

USING A CUSTOM SETTING TO TURN ON AN OUTPUT AT A DEFINED TIME

1. Introduction

Senquip devices can run custom scripts, allowing extra flexibility for the user. The scripting language is called mJS and is a restricted JavaScript variant. In this Application Note, a script will be written to turn on an output for a defined period each hour. The on period will be defined by a start and end minute which will be configurable by a user from the Senquip Portal.

The application note will focus on the script that turns the light on over defined number of minutes each hour. Instead of controlling a light, the script could however be a lot more complex, sending a series of MODBUS or CAN commands to control an attached machine, for instance.

It is assumed that the user has scripting rights. To request scripting rights, contact support@senquip.com.

2. References

The following documents were used in compiling this Application Note.

Reference	Document	Document Number
A	Senquip Scripting Guide	Device Scripting Guide 4.1.0

3. Overview

An LED light is wired to Output 1 on a Senquip ORB-C1-G as shown in Figure 1. The LED will turn on when Output 1 turns on and will remain off when Output 1 is off.

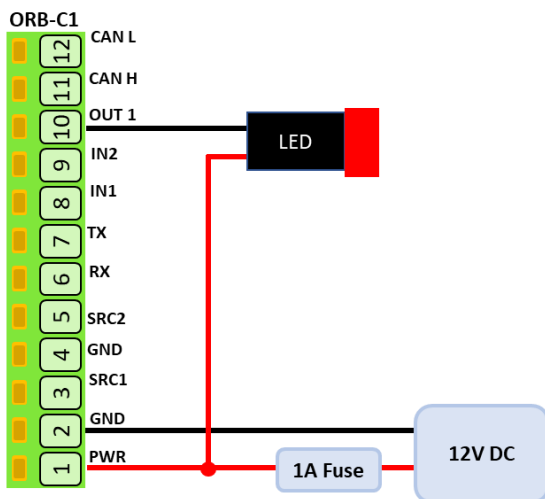
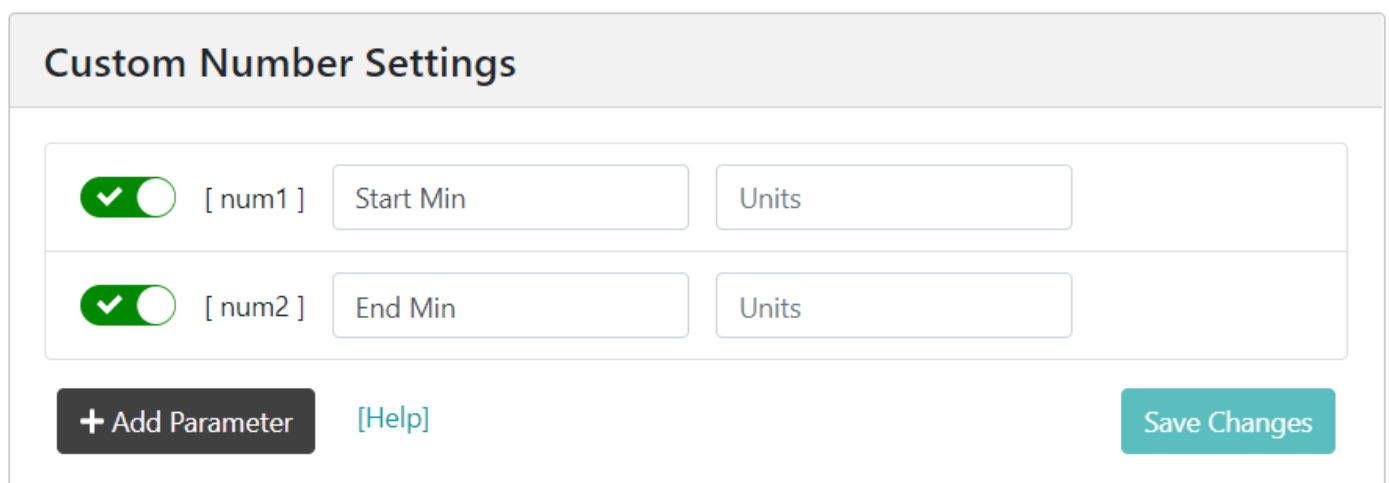


Figure 1 - LED wired to Output 1

The LED will be commanded by a script to turn on and off at a specific minute in each hour. The start and stop minute will be stored using the Senquip Custom Settings feature. Custom settings can be created on the device Scripting page, and once created can be used in a script. These settings are permanently stored in the device's FLASH memory like all other configuration values. Settings can either be a number or string type. The visibility, name and units for each custom setting can be modified from the Scripting page. In this example, two custom number settings, *Start Min* and *End Min*, will be used, as shown in Figure 2.



The screenshot shows a web interface titled "Custom Number Settings". It contains two rows of settings. Each row has a green toggle switch with a checkmark, a label in brackets (e.g., [num1]), a text input field for the name (e.g., "Start Min"), and another text input field for units (e.g., "Units"). At the bottom left, there is a dark button labeled "+ Add Parameter" and a link labeled "[Help]". At the bottom right, there is a teal button labeled "Save Changes".

Figure 2 - Start and End Minute Custom Settings

Once settings have been configured and enabled from the Scripting page, they will show up under the device's Setting page on the Custom tab and their value can be changed by a user who has Admin or User rights for that device. The custom settings for this application are shown in Figure 3.

The following libraries are required for the script:

- **api_timer.js**: used to access the current time from the device
- **api_config.js**: required to read the custom settings from the device

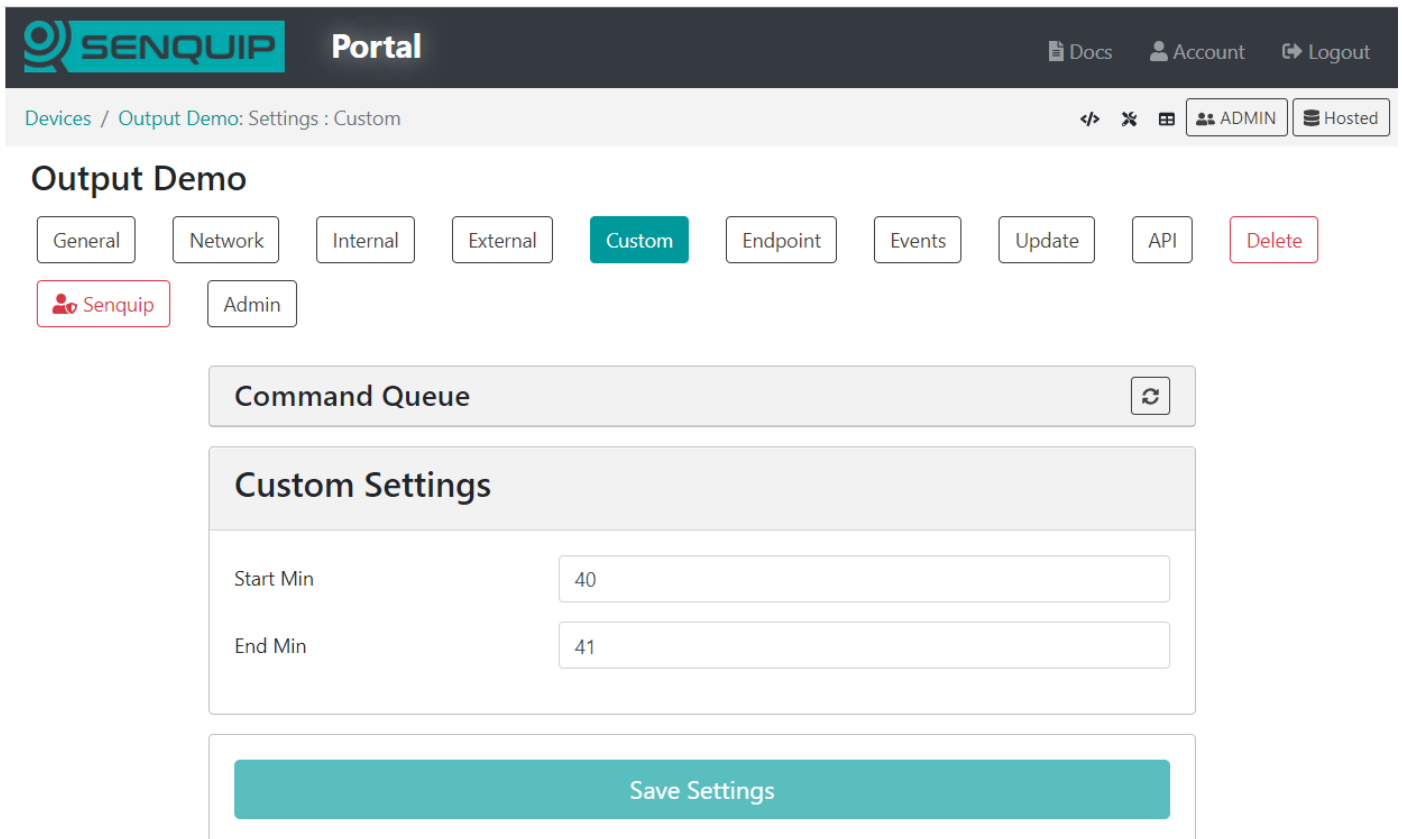


Figure 3 - Custom Settings

Custom parameters will be used to provide visibility of the status of the LED on the Senquip Portal and to provide debug information during script development. Custom data parameters are configured on the Scripting page where their visibility, name and units can be specified. A parameter’s value can be sent in a data message as either a number or a string.

The following custom parameters are created in this example:

- **Time String:** Used for debug purposes to ensure that time is being correctly received
- **Minutes:** Used for debug purposes to ensure the time string is correctly parsed
- **State:** The state of Output 1 – either On or Off

Custom Data Parameters

<input checked="" type="checkbox"/>	[cp1]	Time String	Units
<input checked="" type="checkbox"/>	[cp2]	Minutes	Units
<input checked="" type="checkbox"/>	[cp3]	State	Units

Figure 4 - Custom Data Parameters

4. Writing a Script to Control the Output

This section describes the example script given in Appendix 1 to control Output 1 in response to time.

The script is created in the data handler function which is called every time a measurement cycle finishes. In this example, the device base interval has been set at 5 seconds and so the handler will be called every 5 seconds. The output turn-on could therefore be delayed by 5 up to seconds. If faster response is required, the script could have been called in response to a timed function.

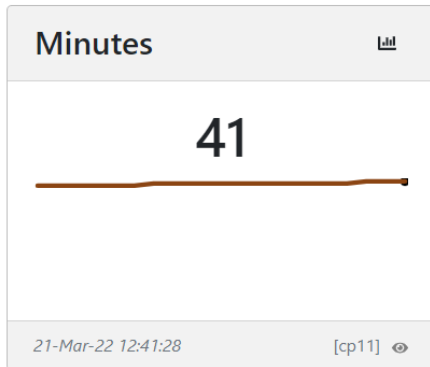
The script starts by retrieving the start and end minutes from the custom settings and loading them into variables *start_min* and *end_min*. The current time is then requested in JSON format. The format of the time returned can be modified in accordance with the [strftime format specification](#). The time string is dispatched to the Senquip Portal for debug purposes.

Time String

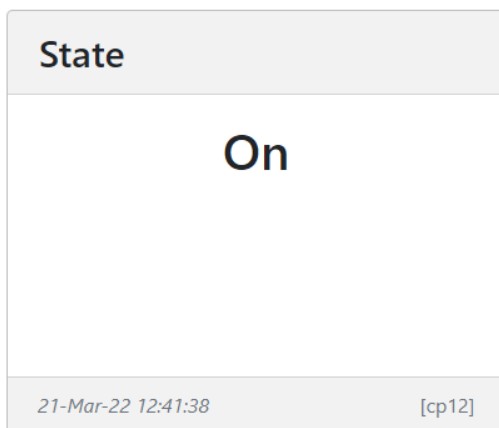
```
{hour:01,minute:40,second:58}
```

21-Mar-22 12:40:58 [cp10]

The minutes are then extracted from the time string and are dispatched to the Senquip Portal, again for debug purposes.



Finally, the current minutes are compared with the start and end minutes and if the current minutes are more than or equal to the start minute and less than or equal to the end minute, the output is turned on. The current status of Output 1 is dispatched to the Senquip Portal.



The function to turn the output on has a parameter that allows the on-time to be specified in seconds. Instead of turning the output off in the else statement, the output could have been set to stay on for the difference between the end and start minutes.

The script as written could be improved by checking that time is correctly returned and handling it if it isn't, and by accounting for overflows if the end time is set as lower than the start time.

5. Conclusion

Controlling an externally connected device based on time can be performed using a simple script that runs on a Senquip device. The use of custom settings allows a user to customise the operation of a script without having to modify the script.

Appendix 1: Source Code

```
load('senquip.js');
load('api_timer.js');
load('api_config.js');

// This function gets called every time a measurement cycle finishes
SQ.set_data_handler(function(data) {
  let obj = JSON.parse(data);

  let start_min = Cfg.get('script.num1');
  let end_min = Cfg.get('script.num2');
  let utc_str = Timer.fmt("{hour:%H,minute:%M,second:%S}", Timer.now());
  SQ.dispatch(1,utc_str); // sanity check
  let utc = JSON.parse(utc_str);
  SQ.dispatch(2,utc.minute); // sanity check

  if ((utc.minute >= start_min) && (utc.minute <= end_min)){
    SQ.set_output(1, SQ.ON, 0);
    SQ.dispatch(3,"On");
  }
  else{
    SQ.set_output(1, SQ.OFF, 0);
    SQ.dispatch(3,"Off");
  }
}, null);
```