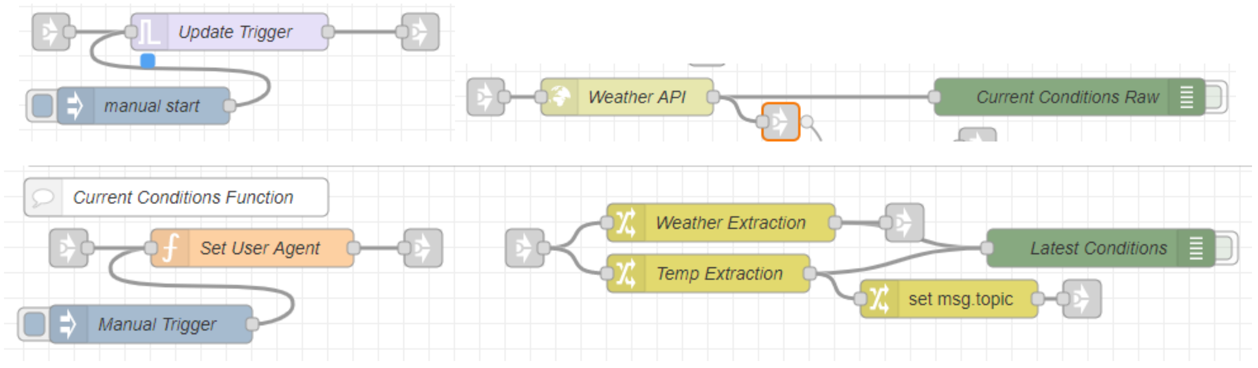
In the case of #010000, this is what the lamp sends on startup, so the message is passed to the startup branch where we change the message payload back to a simple timestamp, and pass the message off to the update trigger in the User defined variables section. This kickstarts the repeating trigger that polls for the current conditions. We also set some flow variables in the function node that are used elsewhere just for checking if things have changed.

In the case of #000100, this is what is sent on a short button press to trigger the forecast mode. Once again we change the payload back to a time stamp and then we trigger two branches. First we again change the message but this time instead of the payload we change msg.source to be "forecast" and then pass the message long to the Forecast API node in the User defined variables section which will kick off the forecast function. At the same time we set some flow variables again used for checking if things changed. The other branch triggers the nodes that pause the normal current conditions trigger. We do this so that the forecast isn't overridden if it's timed right before the current conditions trigger runs. The reset message is sent to that trigger to stop it, and we also trigger the delay node also in User defined variables which will restart the same trigger node after the defined delay time.

Lastly, in the case of #000001, which is the long button press, we do the same thing as for the forecast branch withone exception. In this case we set msg.source to be "tricolor" instead of "forecast". This is because both of these branches use the forecast API node and we need to know where to send the received data so that the right function is triggered. If msg.source is "forecast, it passes to the forecast functions, if it is "tricolor" it passes to the color bands function. The branch triggers the same reset and delay nodes that the forecast branch triggers.

Current Conditions Function

This is the function that runs most of the time. It polls the weather.gov API on a regular interval and extracts the current conditions and temperature which are then used to set the animations for WLED.



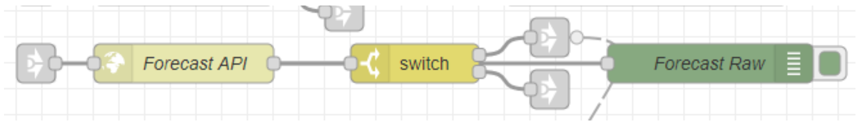
The function starts with the update trigger in the User Variable Definition section. The trigger node gets activated when the lamp is powered up, and then sends out a message every 5 minutes (or whatever time interval you set). This message is passed to the Set User Agent node in the Current Conditions Function section. This node sets the user agent data that gets sent to api.weather.gov along with the request. If you don't set this user agent data the api will think the request is from a bot and won't respond.

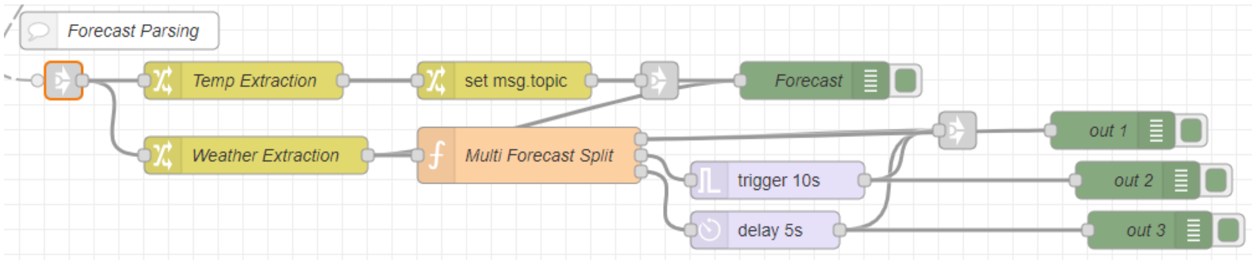
The User Agent node then passes to the Weather API node in the User Variable Definitions section. This is another HTTP request node that is using the GET function to request data from api.weather.gov for the observation station you set at the initial setup. The raw message from the API then gets passed back to the Current Conditions Function section (and passed to the debug window if enabled).

The two change nodes extract either the current conditions or temperature from the raw message and pass along only those pieces. The Weather Extraction passes the conditions to the animation selection section. The Temp Extraction node then passes to a node that changes msg.topic to "single" and then sends the extracted temperature to the Temp Color Handling section.

Forecast Parsing

The Forecast API node will pass the raw forecast received from api.weather.gov will either be passed to the Forecast Parsing or Tri Band Function sections depending on what msg.source is set to (how this is set is covered above). If msg.source is “forecast” it gets sent to the Forecast Parsing Section.



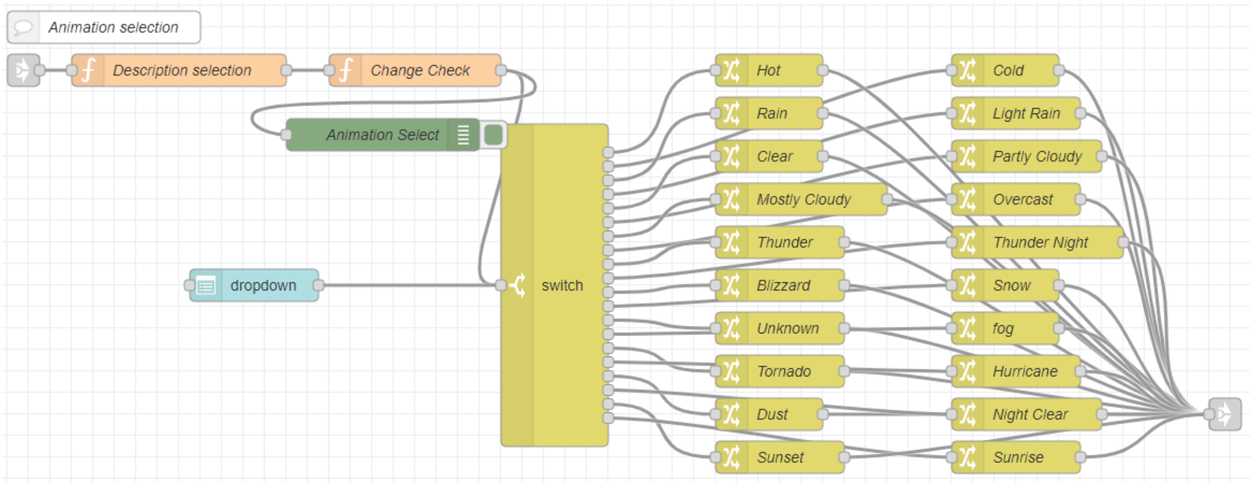


Similar to the Current Conditions Section, the temperature and weather conditions for the next day are then extracted and passed down the flow. The Temperature branch sets the msg.topic to “single”, and then passes the temperature to the Temp Color Handling Section.

The weather extraction node pulls the short summary for the next day. Very often these summaries are a short sentence along the lines of “showers then mostly cloudy”. This description won’t match any of the conditions defined in the animation selection section, so we have to break it up into it’s pieces. The Multi Forecast Split node divides the description at “then” and passes the pieces to different branches. If the description doesn’t have multiple parts to start, it gets passed to the first output and sent directly to the animation selection section. If there are multiple parts It will send the first, then 5 seconds later send the second, then 5 seconds after that re-send the first part. The intent here is to show both parts of the forecast as separate animations, then show the first part until the delay node finishes and re-starts the current conditions function.

Animation Selection

This is the section that picks the animation to pass to the WLED controller, and I promise it is not quite the mess it looks like. There may be a more elegant way to implement this but this is the path I started down.



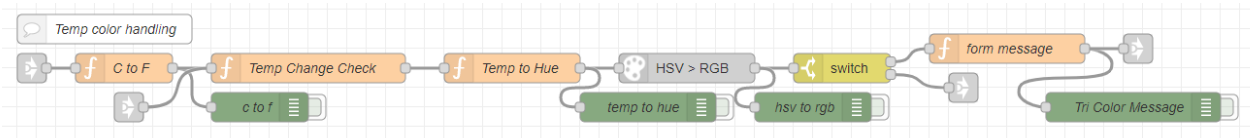
The description selection function first compares the incoming message to an enormous if/else if function. The point here is to group the multitude of weather summaries used by weather.gov into similar groups and then select the animation associated with that group. Also in this function is the logic to select day/night versions of animations where they have been defined.

The change check function compares the animation selected in the previous function to the last animation sent to WLED. If they are the same, nothing happens, no new animation is sent out. This is so the animations are reset every time the current conditions trigger goes off as this can be visually disrupting. If the new animation is different from the last one sent it passed along and also logged to the flow variable for the next time the check happens.

The next massive switch function I can probably condense into a function node, but again, this is the path I went down. The switch node routes the message along to the change node corresponding with the animation we want. The change nodes set the message payload to the json string for that animation, which is then passed to the WLED JSON Send node up in the User defined variables section.

Temp Color Handling

This section takes the current or forecasted temperatures and converts them to a color to be displayed at the bottom of the lamp.



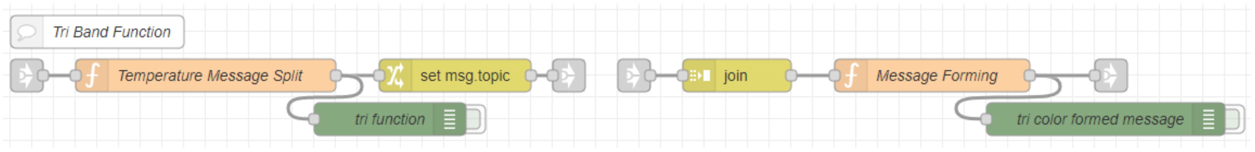
For whatever reason the current observations reports temperature in C, while the forecast reports in F. For that reason the current conditions are converted to F so that things are uniform. The temp change check node is similar to the check node in the animation selection section where the temperature coming in is compared to the stored temperature that was passed last update. If they are the same the function stops, if they are different the new temperature is passed long and recorded.

Temp to Hue takes the temperature and maps it to an HSV (Hue, saturation, value) formatted color. This is done with a custom formula. This function passes an array with the three pieces of information in the HSV format. This function also checks the day/night variable and changes the brightness accordingly.

HSV > RGB is a node explicitly for changing the HSV format to the RGB that WLED uses. The switch node checks msg.topic to know if the temperature that was just converted needs to go directly to WLED, or if it's being passed to the color band function to be assembled. As long as the topic is "single" the RGB payload is sent to the "form message" function which converts it to the correct format to instruct WLED what color to set segment 0 to. This formatted message is then passed along to the WLED JSON Send node to be sent to WLED.

Tri Band Function

This function is triggered by the long press of the button on the lamp. It requests the forecast and then pulls the high/low temp for the next 3 periods of the forecast. These will either be today, tonight, tomorrow, or now (night), tomorrow, tomorrow night. These highs/lows are then converted to a color and assembled into 3 rings on the lamp with the current condition on the bottom, and the furthest away on top.

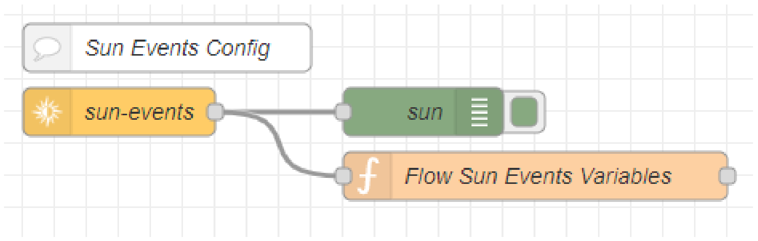


The raw forecast gets passed to this section when msg.source is set to tricolor in the MQTT section. The Temperature Message Split function then extracts the high/low temperatures from the raw forecast and then passes them along one by one. The change node sets msg.topic to "tri" before sending them to the Temp Color Handling section. Setting this topic makes sure the converted temp to RGB data comes back to this function.

The join node collects the three converted RGB messages and then sends them to the next function. The Message Formatting function builds the JSON message with the three colors and then sends it to the WLED JSON Send Node in the User defined Variables section to be sent to WLED.

Time Dependent Animations

Functionality is being added to change the animations depending on the time of day. Doing so allows there to be day/night versions of animations to better depict the outside conditions. Currently this is achieved with the Sun-events node.



Sun-events calculates various solar events for each day and then sends out messages at the start and end of each period. For example sunrise start and end, dawn, dusk, night, etc. Depending on the event being reported several flow variables are set to be used elsewhere. Currently these variables define if it’s day or night, and if it is currently sunrise or sunset. These may expand in the future.