MAT-201

Hands-on with the Arduino Nano Matter

Martin Looker
Mass Market Training Manager
SILICON LABS

The Leader in IoT Wireless Connectivity

#1
Share in Mesh

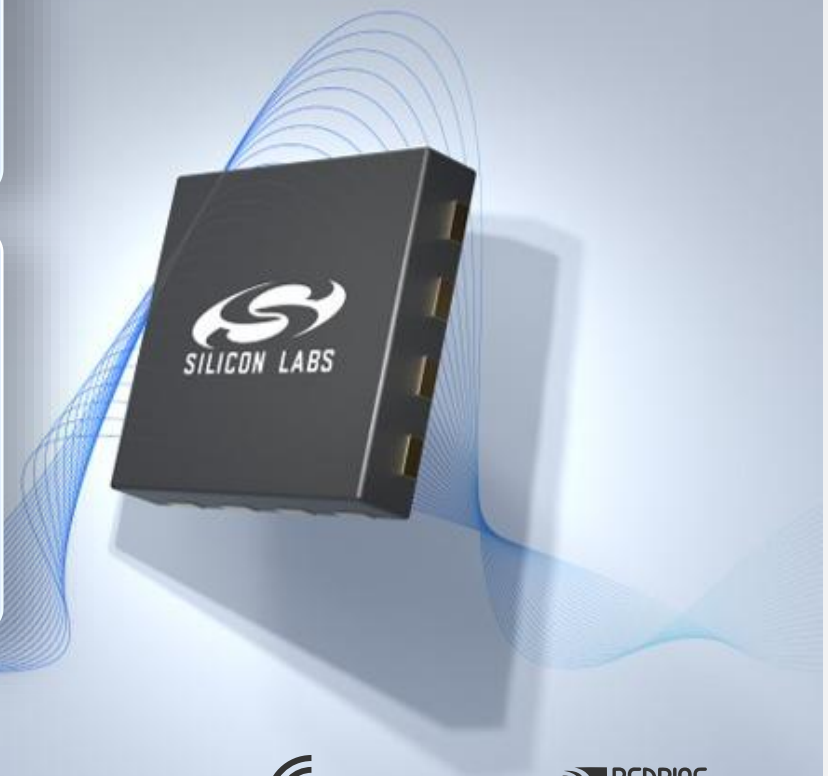
1st
To Market with Multiprotocol, Bluetooth LE, Bluetooth Mesh

Innovation
Performance, Power, CoEx, Modules, Secure Vault™

100%
IoT Focused

amazon sidewalk · Bluetooth · matter · Multiprotocol · Proprietary · THREAD · Wi Fi · Wi SUN · zigbee · Z-WAVE

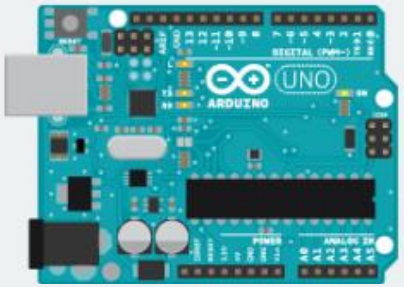
Breadth and Depth of Wireless IoT Protocols



ember	ENERGY micro	blue giga	telegesis	Micriμm	ZENTRI	Z-WAVE	REDPINE SIGNALS
2012 Software ZigBee SoC	**2013** Low-power 32-bit MCUs	**2015** BT Smart Modules	**2015** ZigBee/Thread Modules	**2016** Software RTOS	**2017** Cloud Connected Wi-Fi	**2018** Smart Home Protocol	**2020** Ultra Low Power Wi-Fi

About ARDUINO

WHAT IS ARDUINO?



Arduino is an open-source electronics platform based on easy-to-use hardware and software. It's intended for anyone making interactive projects.

Learn more about Arduino

ARDUINO BOARD

Arduino senses the environment by receiving inputs from many sensors, and affects its surroundings by controlling lights, motors, and other actuators.

Discover the official Arduino boards

ARDUINO SOFTWARE

You can tell your Arduino what to do by writing code in the Arduino programming language and using the Arduino development environment.

Download the Arduino Software

AGENDA

In this workshop:

01 How to add Matter support to the Arduino IDE

02 How to create and adapt example Matter applications

03 How to commission Matter Accessory Devices (MADs) to a network

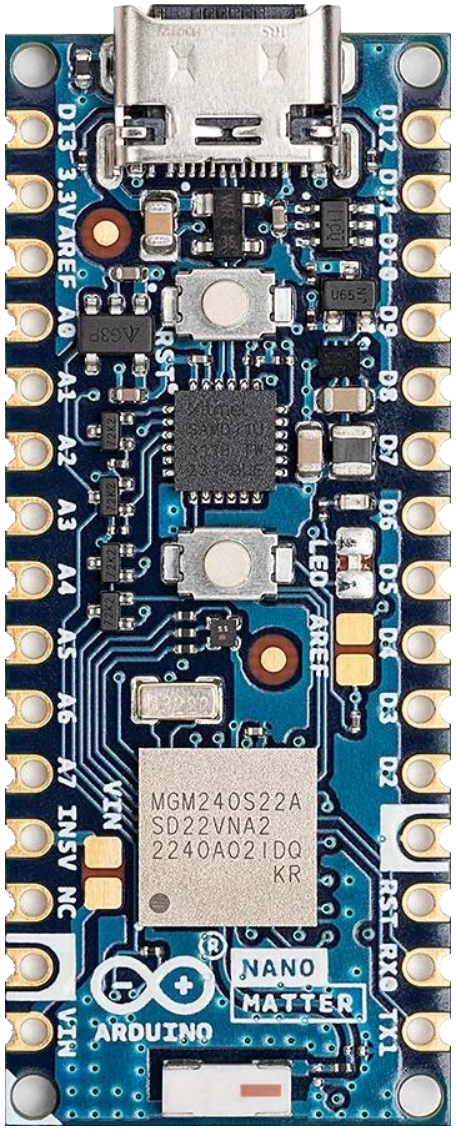
04 How to control and automate Matter Accessory Devices

04 The theory of key Matter features

Pre-requisites - Hardware

Hardware (Presenter)

- Amazon Alexa Echo (4th Gen)
  - With OpenThread Border Router (OTBR)



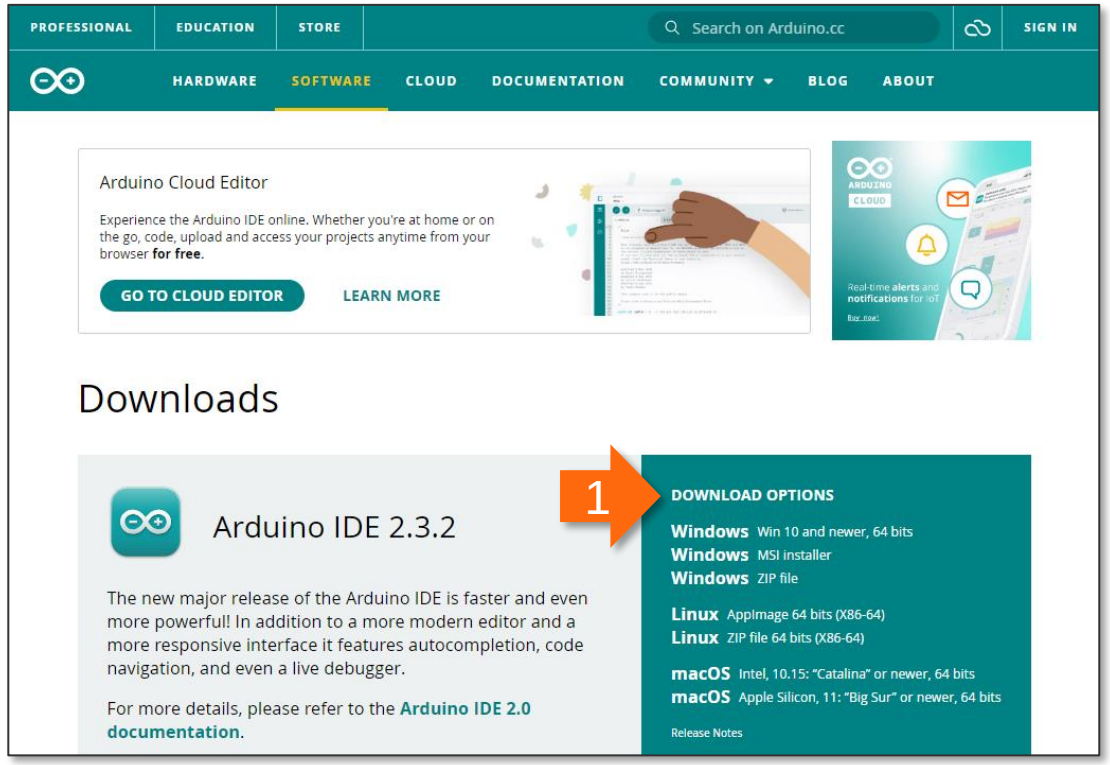
Hardware (Attendees)

- Arduino Nano Matter
  - With PCB for SR602 PIR Sensor (optional)

Pre-requisites - Software

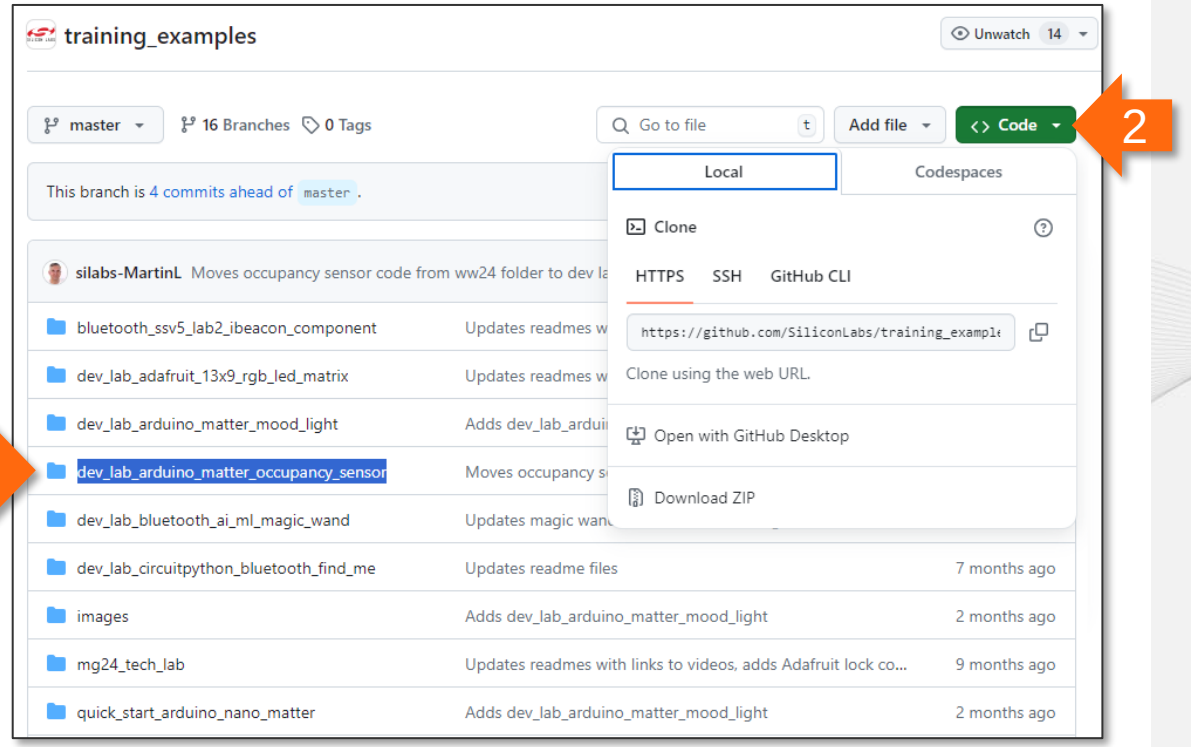
Software

1. Arduino IDE installed and ready to use: https://www.arduino.cc/en/software



Code

2. Clone or download code and presentation from: https://github.com/SiliconLabs/training_examples
3. Files for this workshop are in the folder: dev_lab_arduino_matter_occupancy_sensor



Why Matter?

Current Landscape of IoT Industry

Consumers

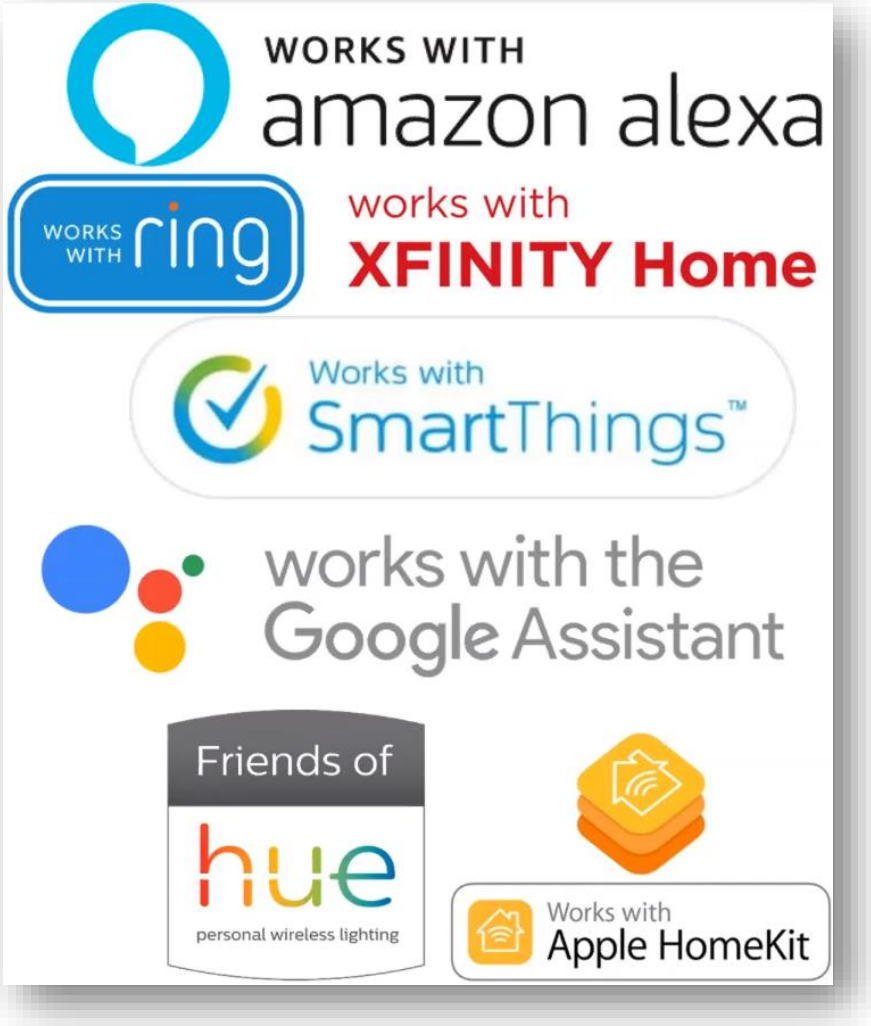
- Extremely hard to mix and match the product they want with their preferred ecosystem
- Very difficult to change once selected

Developers

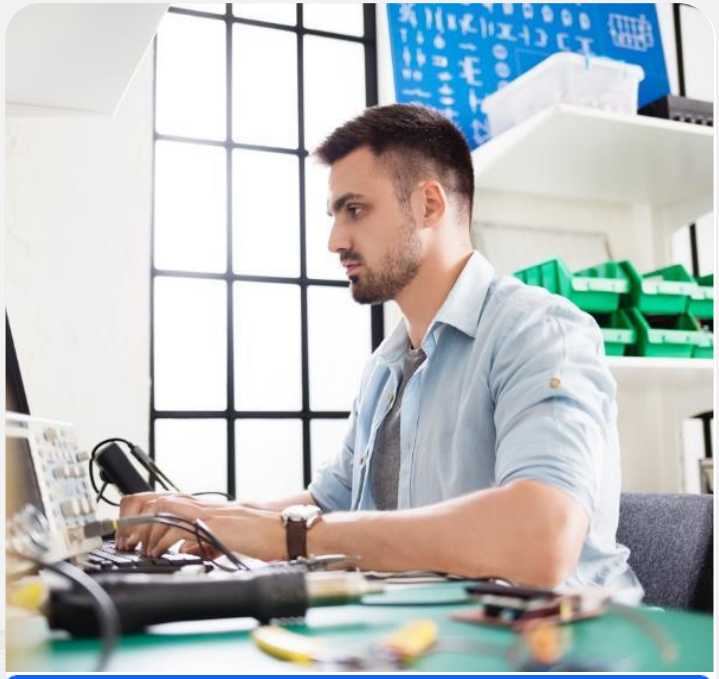
- Developers are forced to pick what ecosystem integrations they support
- Often need to ship multiple SKUs for all connectivity standards
- Need to learn different IoT technologies and ecosystems

Retailers

- Too difficult to provide expert advice to answer consumer questions
- High return rates due to interoperability or incompatibility issues

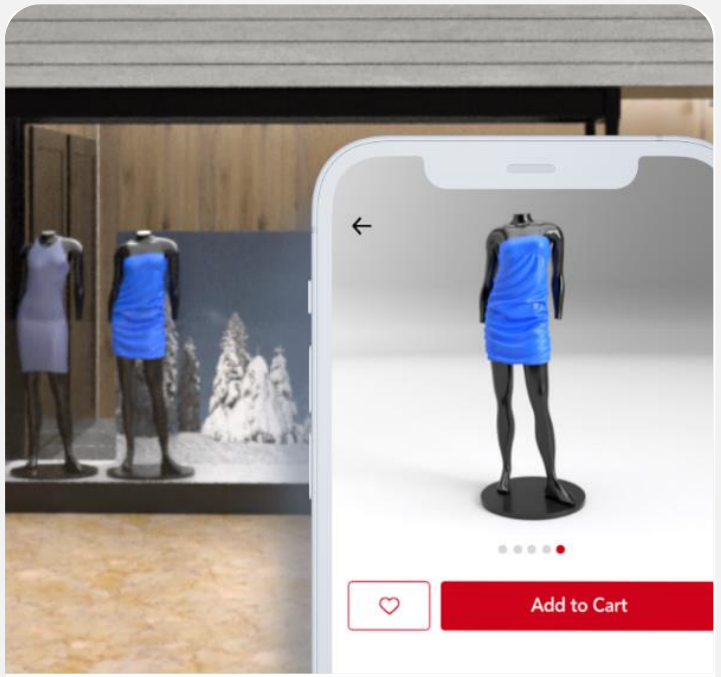


Simplifying IoT Connectivity



DEVELOPER/MANUFACTURER

- **Single SKU**
- **Lower development & operational cost**
- **More time for innovation**



RETAILER

- **Requires less shelf space**
- **Lowers inventory cost**
- **Minimizes returns**

CONSUMER

- **Simplifies purchasing experience**
- **Simplifies setup & control**
- **Provides better user experience**

Arduino IDE and Board Setup

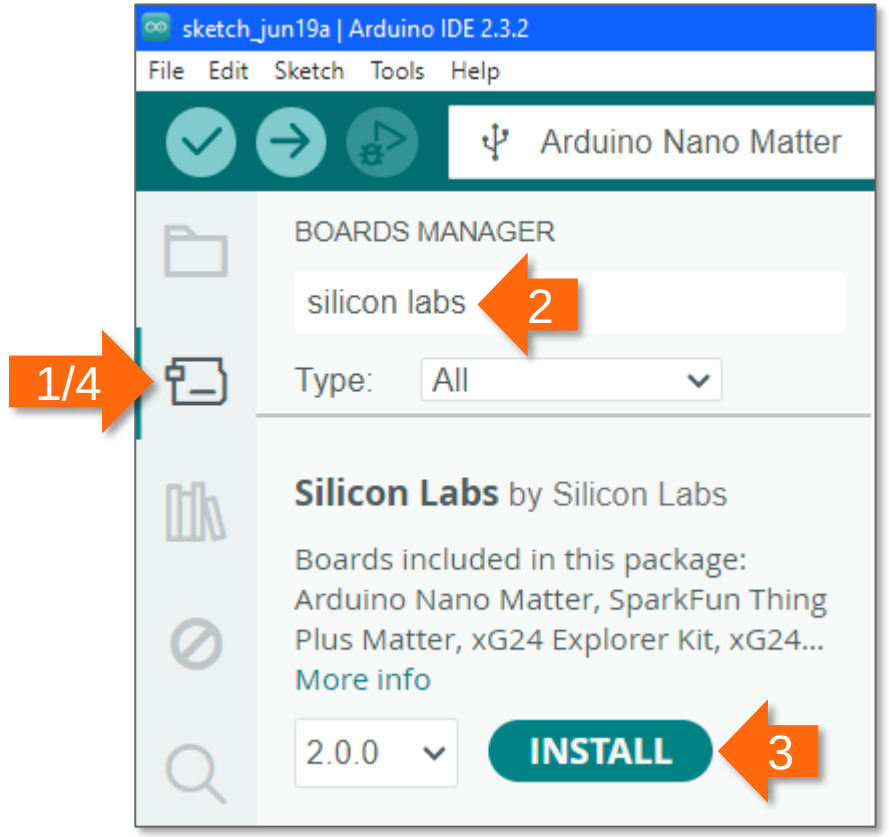
Boards Manager

Silicon Labs Arduino Core installation:

1. Open the Boards Manager by clicking the icon
2. Enter Silicon Labs in the search box
3. Click the Install button
4. Close the Boards Manager by clicking the icon again

Link:

- https://github.com/SiliconLabs/arduino



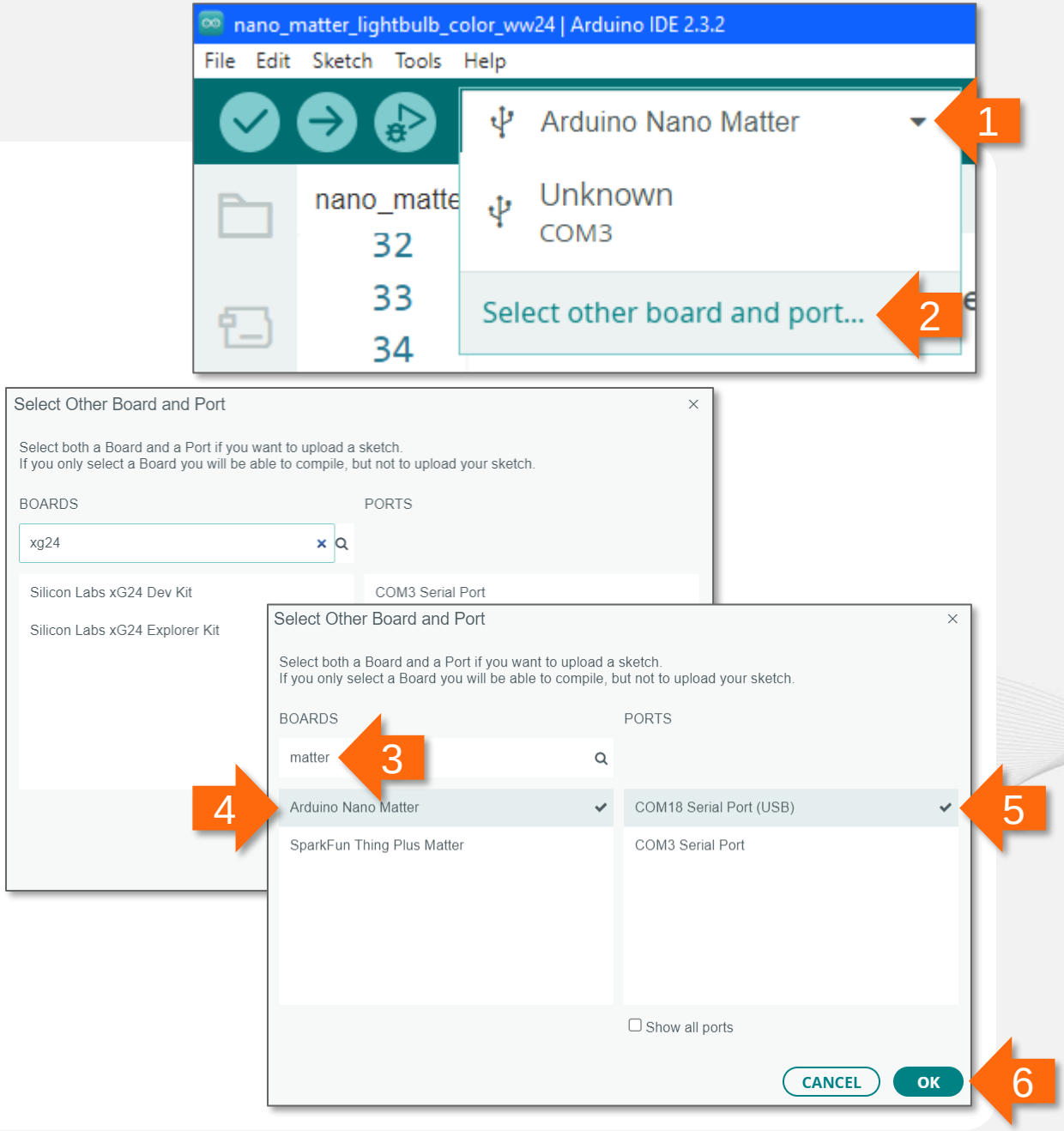
Board Configuration

Board configuration:

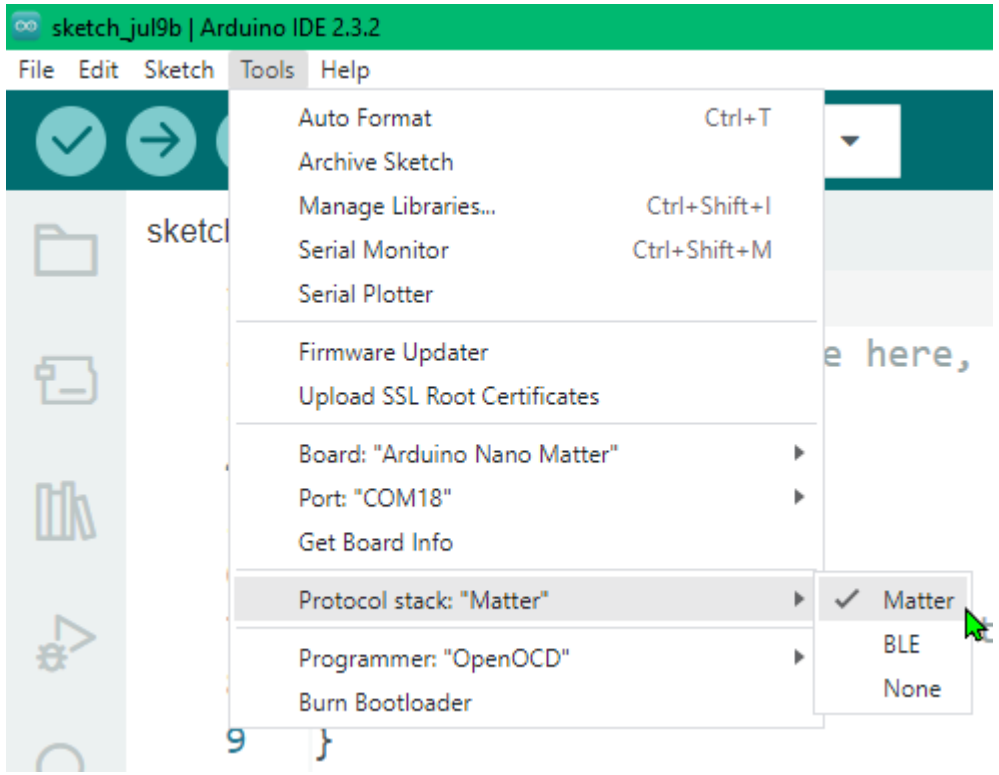
1. Click the Board dropdown
2. Click Select other board and port
3. Search for matter
4. Select correct board
5. Connect board via USB and select COM port
6. Click OK button

Link:

- https://docs.arduino.cc/tutorials/nano-matter/user-manual



Protocol Stack



Three different protocol stacks can be selected from the Tools > Protocol stack menu:

1. Matter for developing Matter over Thread applications
2. BLE for developing Bluetooth LE applications
3. None for developing applications without radio comms

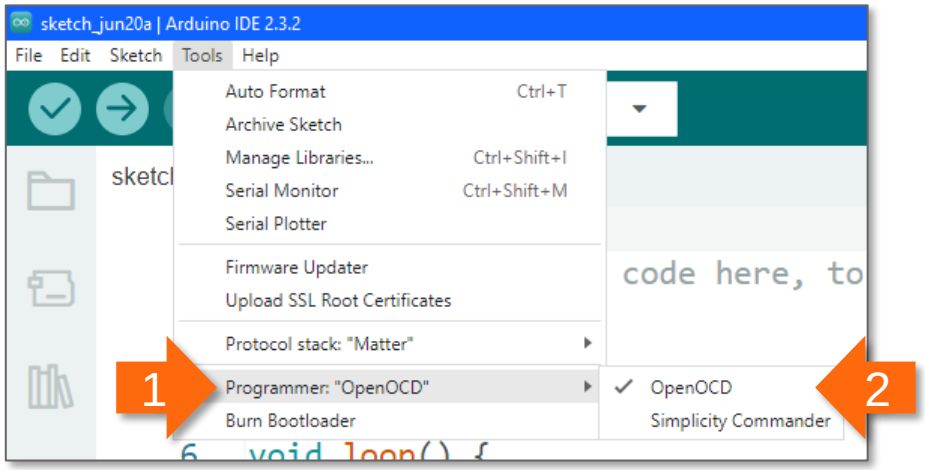
Ensure the Matter option is selected for this workshop

1

Programmer and Bootloader

Different boards use different flash programmers:

1. From the menu click Tools > Programmer
2. For the Arduino Nano Matter board, select Open OCD
3. For other boards, select Simplicity Commander

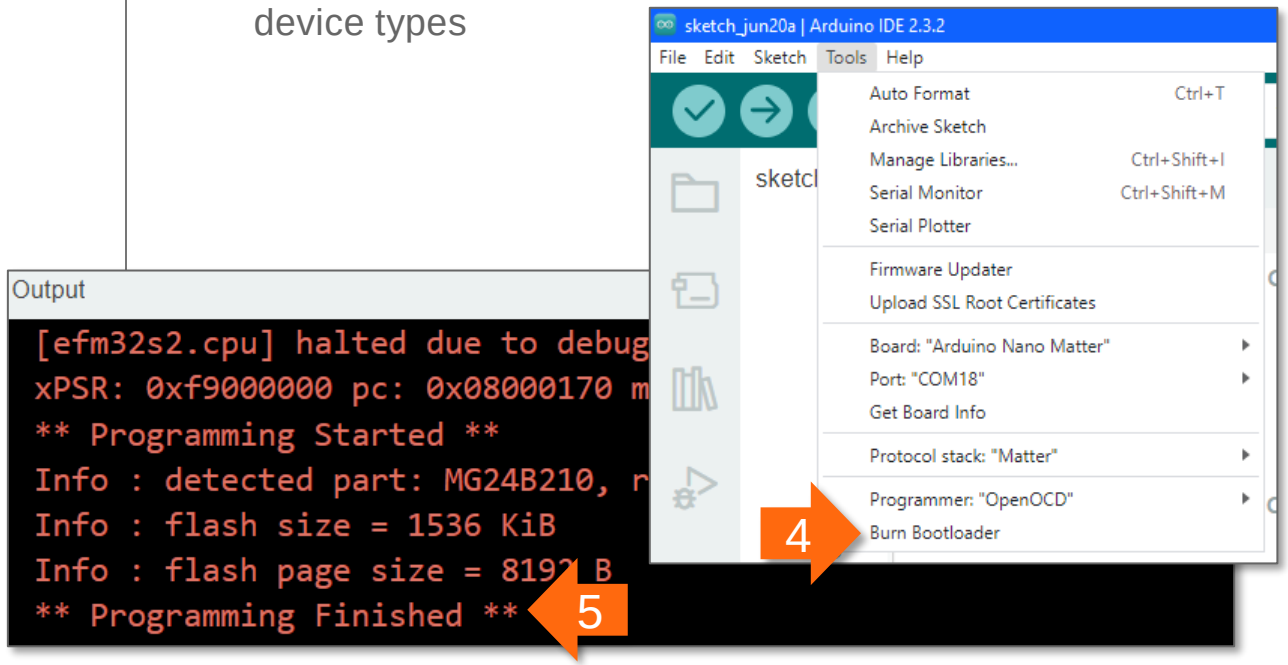


Applications using the Silicon Labs Arduino Core require a bootloader to be programmed:

4. From the menu click Tools > Burn Bootloader
5. The output window will display progress

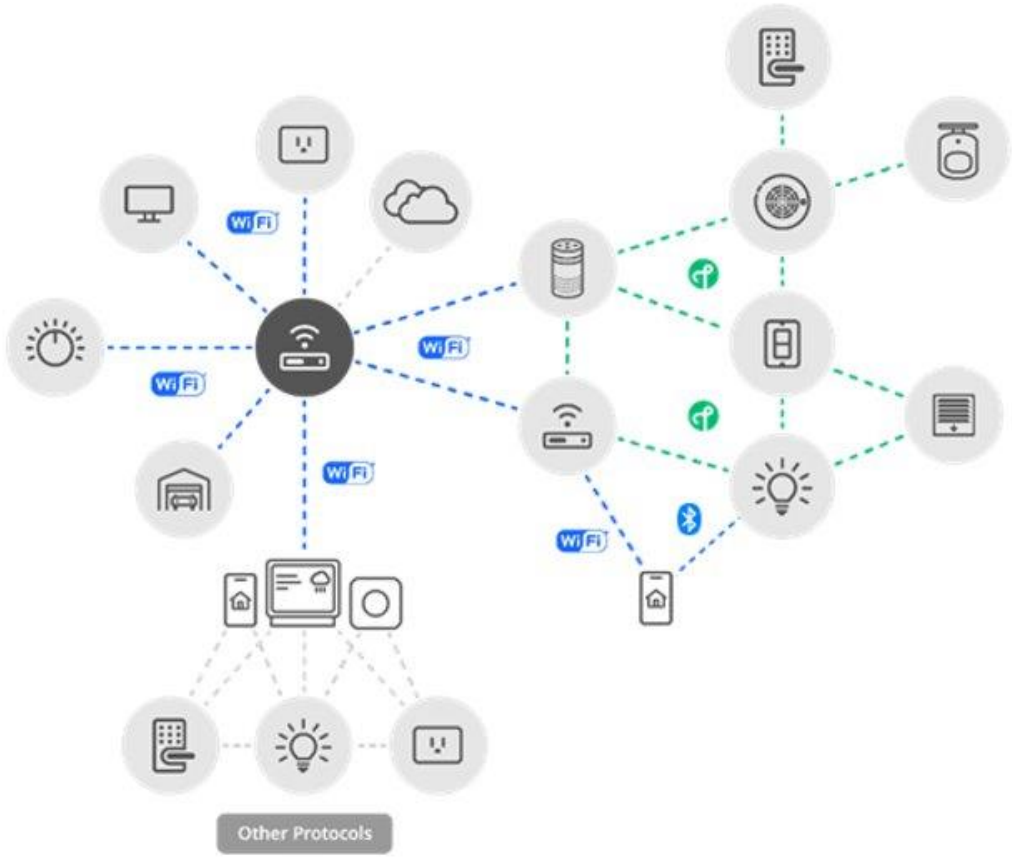
Burning the bootloader also erases the whole flash, this includes Matter credentials and application

- Recommend re-burning the bootloader if changing Matter device types



About Matter

Network Topology



Matter wireless protocols:

- Matter has native support for Thread and Wi-Fi
- Matter uses Bluetooth for commissioning
- Other protocols, like Zigbee and Z-Wave, can be bridged into a Matter network

Matter device types:

- Matter Accessory Devices (MADs) provide end-node functionality such as lights and switches
- OpenThread Border Routers (OTBRs) provide a connection between Thread devices and the local IP network
  - The Raspberry Pi and OpenThread Radio Co-Processor together form an OTBR
- Matter Controllers are used to control MADs, including commissioning them into the network
  - Typically, these are smart phones
  - The Raspberry Pi image includes a command-line Matter Controller
- Bridges can be used to link to other protocols, like ZigBee and Z-Wave

Matter Fabric

Multiple transports

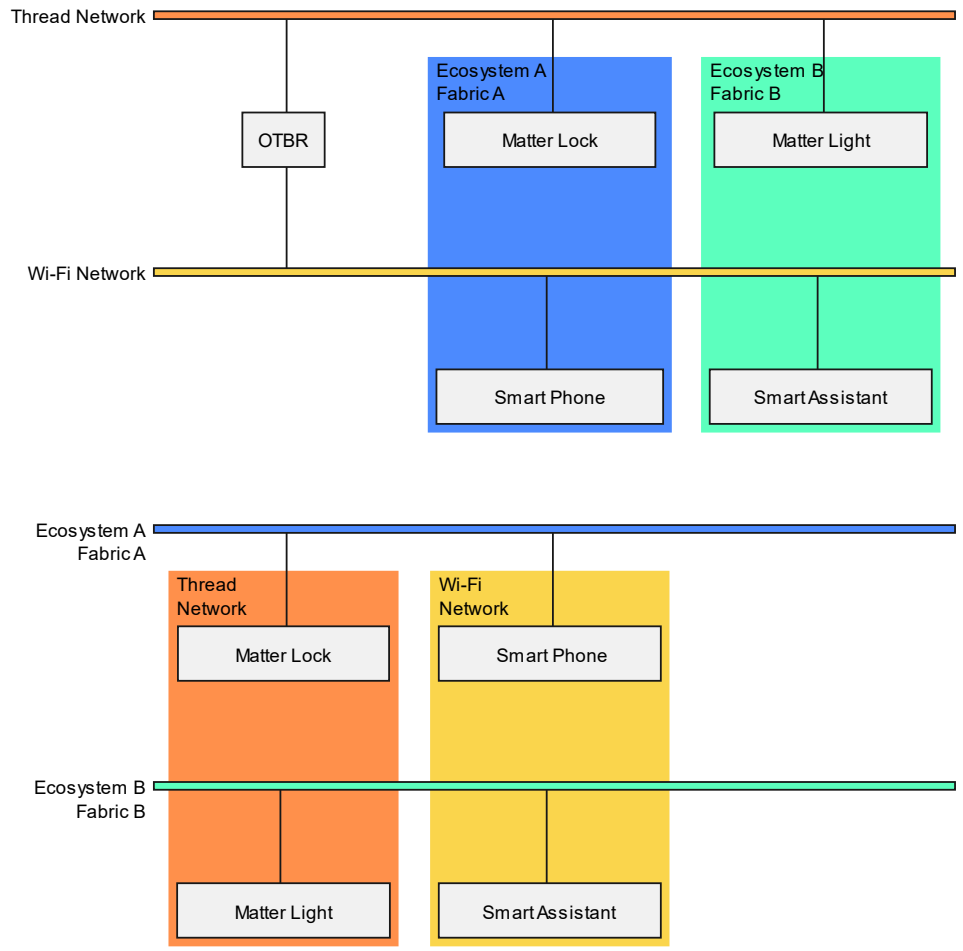
- Matter can work on top of multiple wireless or wired technologies to transport the IP packets

Fabric

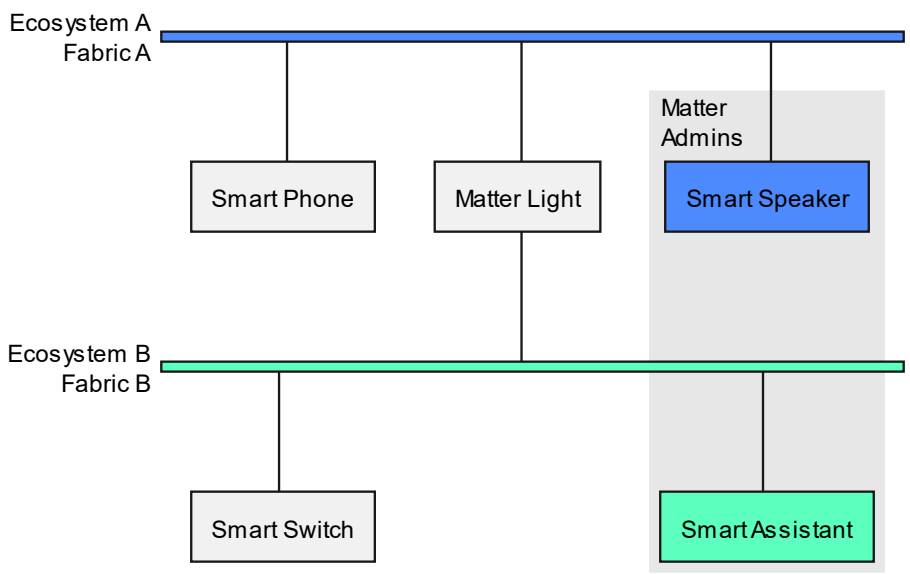
- A collection of Matter devices sharing a trusted root
- A fabric is identified by a fabric ID which is a 64-bit number

Node

- In a Matter fabric, each physical device is called a node
- Each node is identified by a node ID which is a 64-bit number



Matter Multi-Admin



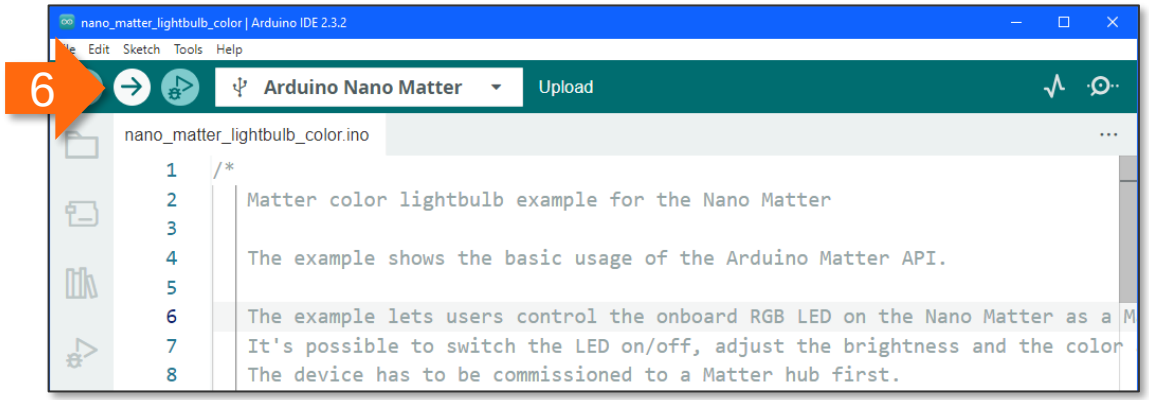
- Provides a means for multiple Matter Fabrics and their administrators to manage devices
- Each Matter Fabric can have unique root authority
- Devices must support multiple Matter Admins
- Matter admins dictate the access control lists for their Matter fabric, and thus the devices can access the device
- Example:
  - Smart Speaker, Admin for Fabric A, can grant control privileges to Smart Phone on Fabric A
  - Smart Assistant, Admin for Fabric B, can grant control privileges to Smart Switch on Fabric B
- Access Control is separate for both fabrics

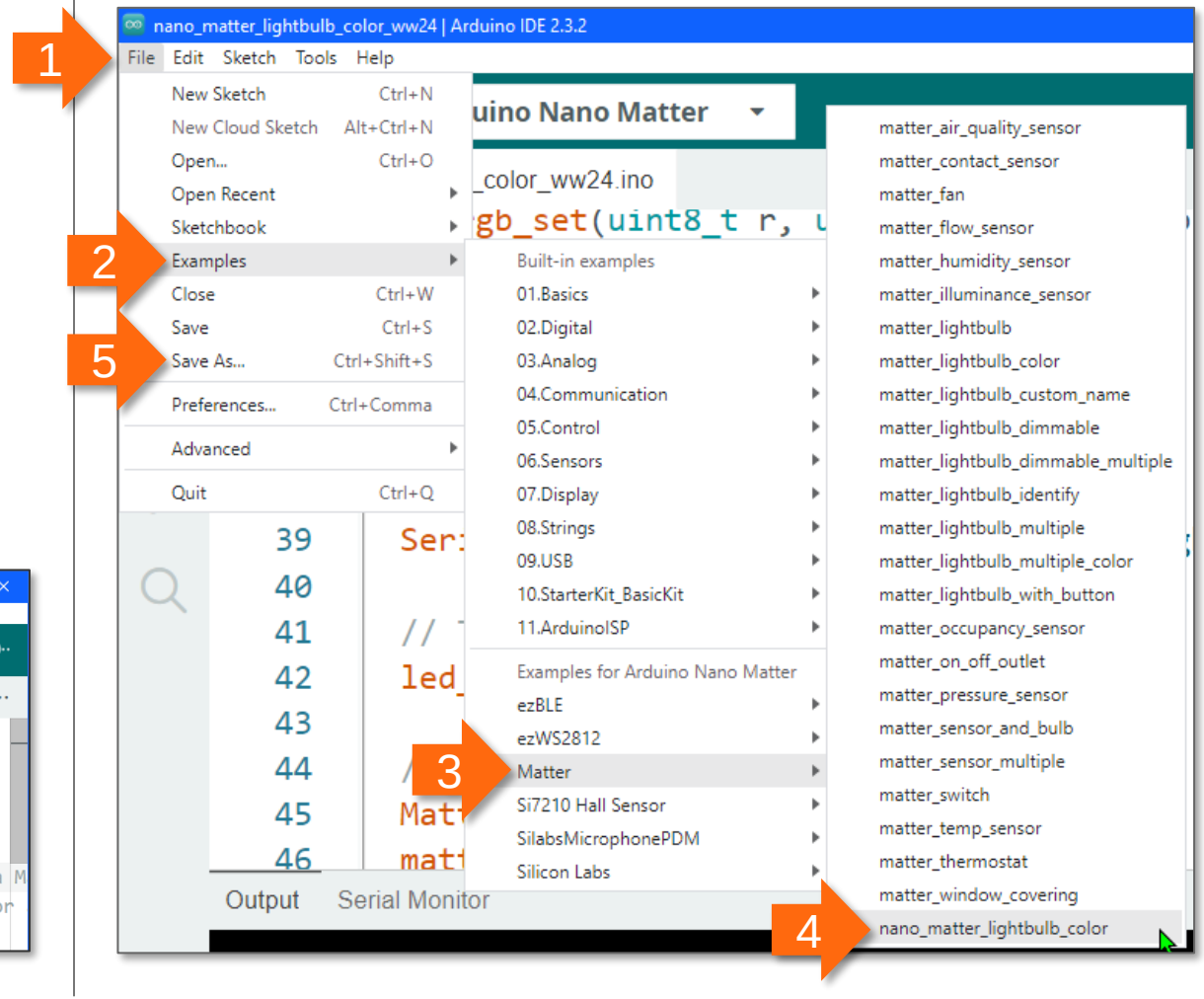
Nano Matter Lightbulb Color

Nano Matter Lightbulb Color Example

To create the Nano Matter Lightbulb Color example:

1. From the File menu select:
2. Examples
3. Matter
4. nano_matter_lightbulb_color the example sketch will be opened in a new window
5. From the File menu select Save As and save the sketch to your local filesystem
6. Click the Upload button to compile and transfer the application to the board





Color Lightbulb: Code Review

Color Lightbulb: Code

The lightbulb code works as follows:

- Matter.h – provides access to Matter APIs common to all devices
- MatterLightbulb.h – provides access to lightbulb device types
  - includes, on/off, dimmable and color
- LED_*x* – these defines are for the red, green and blue pins
  - This example uses the built-in LED pins, other pins could be used for an external RGB LED
- matter_color_bulb – this object encapsulates the color bulb Matter functionality
- Function prototypes – covered later
- button_pressed – used to flag when the button is pressed

```
/*
  Matter color lightbulb example for the Nano Matter

  The example shows the basic usage of the Arduino Matter API.

  The example lets users control the onboard RGB LED on the Nano Matter as a
  Matter lightbulb. It's possible to switch the LED on/off, adjust the
  brightness and the color as well. The device has to be commissioned to a
  Matter hub first.

  Compatible boards:
  - Arduino Nano Matter
  - xG24 Dev Kit

  Author: Tamas Jozsi (Silicon Labs)
 */
#include <Matter.h>
#include <MatterLightbulb.h>

#define LED_R LED_BUILTIN
#define LED_G LED_BUILTIN_1
#define LED_B LED_BUILTIN_2

MatterColorLightbulb matter_color_bulb;

void update_led_color();
void led_off();
void handle_button_press();
volatile bool button_pressed = false;
```

Color Lightbulb: setup() - Initialisation

The setup() function is called once at startup, initialization takes place here:

- First the serial port is initialized for outputting debug messages
- The common Matter interface is started
- The Matter Color Bulb object is started
- The boost_saturation() call is made
  - When setting colors on the Google Home mobile app's color wheel selecting at the edge of the wheel does not fully saturate, this setting ensures that such selections are fully saturated
- The on-board button is initialized, and an interrupt callback function set to handle a falling state
- The RGB LED is turned off
- A welcome debug message is output to the serial port

```
30  void setup()
31  {
32    Serial.begin(115200);
33    Matter.begin();
34    matter_color_bulb.begin();
35    matter_color_bulb.boost_saturation(51); // Boost saturation by 20 percent
36
37    // Set up the onboard button
38    pinMode(BTN_BUILTIN, INPUT_PULLUP);
39    attachInterrupt(BTN_BUILTIN, &handle_button_press, FALLING);
40
41    // Turn the LED off
42    led_off();
43
44    Serial.println("Arduino Nano Matter - color lightbulb");
```

Color Lightbulb: setup() - Commissioning

Processing remains in the setup() function until the device is in a network, there are three stages:

- isDeviceCommissioned() checks whether the device has been commissioned into a network
  - When not in a network, commissioning data is output to the serial port
  - A device only needs to be commissioned once
  - A loop waits for commissioning to be completed
- isDeviceThreadConnected() checks whether the device is running in a Thread network
  - The device will need to rejoin the Thread network each time it is started
  - The device may form an isolated Thread network until it can rejoin the original network
  - A loop waits for the device to be in a Thread network
- is_online() returns true when a controller device, such as a hub, queries the device
  - Confirms the device can be controlled by a hub
  - A loop waits for the device to be confirmed online

These checks could be removed, but they provide a clean startup

```
53    if (!Matter.isDeviceCommissioned()) {
54      Serial.println("Matter device is not commissioned");
55      Serial.println("Commission it to your Matter hub with the manual pairing
56      Serial.printf("Manual pairing code: %s\n", Matter.getManualPairingCode().
57      Serial.printf("QR code URL: %s\n", Matter.getOnboardingQRCodeUrl().c_str(
58    }
59    while (!Matter.isDeviceCommissioned()) {
60      delay(200);
61    }
62
63    Serial.println("Waiting for Thread network...");
64    while (!Matter.isDeviceThreadConnected()) {
65      delay(200);
66    }
67    Serial.println("Connected to Thread network");
68
69    Serial.println("Waiting for Matter device discovery...");
70    while (!matter_color_bulb.is_online()) {
71      delay(200);
72    }
73    Serial.println("Matter device is now online");
74  }
```

Color Lightbulb: loop() – Bulb On/Off Control

The loop() is called repeatedly while the application runs and contains the main application code

The first part is concerned with the on/off state of the bulb:

- Local control is available, toggling the bulb on/off state whenever the on-board button is pressed
- The last known logical state is held in a static variable
- The current logical state is read from the color bulb object
- When the logical state is changed to on, the physical state is updated with a call to update_led_color()
- When the logical state is changed to off, the physical state is updated with a call to led_off()

The logical state, as set over the air, is being polled for changes and the RGB LED is updated to match

```
76  void loop() {
77    // If the physical button state changes - update the lightbulb's on/off state
78    if (button_pressed) {
79      button_pressed = false;
80      // Toggle the on/off state of the lightbulb
81      matter_color_bulb.toggle();
82    }
83
84    // Get the current on/off state of the lightbulb
85    static bool matter_lightbulb_last_state = false;
86    bool matter_lightbulb_current_state = matter_color_bulb.get_onoff();
87
88    // If the current state is ON and the previous was OFF - turn on the LED
89    if (matter_lightbulb_current_state && !matter_lightbulb_last_state) {
90      matter_lightbulb_last_state = matter_lightbulb_current_state;
91      Serial.println("Bulb ON");
92      // Set the LEDs to the last received state
93      update_led_color();
94    }
95
96    // If the current state is OFF and the previous was ON - turn off the LED
97    if (!matter_lightbulb_current_state && matter_lightbulb_last_state) {
98      matter_lightbulb_last_state = matter_lightbulb_current_state;
99      Serial.println("Bulb OFF");
100     led_off();
101   }
```

Color Lightbulb: loop() – Bulb Color Control

Color control works in a similar way to on/off control:

- The last known logical values for hue, saturation and brightness are held in static variables
- The current logical values are read from the color bulb object
- When a logical value is, the physical state is updated with a call to update_led_color()

Once again, logical values are being polled for changes and applied to the physical RGB LED

```
103     static uint8_t hue_prev = 0;
104     static uint8_t saturation_prev = 0;
105     static uint8_t brightness_prev = 0;
106     uint8_t hue_curr = matter_color_bulb.get_hue();
107     uint8_t saturation_curr = matter_color_bulb.get_saturation_percent();
108     uint8_t brightness_curr = matter_color_bulb.get_brightness_percent();
109
110     // If either the hue, saturation or the brightness changes - update the LED
111     if (hue_prev != hue_curr ||
112         saturation_prev != saturation_curr ||
113         brightness_prev != brightness_curr) {
114       update_led_color();
115       hue_prev = hue_curr;
116       saturation_prev = saturation_curr;
117       brightness_prev = brightness_curr;
118     }
119   }
```

Color Lightbulb: update_led_color() and led_off() Functions

The update_led_color() function drives the LED when on:

- First a check ensures that the bulb is in the on state
- The logical red, green and blue values are read back from the Matter Color Bulb object
- The colors are applied to the LEDs using the analogWrite() function
- This takes into account whether an LED is on when the signal is low or high

The led_off() function drives the LED when off:

- The LEDs are turned off using the analogWrite() function

```
121  // Updates the color of the RGB LED to match the Matter lightbulb's color
122  void update_led_color() {
123    if (!matter_color_bulb.get_onoff()) {
124      return;
125    }
126    uint8_t r, g, b;
127    matter_color_bulb.get_rgb(&r, &g, &b);
128    // If our built-in LED is active LOW, we need to invert the brightness values
129    if (LED_BUILTIN_ACTIVE == LOW) {
130      analogWrite(LED_R, 255 - r);
131      analogWrite(LED_G, 255 - g);
132      analogWrite(LED_B, 255 - b);
133    } else {
134      analogWrite(LED_R, r);
135      analogWrite(LED_G, g);
136      analogWrite(LED_B, b);
137    }
138    Serial.printf("Setting bulb color to > r: %u  g: %u  b: %u\n", r, g, b);
139  }
140
141  // Turns the RGB LED off
142  void led_off() {
143    // If our built-in LED is active LOW, we need to invert the brightness values
144    if (LED_BUILTIN_ACTIVE == LOW) {
145      analogWrite(LED_R, 255);
146      analogWrite(LED_G, 255);
147      analogWrite(LED_B, 255);
148    } else {
149      analogWrite(LED_R, 0);
150      analogWrite(LED_G, 0);
151      analogWrite(LED_B, 0);
152    }
153  }
```

Color Lightbulb: handle_button_press() Function

The handle_button_press() function is called when a falling edge is detected on the button input:

- A check is made to ensure button presses within 200ms of each other are ignored
- The button_pressed variable is set to true (which is picked up in the loop() function)

The led_off() function drives the LED when off:

```
155  void handle_button_press() {
156    static uint32_t btn_last_press = 0;
157    if (millis() < btn_last_press + 200) {
158      return;
159    }
160    btn_last_press = millis();
161    button_pressed = true;
162  }
```

Color Lightbulb:
Commissioning and Operation

Color Lightbulb: Commissioning Codes

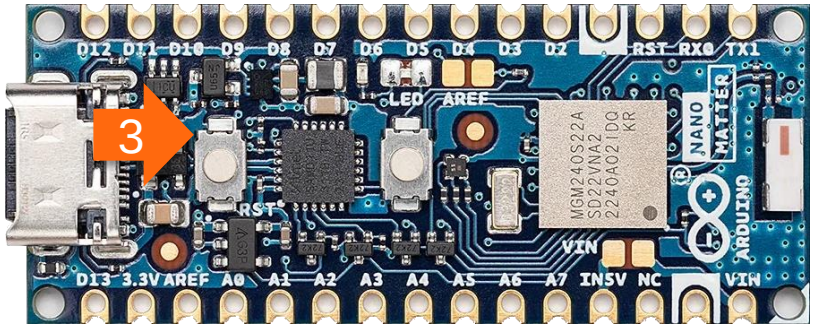
Commissioning takes place from an ecosystem mobile app

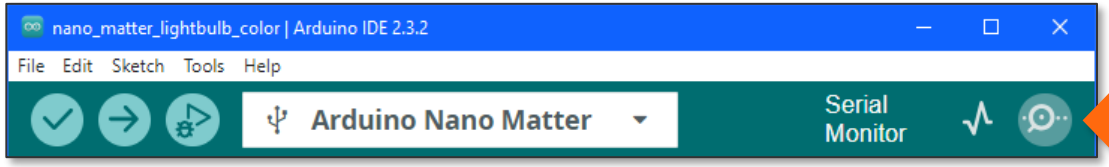
- We will be using Amazon Alexa in this workshop

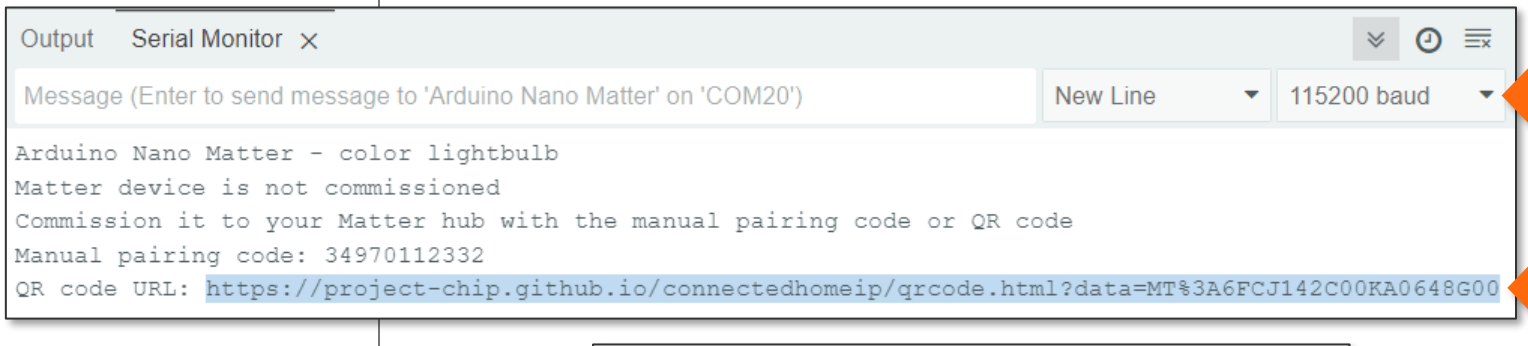
To obtain commissioning codes:

1. Open the Serial Monitor
2. Set baud rate to 115200
3. Reset board
4. Copy the QR Code URL from the debug output
5. Paste URL into a browser

The Manual Pairing Code can be typed in instead of scanning a QR Code





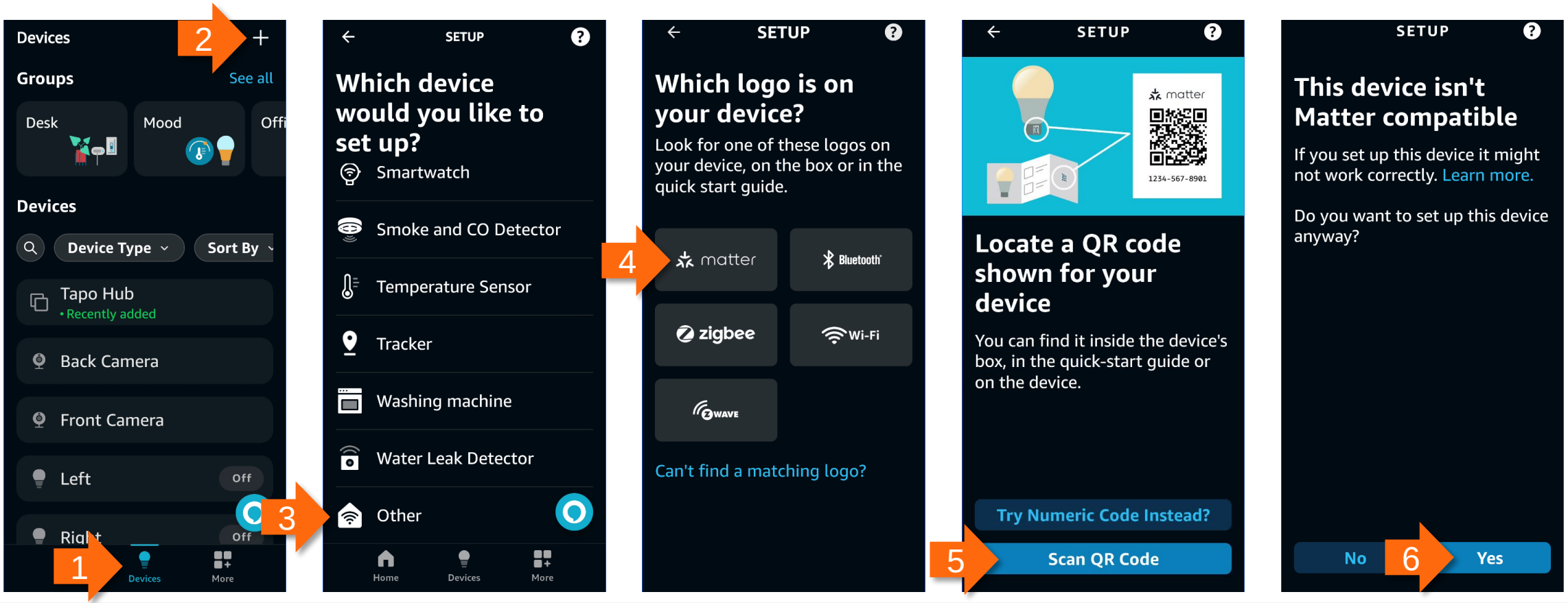


Commissioning Add

The commissioning process is similar in all the ecosystems

Test certificates are used by the example applications

6. There may be warnings of this during commissioning



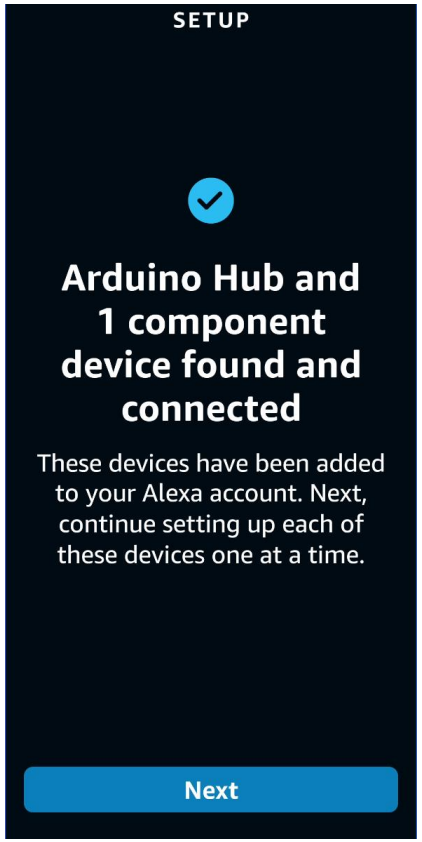
Color Lightbulb: Device Setup

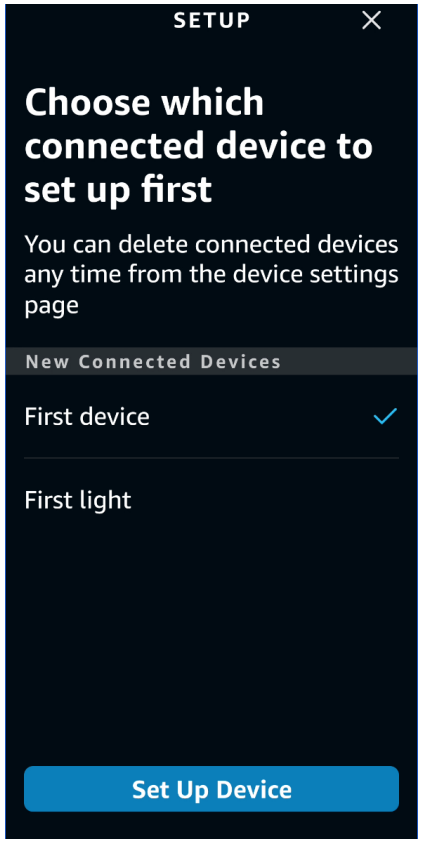
Select each device and click Set Up Device

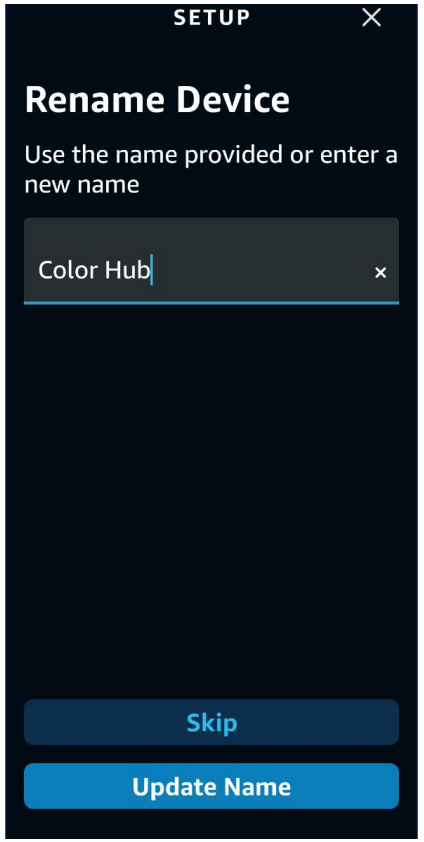
- Devices can be renamed
- The light device can be added to a group

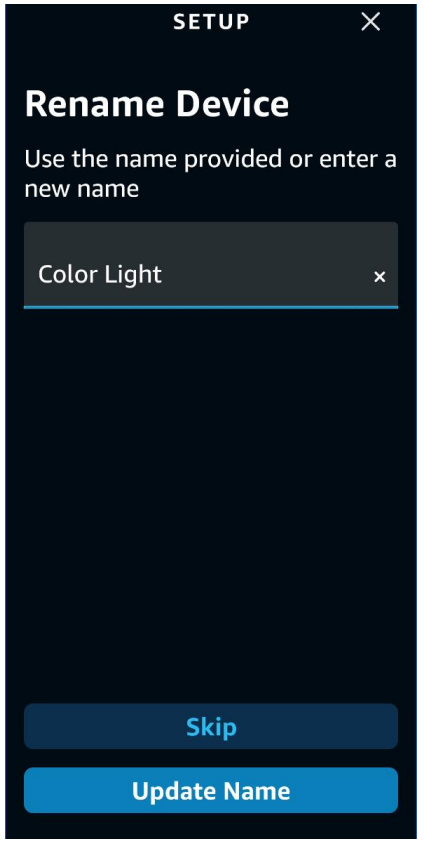
Two devices are found

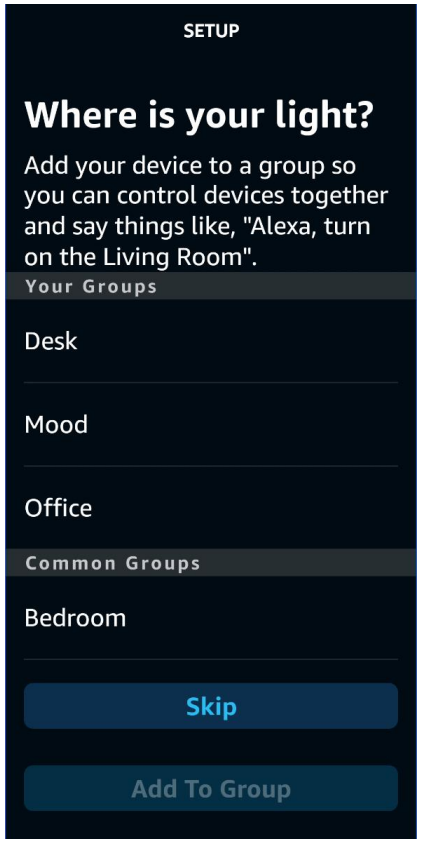
- The First light is the light endpoint
- The First device is a hub and is a side-effect of creating the light endpoint at runtime











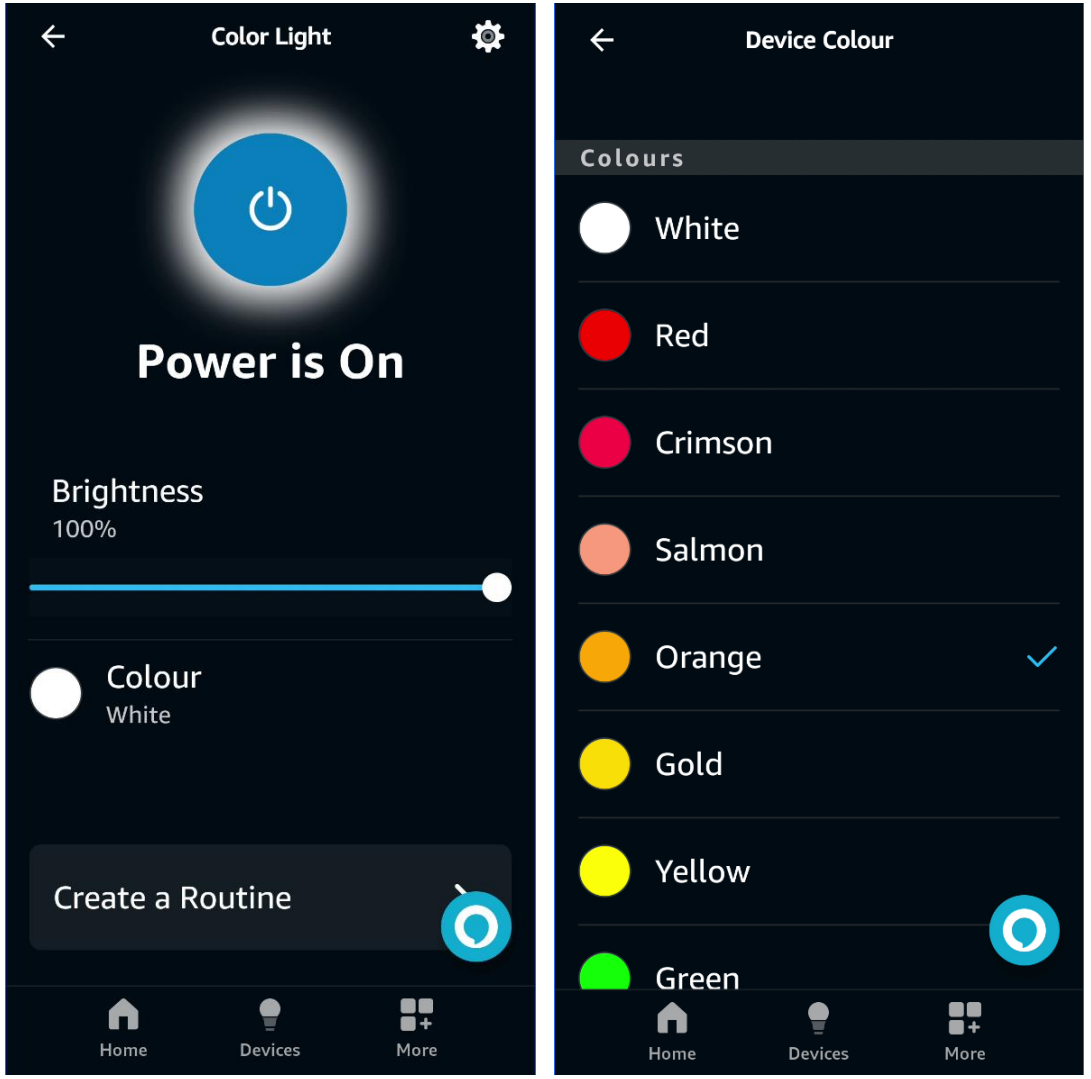
Color Lightbulb: Operation

The color light can be controlled using controls in the App including:

- Turning the light on and off
- Changing the brightness of the light
- Changing the color of the light
- Setting up routines, to turn a light on or off at particular times

The light can also be controlled using voice commands

- Using the App
- Using the Smart Assistant



Color Lightbulb: Theory

Zigbee Cluster Library

Matter leverages the Zigbee Cluster Library (ZCL)

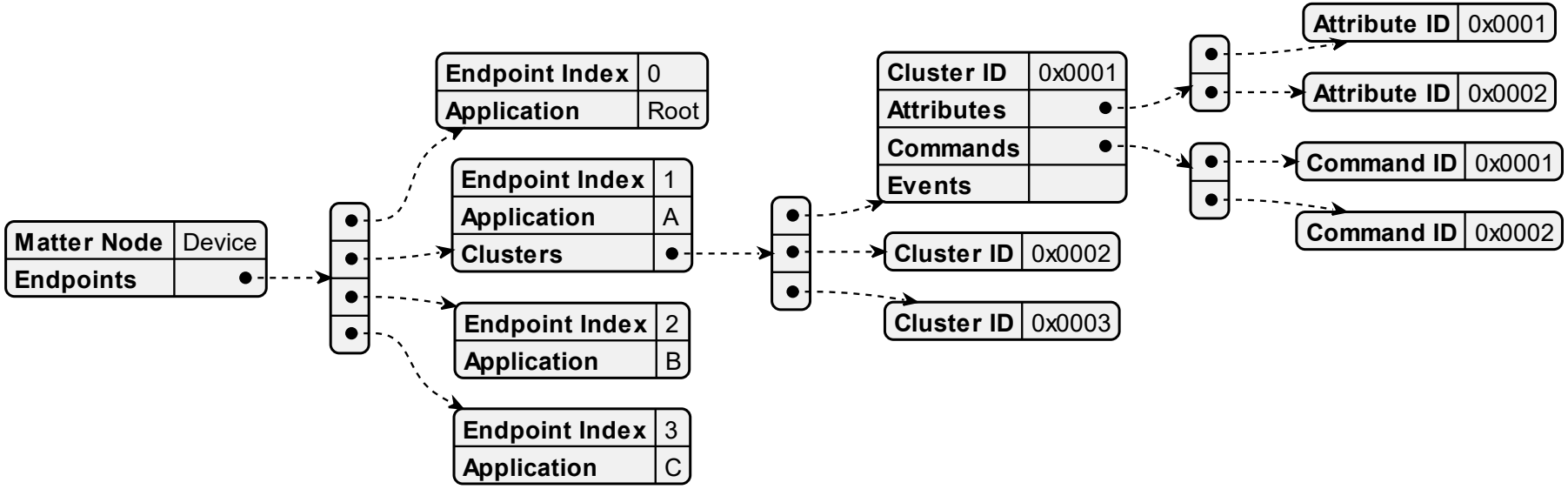
- Provides a set of field-proven methods to form and control devices
- Does not perfectly match the ZCL, has been extended with new functionality as needed

An endpoint is generally a single device type

- Interactions happen between local endpoints and remote endpoints in a client/server model

Specific functions are described by clusters. They contain:

- Attributes (state or modes)
- Commands (operations to be acted on)
- Events (transitions and alarms)



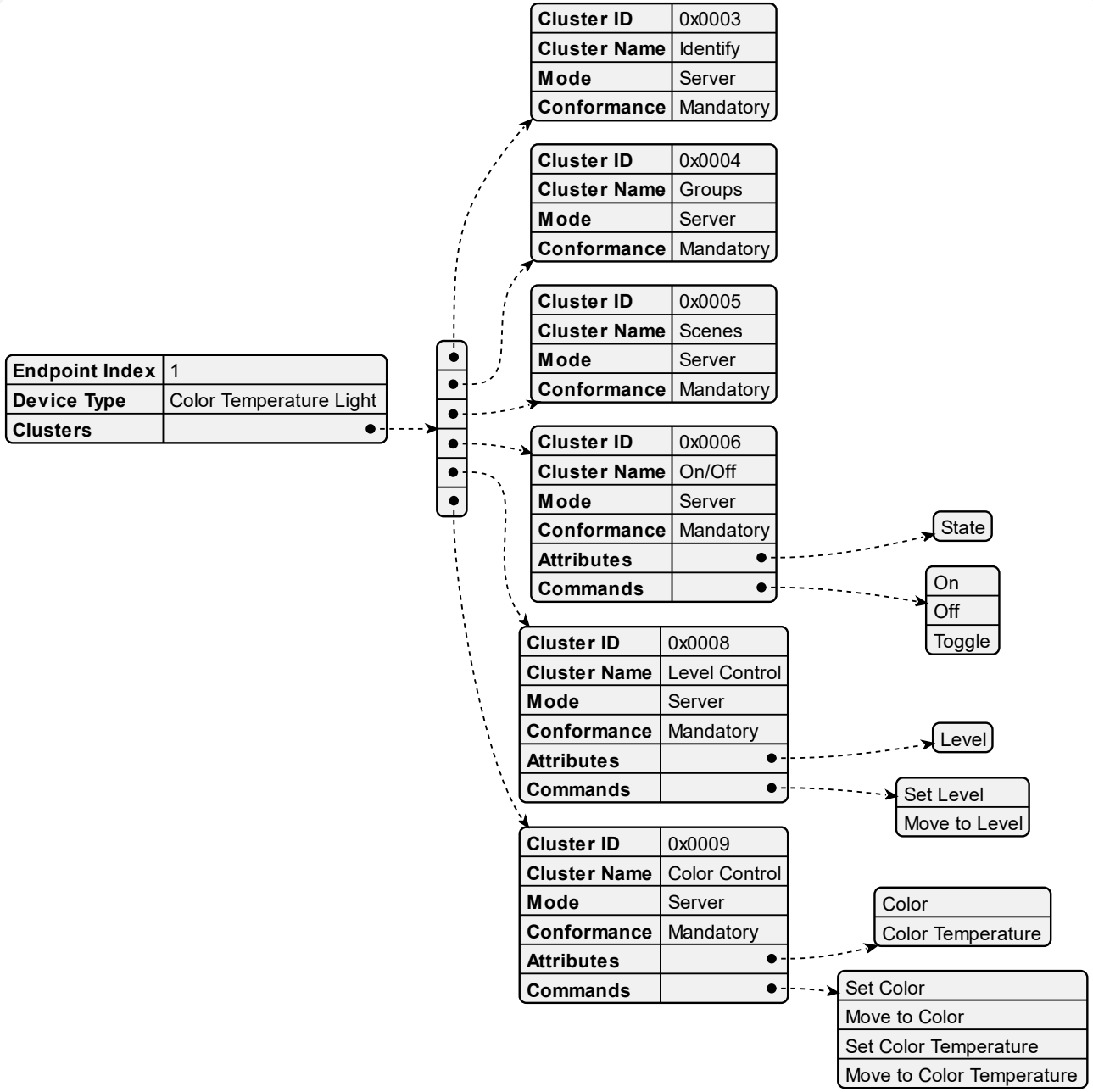
Device Types

A Matter product may contain one or more device types

- Each endpoint represents a logical device
- You could have a bulb with an integrated occupancy sensor and two endpoints, one for the light and a second for the occupancy sensor

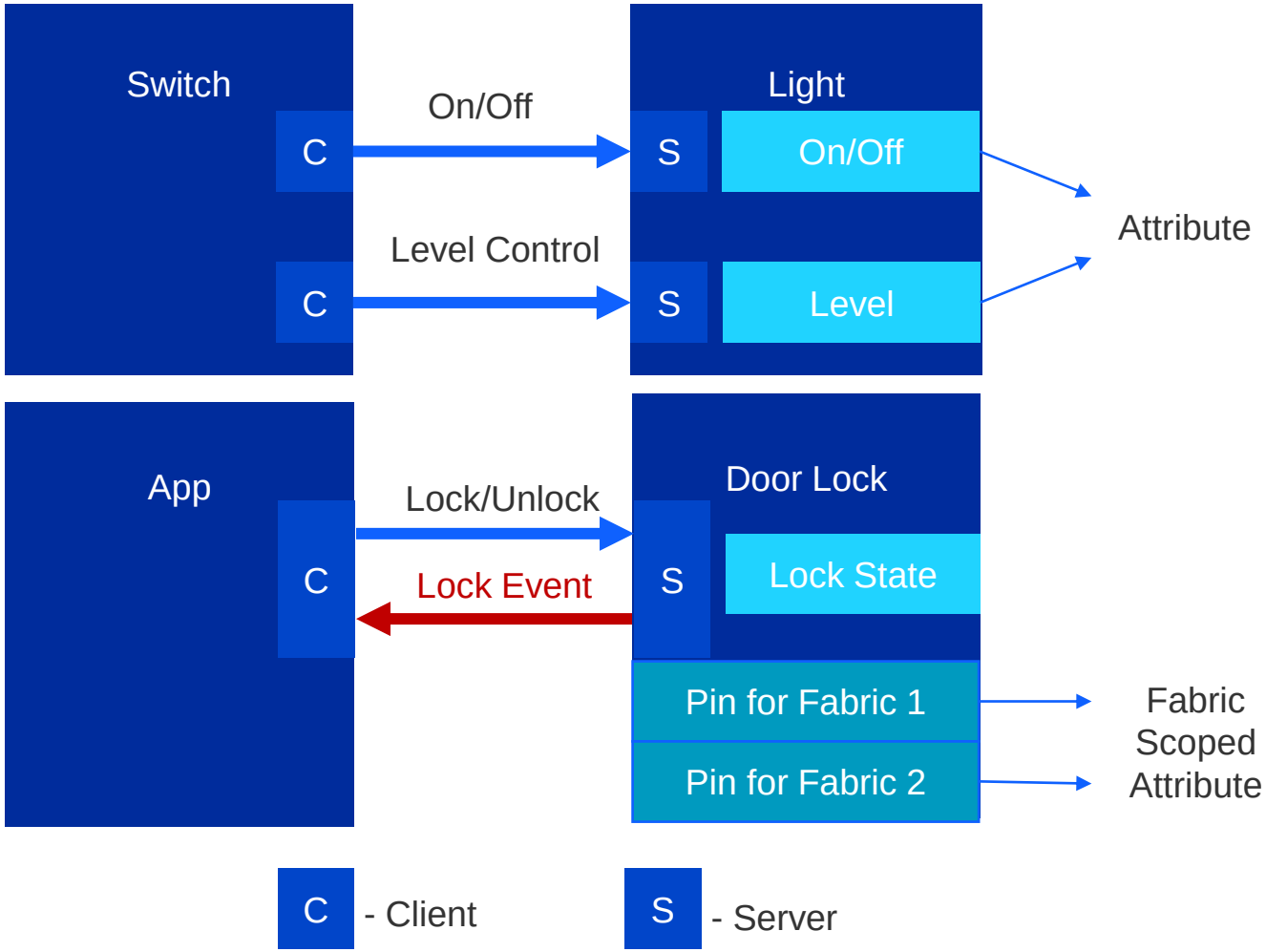
Device Types encapsulate a number of mandatory and optional clusters, attributes, and features

Example: Color Temperature Light Device Requirements



Data Model

- Client/Server communication model
  - Attributes
  - Commands
  - Events
- Inherited from Zigbee Cluster Library (ZCL)
- Security related attributes are fabric scoped



Matter Occupancy Sensor

Matter Occupancy Sensor Boards

Move commissioned light boards to a USB power supply

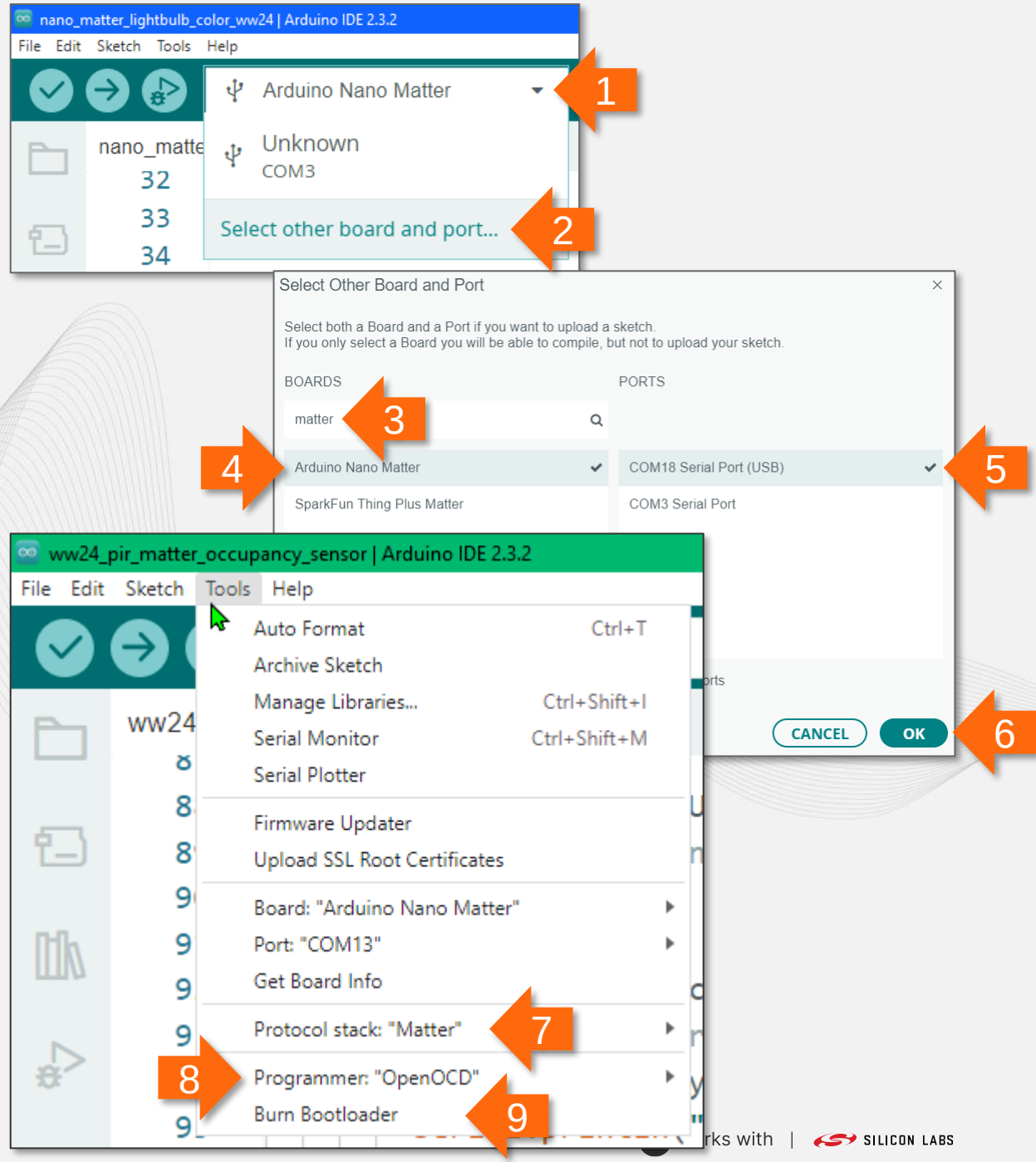
- The Arduino IDE only programs the first connected board

Connect a second board to the PC to use as an occupancy sensor and check the configuration:

1. Click the Board dropdown
2. Click Select other board and port
3. Search for matter
4. Select correct board
5. Connect board via USB and select COM port
6. Click OK button

From the Tools menu:

7. Select Protocol stack > Matter
8. Select Programmer > OpenOCD
9. Select Burn Bootloader



Matter Occupancy Sensor Example

The Silicon Labs Arduino Core includes a Matter Occupancy Sensor example

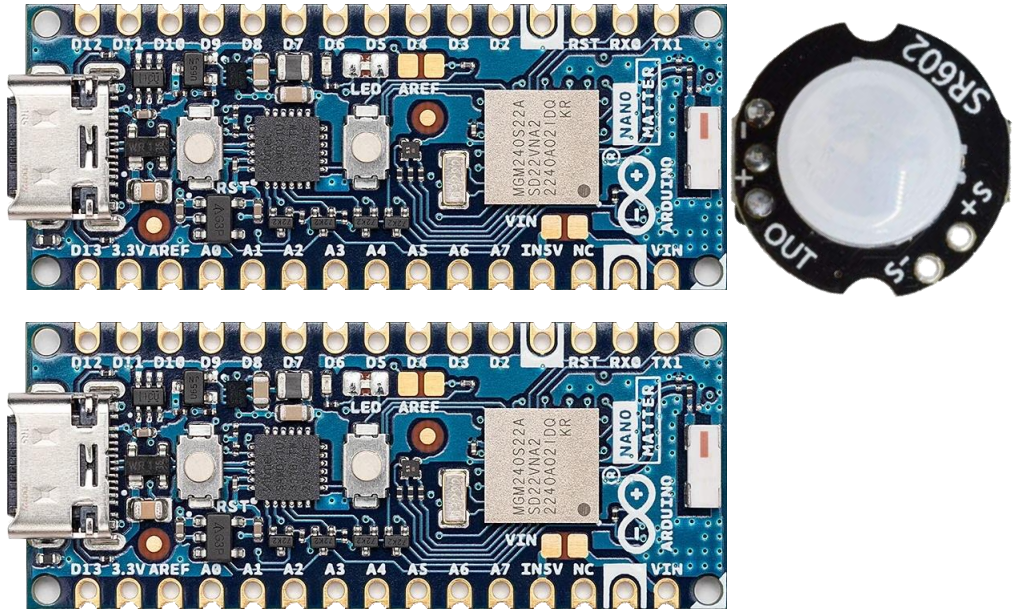
- This example alternates between occupied and unoccupied every ten seconds

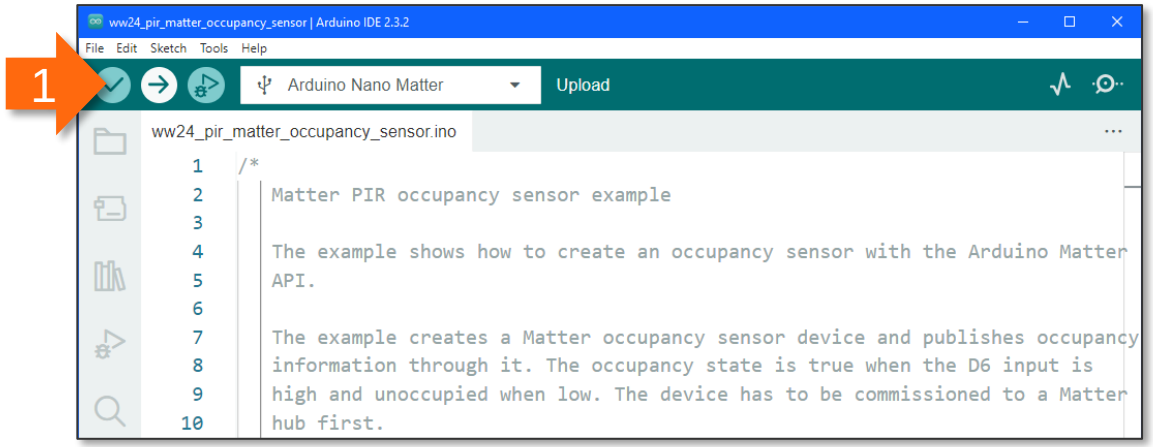
For this workshop we will use one of two adapted versions:

- The first version uses a SR602 PIR sensor to provide a realistic example
  - The green LED will be lit when in the occupied state
  - This code is in the ww24_pir_matter_occupancy_sensor folder
- The second version uses the on-board button to swap between occupied and unoccupied
  - The red LED will be lit when in the occupied state
  - This code is in the ww24_btn_matter_occupancy_sensor folder

Open the .ino file for the board you have:

1. Click the Upload button to compile and upload





Occupancy Sensor: Code Review

Occupancy Sensor: matter_occupancy_sensor.ino

PIR

```
21  #include <Matter.h>
22  #include <MatterOccupancy.h>
23
24  #define OUTPUT_OCCUPANCY LED_BUILTIN_1
25  #define INPUT_OCCUPANCY D6
26  #define INPUT_PERIOD  20
27
28  void update_output();
29
30  MatterOccupancy matter_occupancy_sensor;
31  uint32_t now_millis;
32  uint32_t input_millis;
33  uint8_t  input_occupancy = 0b10101010;
```

Button

```
20  #include <Matter.h>
21  #include <MatterOccupancy.h>
22
23  #define OUTPUT_OCCUPANCY LED_BUILTIN
24  #define INPUT_OCCUPANCY BTN_BUILTIN
25  #define INPUT_PERIOD  20
26
27  void update_output();
28
29  MatterOccupancy matter_occupancy_sensor;
30  uint32_t now_millis;
31  uint32_t input_millis;
32  uint8_t  input_occupancy = 0b10101010;
```

Some of the code is similar to the Color Lightbulb code, the key differences are:

- MatterOccupancy.h is included to provide access to the MatterOccupancy object
- Defines are used to specify the input and output pins for the sensor or button, and LED
- A define is set for the period to poll the input, 20ms
- The update_output() function prototype is used to set the LED appropriately
- An instance of the Matter Occupancy Sensor object is created
- Variables are defined to time the input polling in the main loop
- A bitmask is used to debounce the input and is initialized to an unknown state

Occupancy Sensor: setup()

```
35  void setup()
36  {
37    Serial.begin(115200);
38    Matter.begin();
39    matter_occupancy_sensor.begin();
40
41    Serial.println("WW24 PIR Matter Occupancy Sensor");
```

```
40    Serial.println("WW24 Button Matter Occupancy Sensor");
```

```
65    // Setup inputs
66    pinMode(INPUT_OCCUPANCY, INPUT);
67    // Setup outputs
68    pinMode(OUTPUT_OCCUPANCY, OUTPUT);
69    update_output();
70    // Start loop timers
71    now_millis = input_millis = millis();
72  }
```

The setup() function initialization is similar to the Color Bulb:

- The matter and matter_occupancy_sensor objects are started
- The commissioning, network and online checks are the same (not shown)
- The input pin is initialized
- The LED pin is initialized and set using the update_output() function
- The timing variables are initialized

Occupancy Sensor: loop() – Debouncing

```
74  void loop()
75  {
76    // Update time
77    now_millis = millis();
78    // Input timer fired
79    if (now_millis - input_millis >= INPUT_PERIOD) {
80      uint8_t old_input;
81      // Restart timer
82      input_millis = now_millis;
83      // Note old debounce value
84      old_input = input_occupancy;
85      // Shift debounce
86      input_occupancy <<= 1;
87      // Read input
88      if (digitalRead(INPUT_OCCUPANCY) == HIGH) input_occupancy |= 1;
```

The loop() monitors the input controlling the occupied state:

- The timers are checked to see if the polling period has expired
- The old debounce value is retained to check for changes
- The debounce mask is shifted left
- If the input is high a 1 is placed in the least significant bit

Occupancy Sensor: loop() – Occupancy State

When the PIR debounce value is changed:

- When none of the bits are set, the occupancy state is set to unoccupied
- When all the bits are set, the occupancy state is set to occupied
- The state can be set by writing directly to the matter_occupany_sensor object
- The update_output() function is called to update the LED

```
89      // Debounce has changed?
90      if (input_occupancy != old_input) {
91        // Unoccupied ?
92        if (input_occupancy == 0) {
93          // Update matter state
94          matter_occupancy_sensor = false;
95          Serial.println("PIR - Unoccupied");
96          update_output();
97        }
98        // Occupied ?
99        else if (input_occupancy == 0b11111111) {
100         // Update matter state
101         matter_occupancy_sensor = true;
102         Serial.println("PIR - Occupied");
103         update_output();
104       }
105     }
```

When the button debounce value is changed:

- When none of the bits are set, the button has been pressed
- The occupancy state is toggled
- The current state is obtained using the get_occupancy() function
- The state can be set by writing directly to the matter_occupany_sensor object
- The update_output() function is called to update the LED

```
88      // Debounce has changed?
89      if (input_occupancy != old_input) {
90        // Occupancy button pressed ?
91        if (input_occupancy == 0) {
92          // Toggle matter state
93          matter_occupancy_sensor = !matter_occupancy_sensor.get_occupanc
94          Serial.print("Occupancy Pressed");
95          if (matter_occupancy_sensor.get_occupancy()) Serial.print(" - O
96          else                                         Serial.print(" - U
97          update_output();
98        }
99      }
```

Occupancy Sensor: update_output()

```
109  void update_output() {
110    // Apply matter state to output
111    if (matter_occupancy_sensor.get_occupancy()) {
112      digitalWrite(OUTPUT_OCCUPANCY, LED_BUILTIN_ACTIVE);
113    }
114    else {
115      digitalWrite(OUTPUT_OCCUPANCY, LED_BUILTIN_INACTIVE);
116    }
117  }
```

The update_output() function:

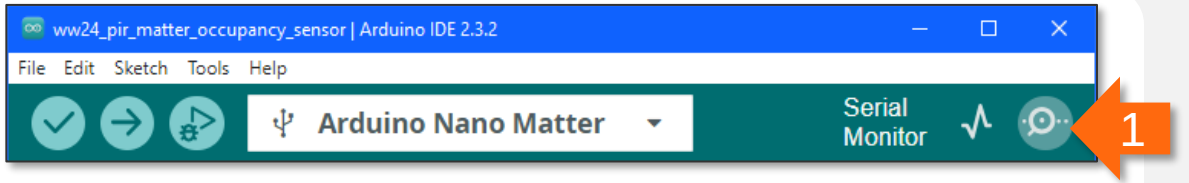
- Sets the occupied state LED to match the Matter occupied state

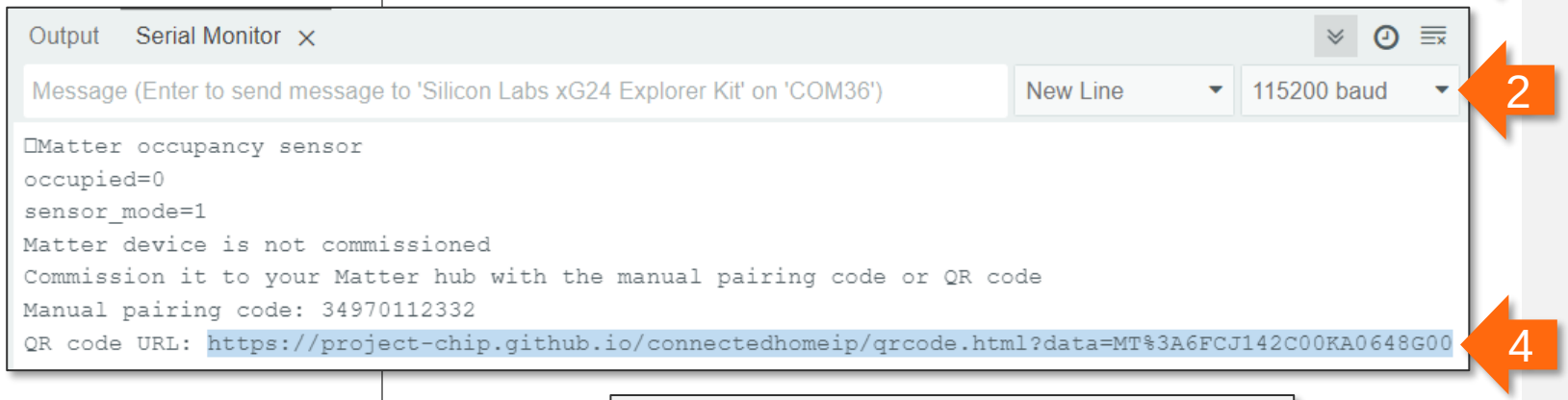
Occupancy Sensor: Commissioning and Operation

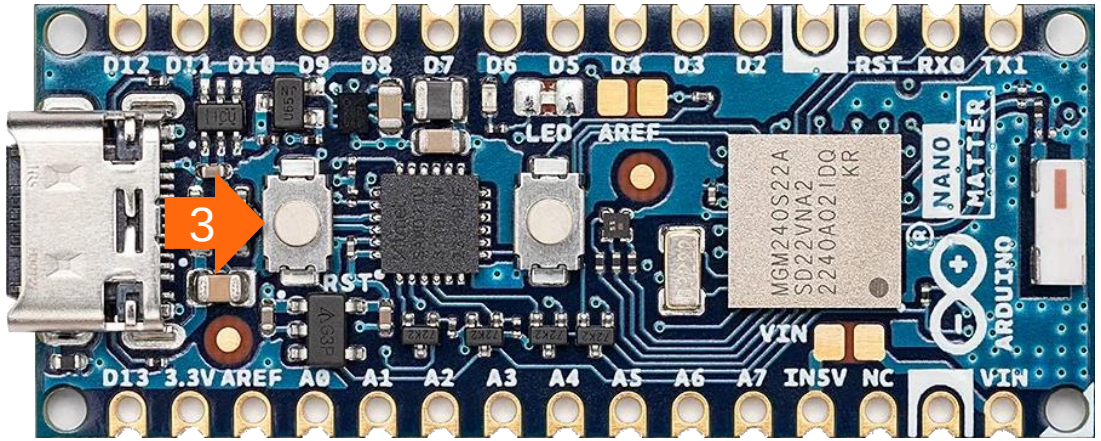
Occupancy Sensor: Commissioning Codes

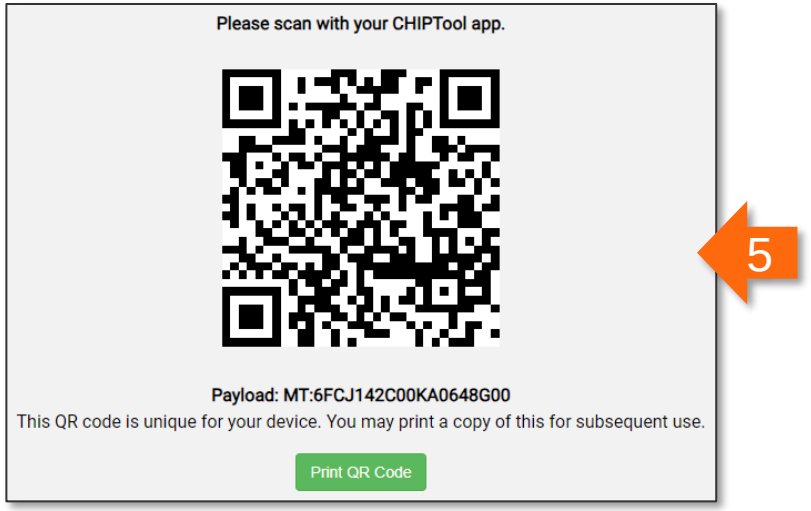
To obtain commissioning codes:

1. Open the Serial Monitor
2. Set baud rate to 115200
3. Reset board
4. Copy the QR Code URL from the debug output
5. Paste URL into a browser







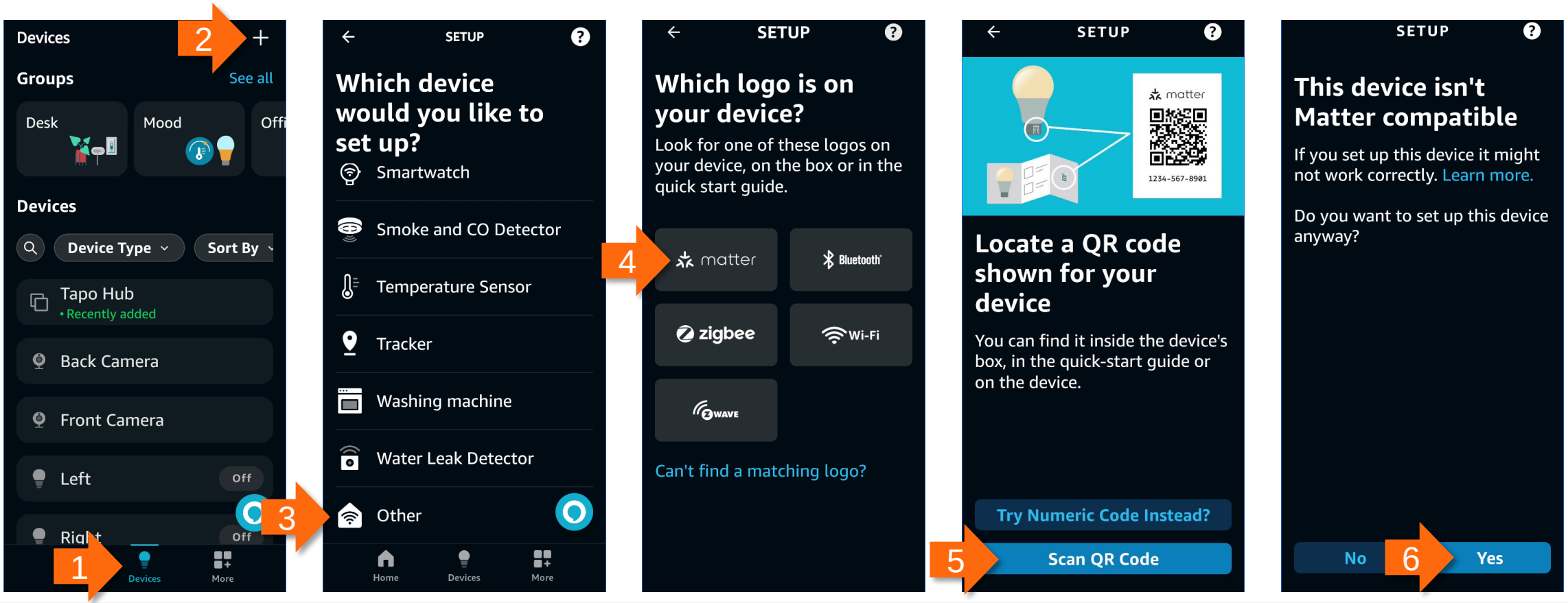


Occupancy Sensor: Commissioning Add

The commissioning process is similar in all the ecosystems

Test certificates are used by the example applications

6. There may be warnings of this during commissioning



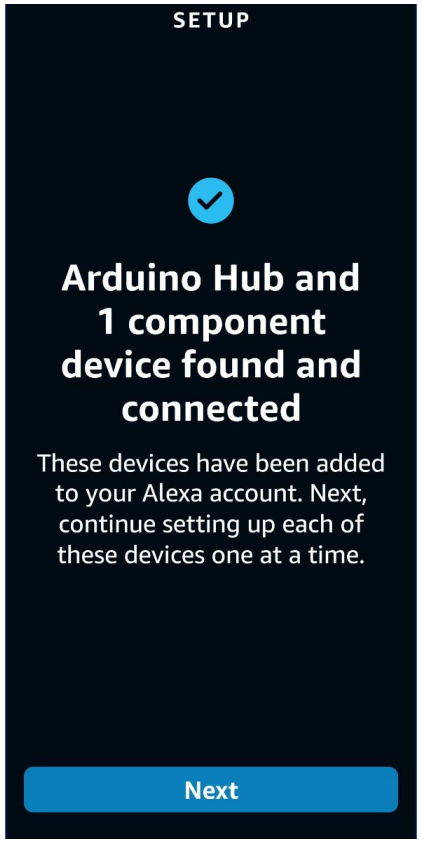
Occupancy Sensor: Device Setup

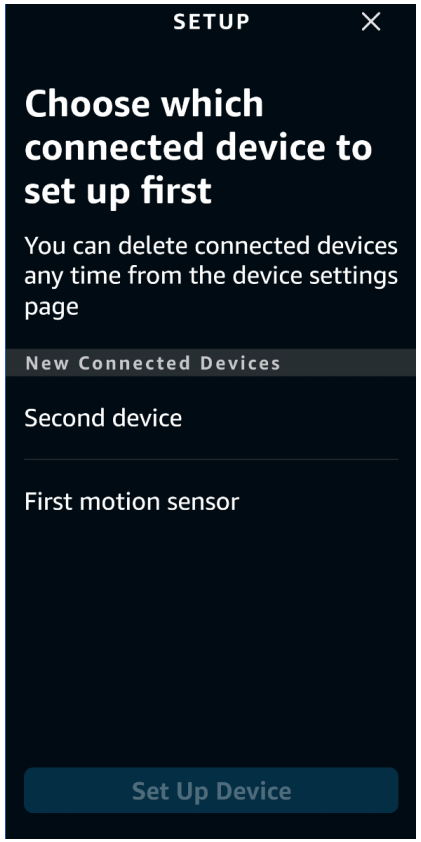
Two devices are found

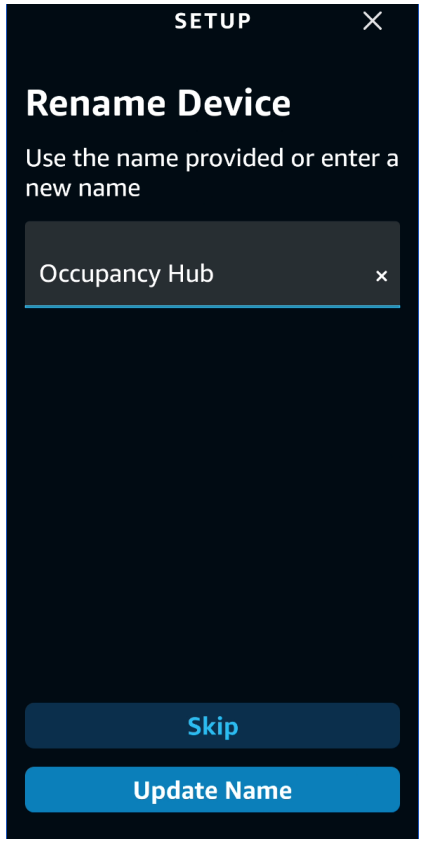
- The First motion sensor is the occupancy sensor
- The Second device is a hub and is a side-effect of creating the sensor endpoint at runtime

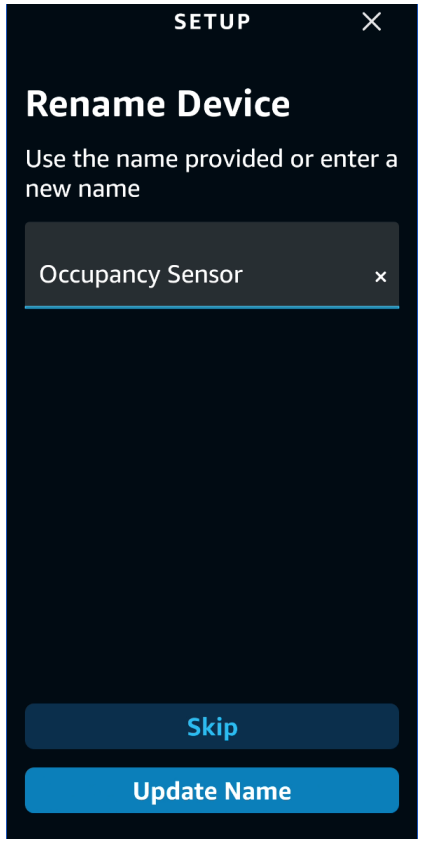
Select each device and click Set Up Device

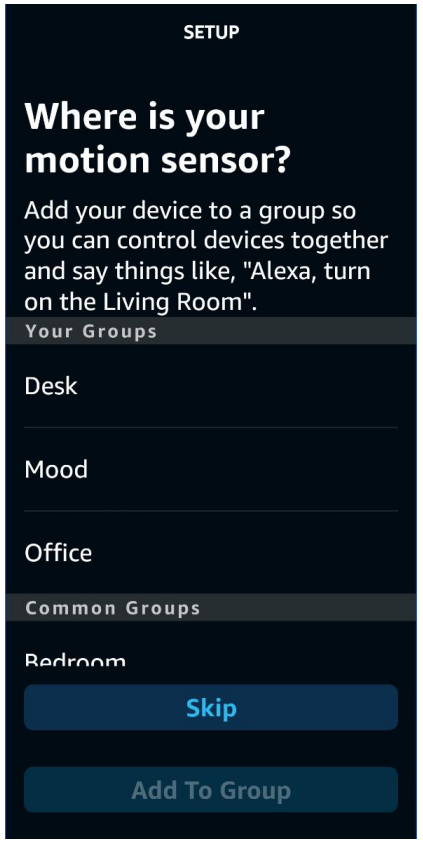
- Devices can be renamed
- The occupancy sensor can be added to a group







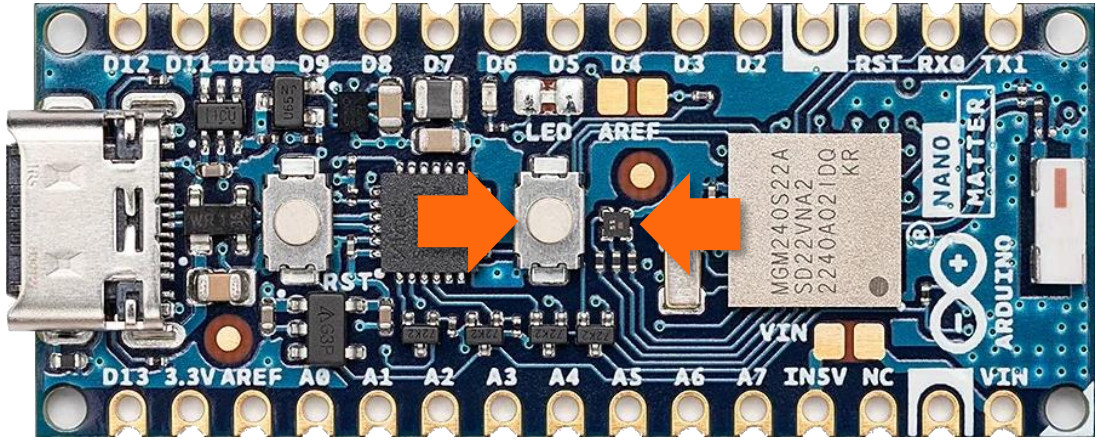


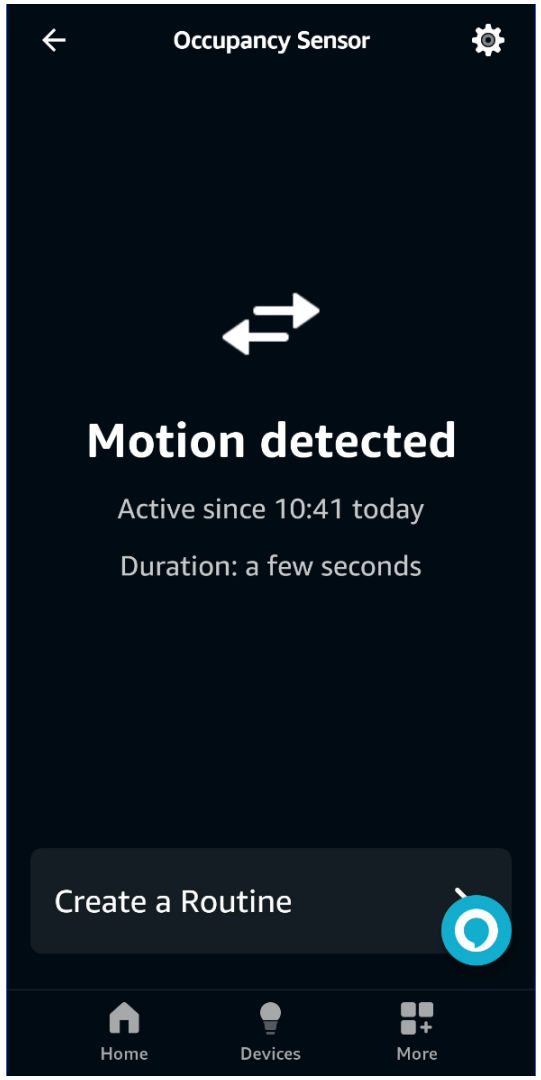


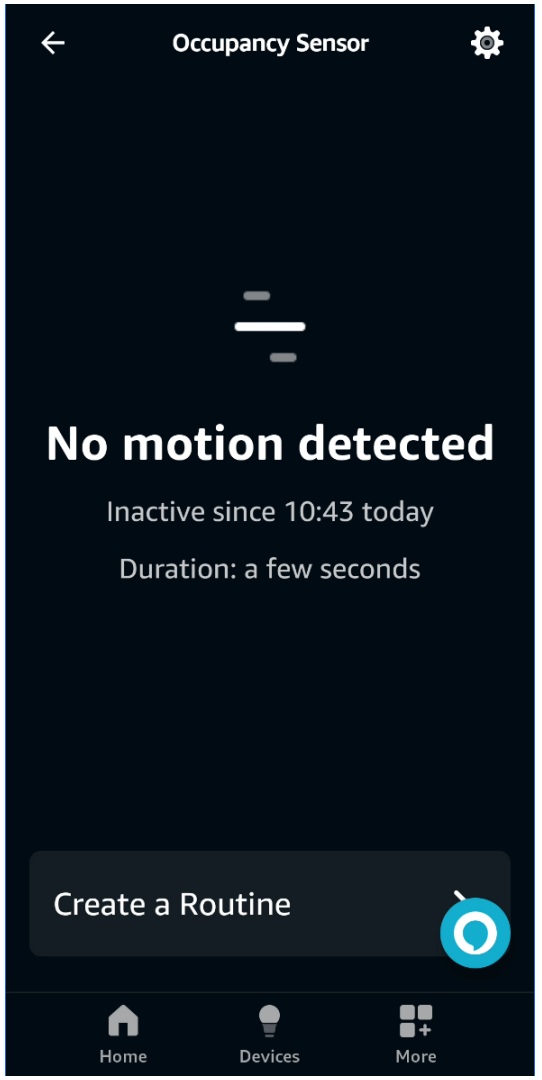
Occupancy Sensor: Operation

Opening the Occupancy Sensor device in the app allows the state to be monitored

- The PIR sensor will work automatically
  - Covering the sensor can force the unoccupied state
- The button sensor toggles the state when the button is pressed
- The RGB LED is lit when in the occupied state on both sensor types
  - The PIR sensor light is green
  - The button sensor light is red
- Alexa introduces a delay when moving to unoccupied







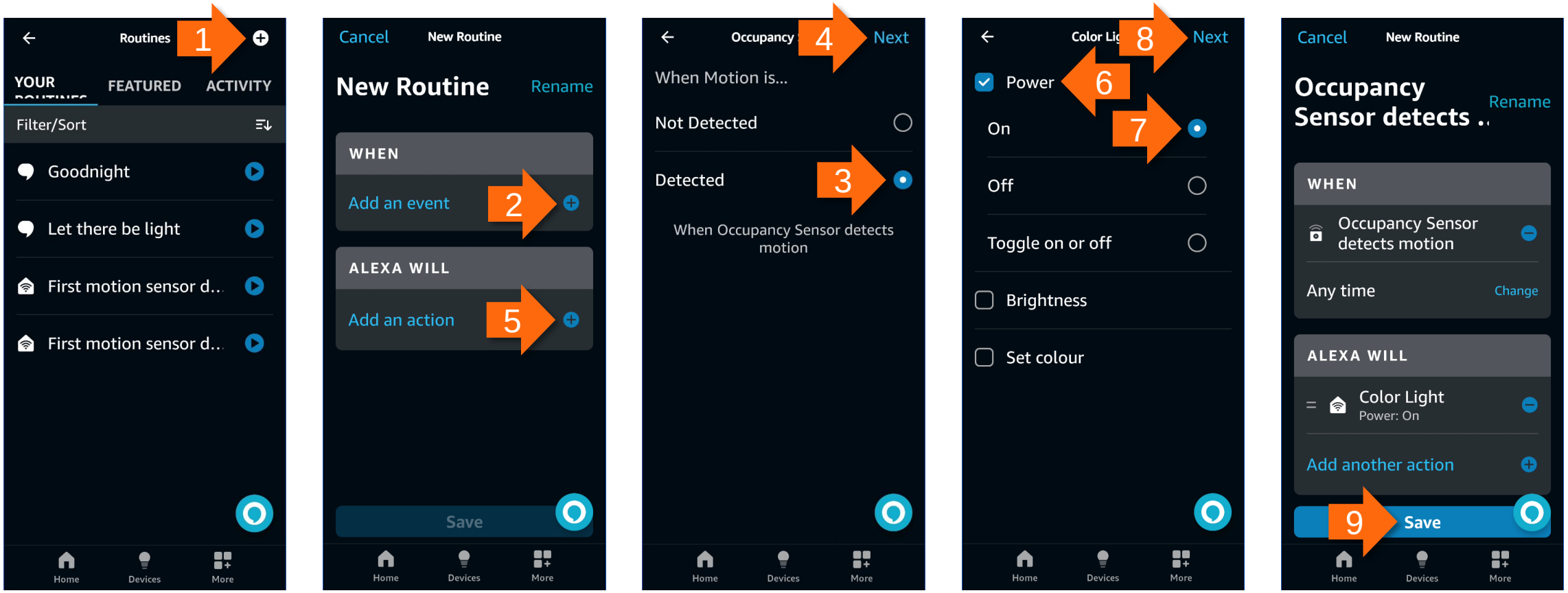
Occupancy Sensor: Occupied Routine

Adding a routine involves an event that will trigger the routine:

- The Occupancy Sensor going to the occupied state

The action that will take place also needs to be specified:

- Turn on the color light



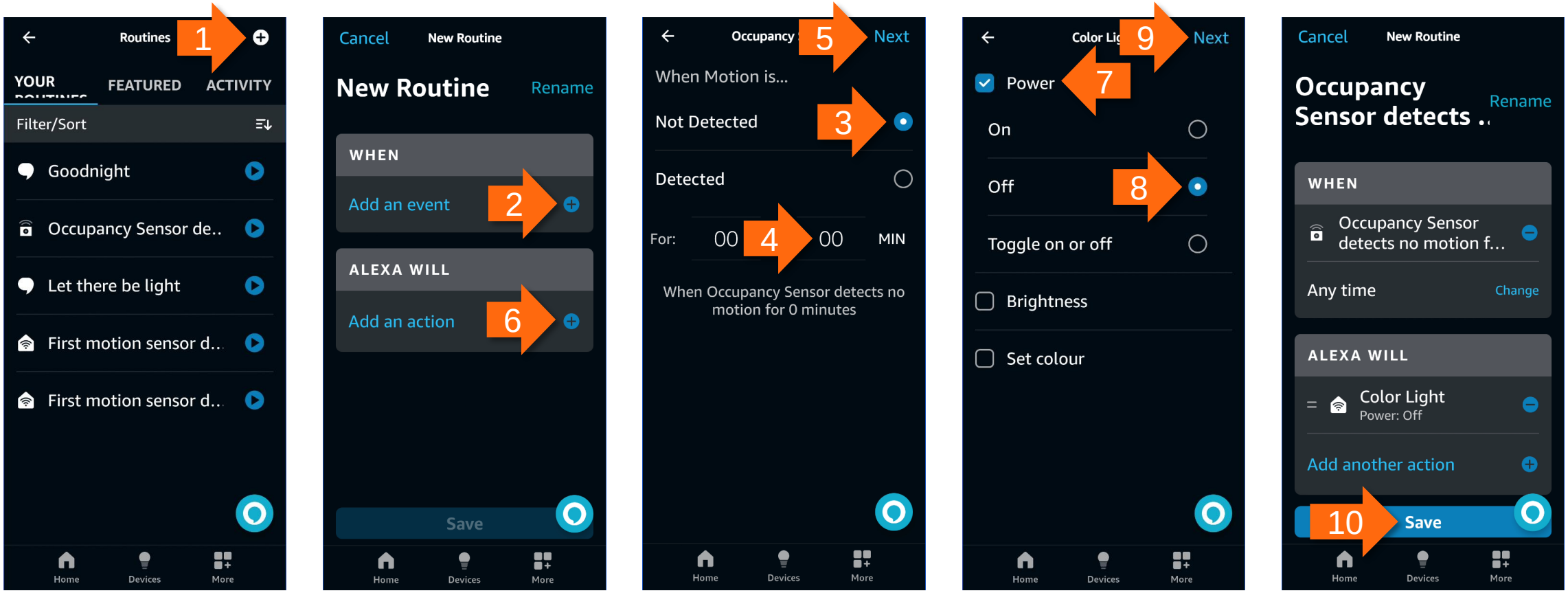
Occupancy Sensor: Unoccupied Routine

Adding a routine involves an event that will trigger the routine:

- The Occupancy Sensor going to the unoccupied state

The action that will take place also needs to be specified:

- Turn off the color light



Occupancy Sensor: Theory

Subscriptions and Reporting

The hub could poll the Occupancy Sensor's Occupied attribute for monitoring

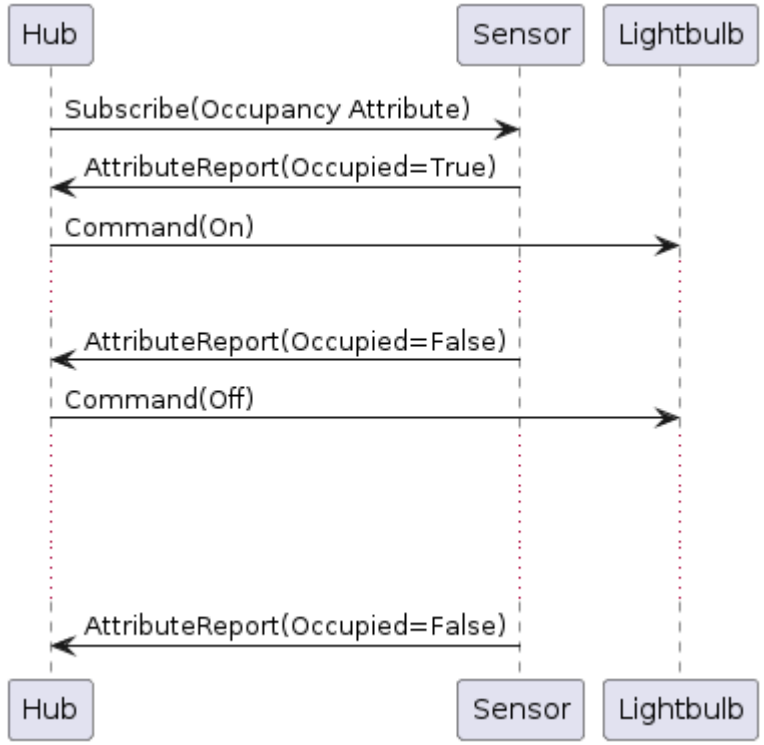
- But this would be inefficient with unneeded communications
- The polling interval may impose delays

Instead, the hub subscribes to the Occupied attribute

- The Occupancy Sensor then sends an Attribute Report to the hub whenever the Occupied attribute changes
- Communications are only used when the value is changed
  - A maximum report interval can be set by the hub to receive reports even when the attribute is unchanged

When the hub receives Attribute Reports:

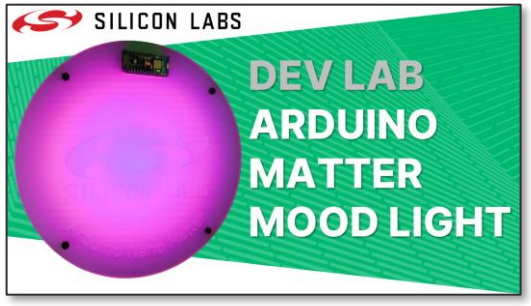
- Commands are sent to the lightbulb to turn it on or off

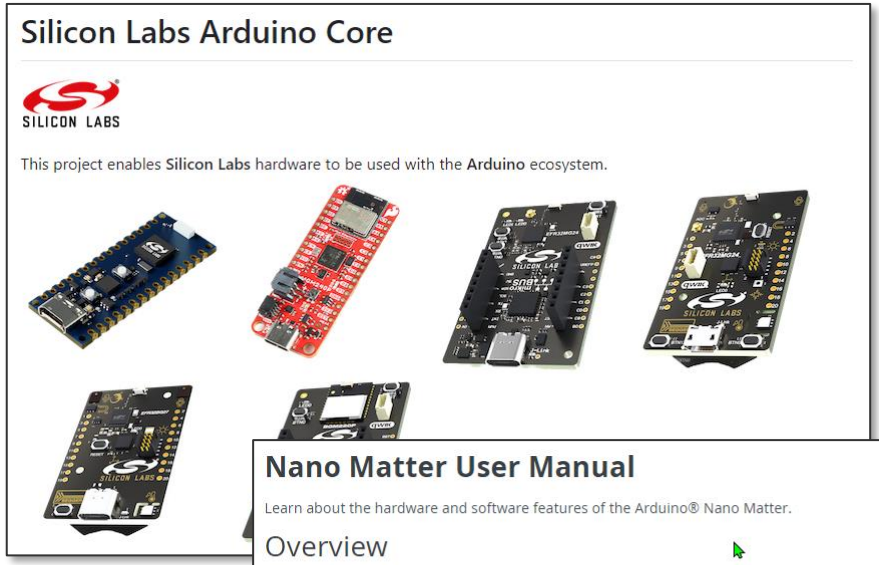


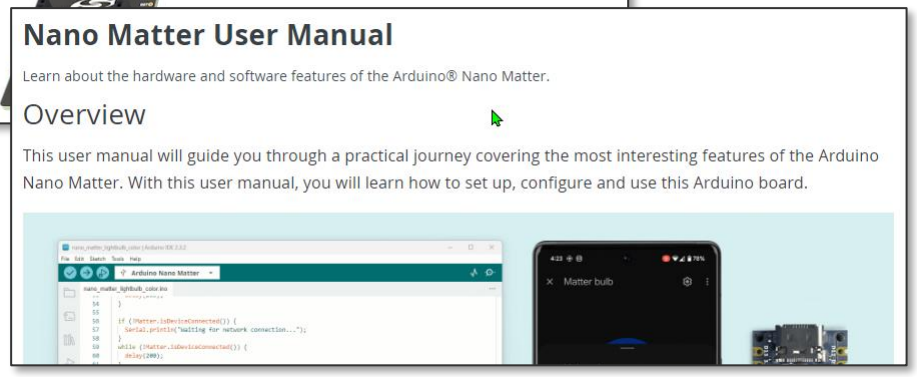
Next Steps

Next Steps – Matter over Thread in Arduino IDE

- There are lots more Matter examples to explore in the Arduino IDE
- Quick Start – Arduino Nano Matter video: https://youtu.be/7pXFT4D3ui4
  - Walks through the setup of the Arduino IDE for Matter
  - Adapts the Matter Dimmable Lightbulb example to animate LEDs
- Dev Lab – Arduino Matter Mood Light video: https://youtu.be/IpvyQGEW5cw
  - Creates a mood light using a single device with two color bulb endpoints, creating plasma effects between the two colors
- Silicon Labs Arduino Core on GitHub: https://github.com/SiliconLabs/arduino
  - Contains a useful readme detailing pin outs for all the boards
  - Contains the source code for the Arduino Core
- Arduino Nano Matter User Manual: https://docs.arduino.cc/tutorials/nano-matter/user-manual
  - Walks through setting up the board
  - Has tutorials including commissioning into the major Matter ecosystems



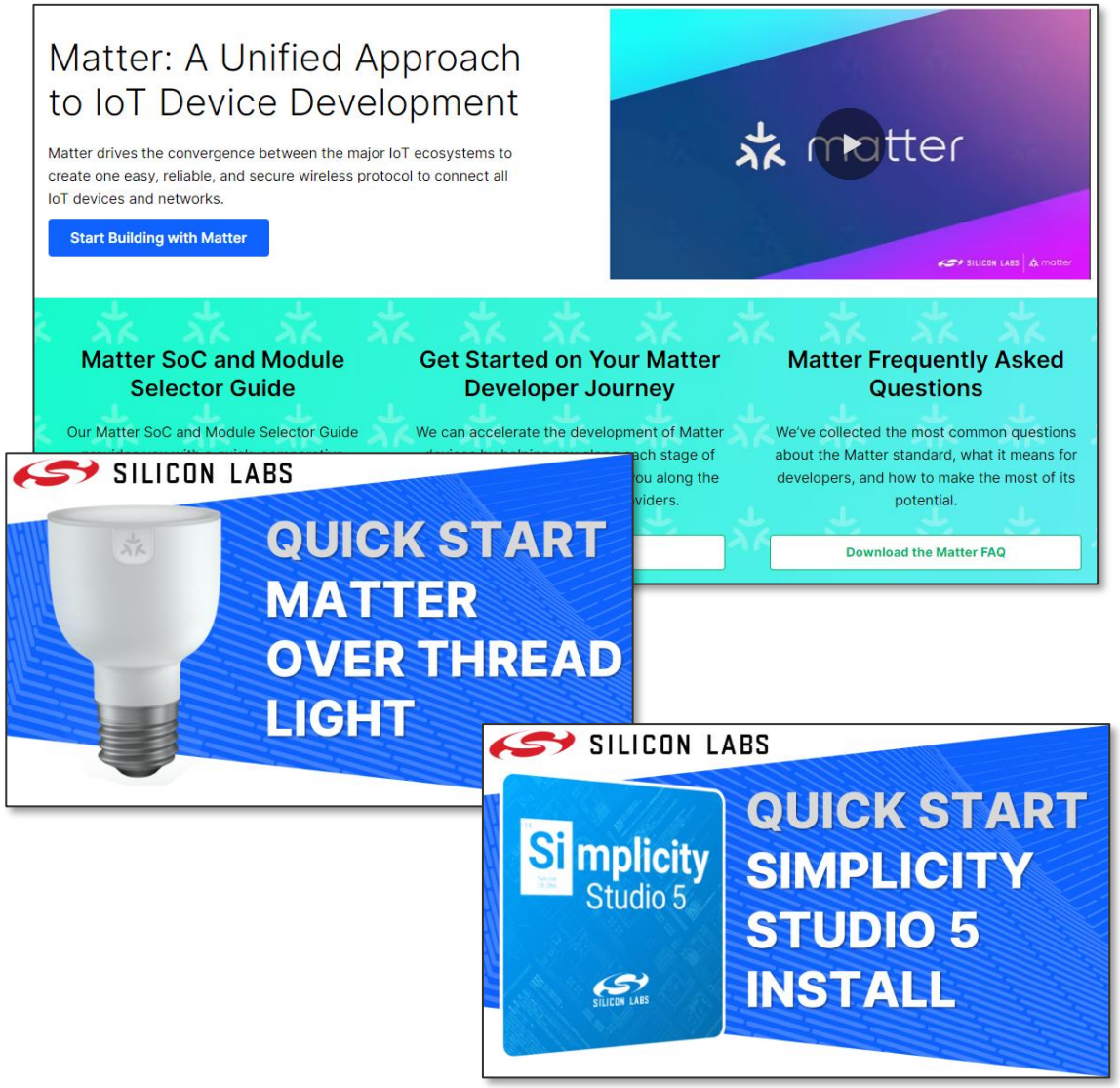




Next Steps – Matter in Simplicity Studio

Simplicity Studio is the IDE for all Silicon Labs boards and technologies including Matter over Thread and Matter over Wi-Fi

- Simplicity Studio can be downloaded from: https://www.silabs.com/developers/simplicity-studio
- Example applications are built into the IDE
- Additional example applications are available from GitHub: https://github.com/SiliconLabs/matter
- General information is available from the Silicon Labs website: https://www.silabs.com/wireless/matter
- The Matter Developer Journey walks through the process of creating a Matter product: https://www.silabs.com/wireless/matter/matter-developer-journey
- For detailed information on developing Matter applications: https://docs.silabs.com/matter/latest/matter-start
- There are a series of Quick Start videos on YouTube covering Matter over Thread development in Simplicity Studio, start from: https://youtu.be/Dlcbz7X2yKE
- A series of Quick Start videos also covers the installation and use of the Simplicity Studio IDE, start from: https://youtu.be/Pd1E2ZL6F0c



MAT-201

Thank You