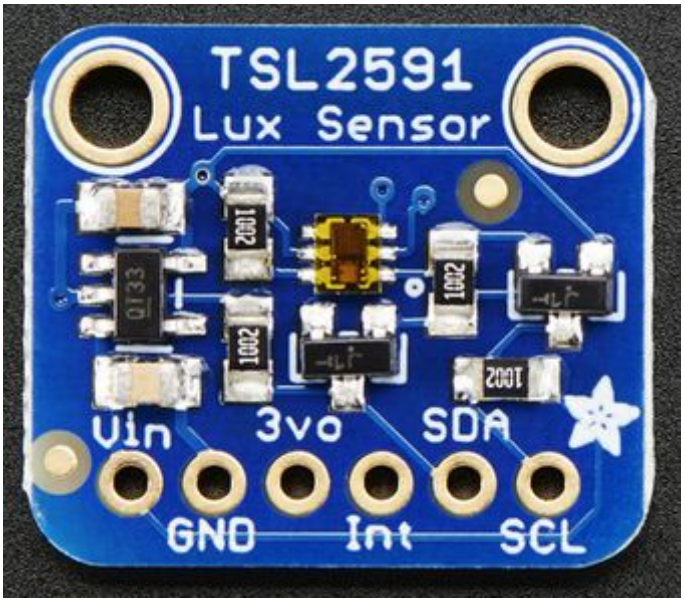
Using class-based Arduino libraries in XOD

XOD documentation
XOD forum
Existing XOD libraries
Arduino libraries

Overview

Arduino libraries exist for a huge range of breakout boards and other devices (see https://www.arduinolibraries.info/). If you have a little C++ experience, it is easy to incorporate these libraries into XOD.

In this tutorial we will create a XOD library for the TSL2591 high dynamic range digital light sensor. Adafruit produce a breakout board for this sensor: https://learn.adafruit.com/adafruit-tsl2591/



001_tls2591-breakout.png

New device

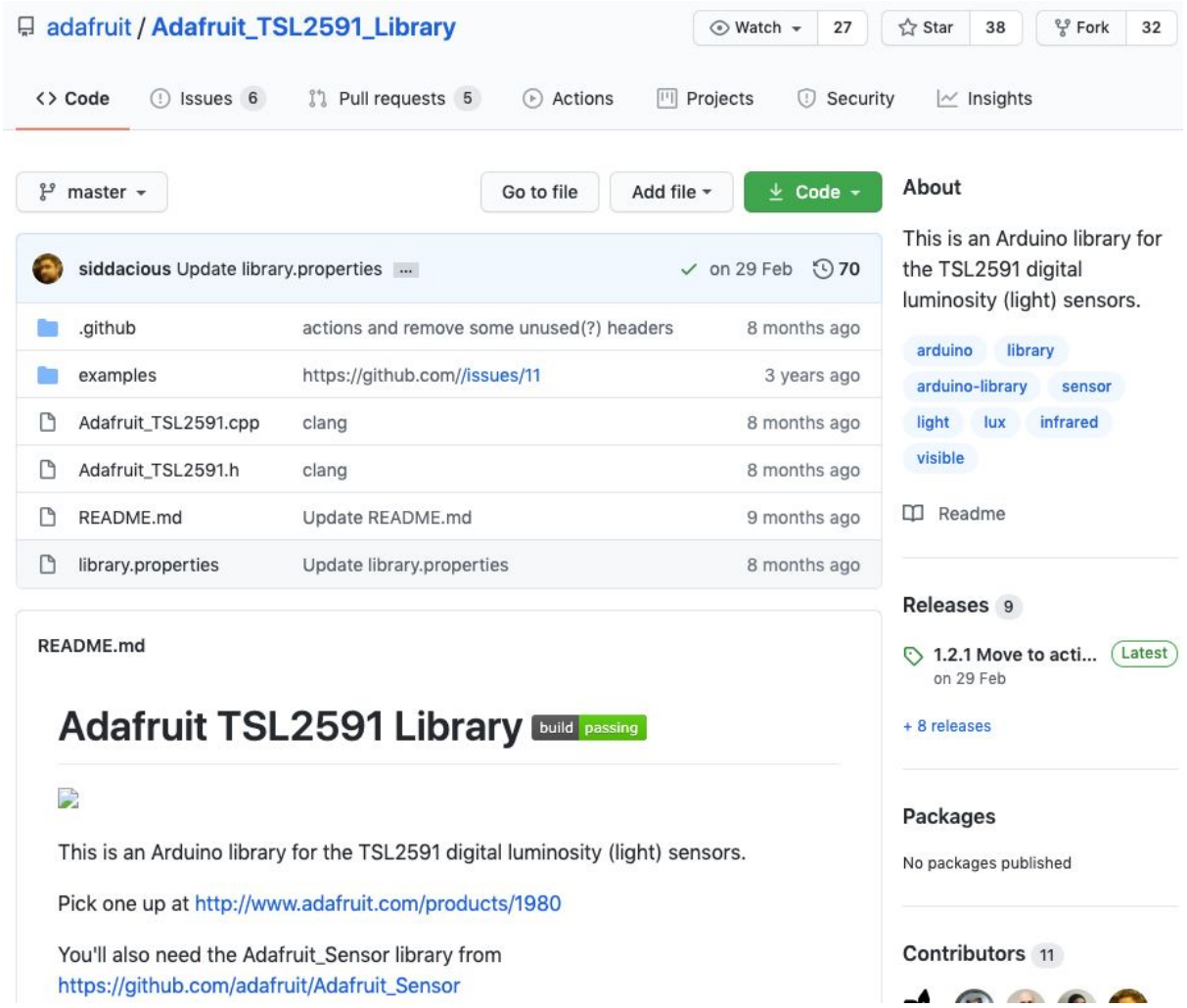
When presented with a new device the first thing you should do is check if it is already supported in XOD. Fortunately there is a searchable database of core and contributed libraries:
https://xod.io/libs/

If you search for “light sensor” or “TSL2591” you will find that a library already exists for this device (https://xod.io/libs/wayland/tsl2591-light-sensor/). However, for the purposes of this tutorial, we will pretend that there is no library for the TSL2591.

Find Arduino library for device

If you cannot find a XOD library for your device, you will need to look for a class-based Arduino library. Manufacturers of breakout boards typically provide C++ libraries for their devices. On the product pages of companies such as Adafruit, Polulu and Sparkfun you will typically find links to code repositories. For more unusual devices a web search will often find libraries developed by hobbyists.

Adafruit's code repository for their TSL2591 library is on github:
https://github.com/adafruit/Adafruit_TSL2591_Library



002_adafruit_tls2591_library.png

Test the Arduino library

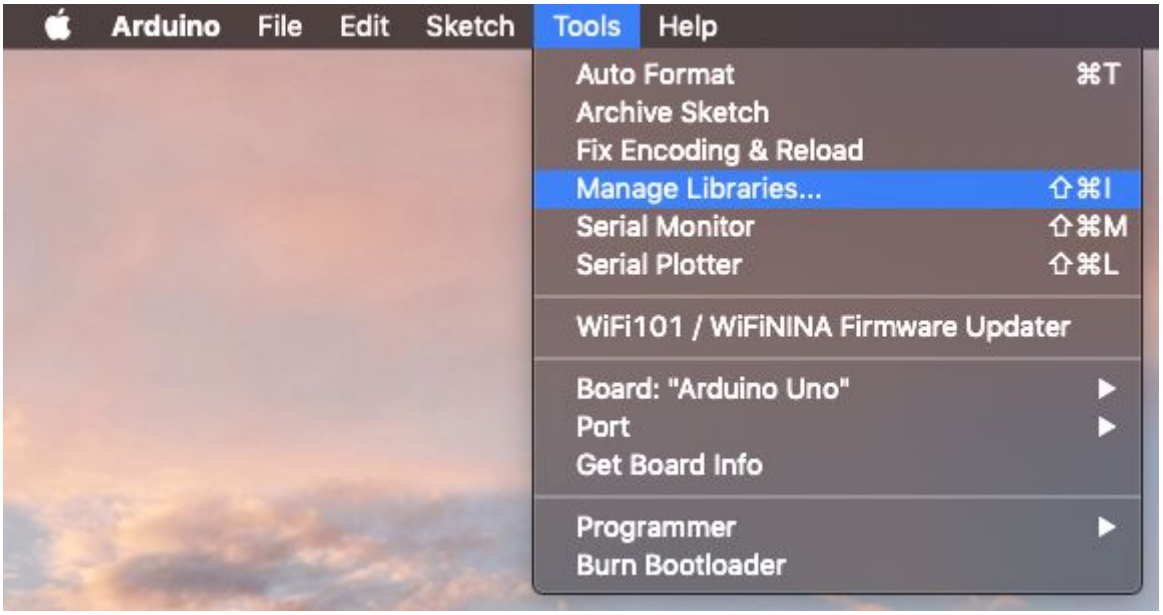
Once you've found a library for your device it is a good idea to test it using the Arduino IDE. Well written libraries will include example sketches. Reading through the sketches can help you to understand how the methods in the library are used.

Install Arduino IDE

Download and install Arduino IDE on your computer:
https://www.arduino.cc/en/Main/Software

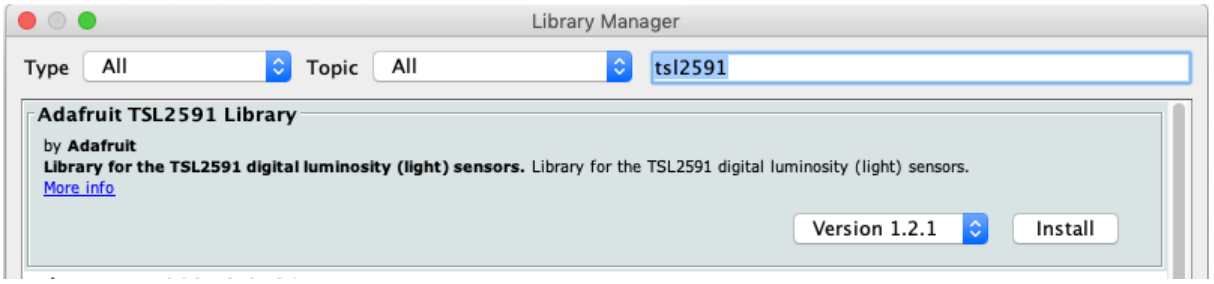
Add library to IDE

From the **Tools** menu select **Manage Libraries...**



003_Arduino-IDE-manage-install-lib.png

In the **Library Manager** search for **tsl2591**. Select the most recent version of the Adafruit TSL2591 Library and click **Install**.



004_Arduino-IDE-manage-install-lib.png

You will receive the following prompt informing you that the **Adafruit TSL2591 Library:1.2.1** is dependent on another library, the **Adafruit Unified Sensor**. Click **Install all**.

005_Adafruit_Unified_Sensor_prompt.png

Run an example sketch

Running an example sketch is a good way of checking that:

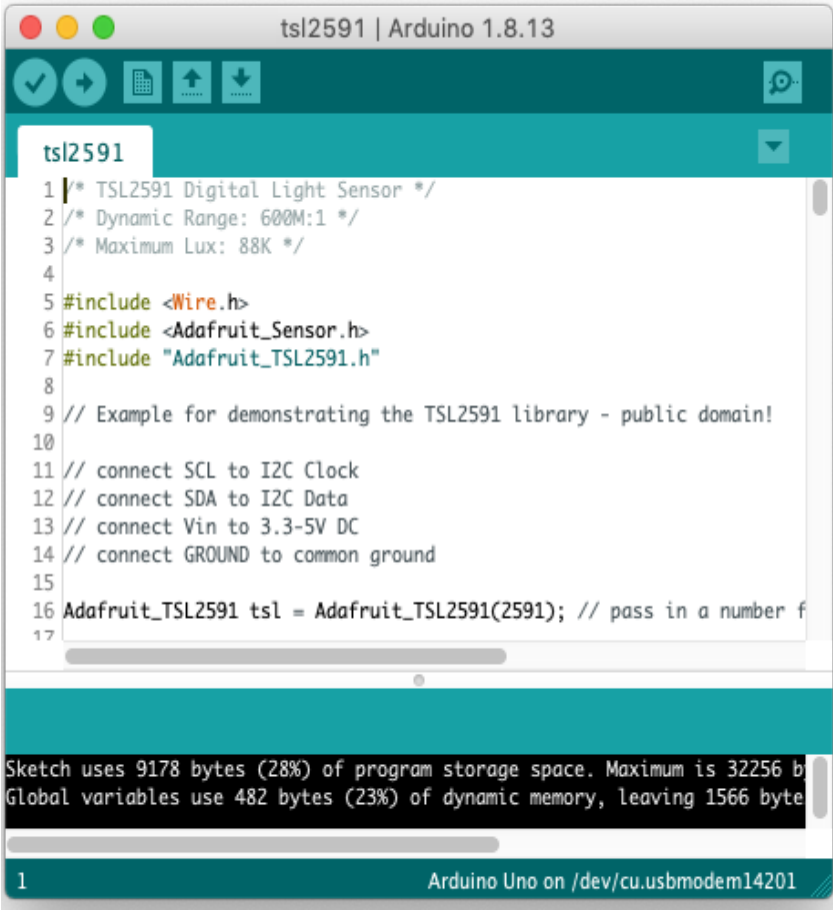
- The device is wired correctly to the Arduino board.
- The device is working.
- The library is working.

Open an example sketch:

File → Examples → Adafruit TSL2591 → tsl2591

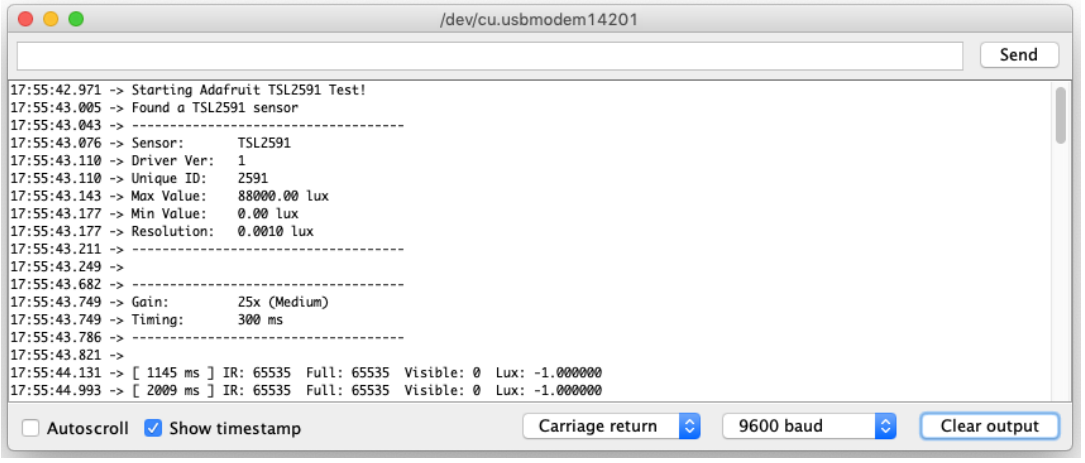
006_open_example_sketch.png

This example sketch will transmit data via serial. Click on the **Upload** button.



008_tsl2591_sketch.png

Once the program is running on the Arduino, you can open the **serial monitor**:
Tools → Serial Monitor
The example sketch transmits serial at **9600 baud**, so make sure this speed is selected in the **serial monitor**. If everything is working data will be printed to the **serial monitor**.



009_Arduino-IDE-test-library.png

We are now ready to start working in XOD.

Inspect the Arduino library

Dependencies

The readme file for the Adafruit TSL2591 Library tells us that we also need the Adafruit_Sensor library from https://github.com/adafruit/Adafruit_Sensor. We don't need to rely on a readme file to inform us of dependencies, as they will also be declared in the library header file (Adafruit_TSL2591.h).

```
/**************************************************************************/
/*!
    @file     Adafruit_TSL2591.h
    @author   KT0WN (adafruit.com)

    This is a library for the Adafruit TSL2591 breakout board
    This library works with the Adafruit TSL2591 breakout
    ----> https://www.adafruit.com/products/1980

    Check out the links above for our tutorials and wiring diagrams
    These chips use I2C to communicate

    Adafruit invests time and resources providing this open source code,
    please support Adafruit and open-source hardware by purchasing
    products from Adafruit!
*/
/**************************************************************************/

#ifndef _TSL2591_H_
#define _TSL2591_H_

#include <Adafruit_Sensor.h>
#include <Arduino.h>
#include <Wire.h>
```

024_tsl2591_header_top.png

Class declaration

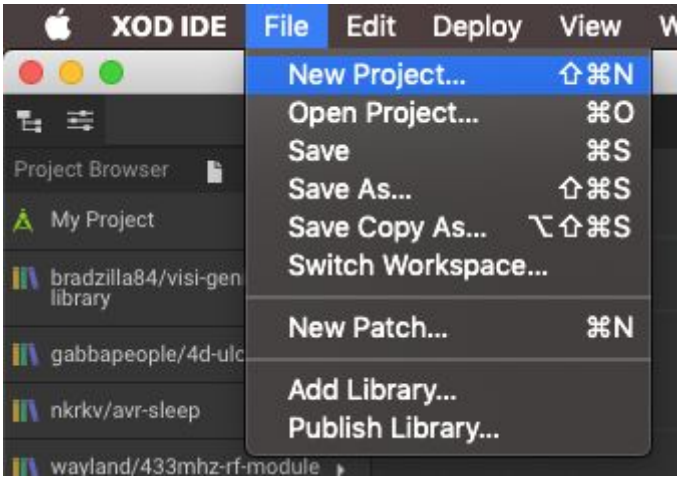
The public interface to the class provides the class constructor and various member functions. We need to create an action node for each of the member functions we want to use in XOD. We'll see how this is done in the next section.

```
131  class Adafruit_TSL2591 : public Adafruit_Sensor {
132  public:
133    Adafruit_TSL2591(int32_t sensorID = -1);
134
135    boolean begin(TwoWire *theWire);
136    boolean begin();
137    void enable(void);
138    void disable(void);
139
140    float calculateLux(uint16_t ch0, uint16_t ch1);
141    void setGain(tsl2591Gain_t gain);
142    void setTiming(tsl2591IntegrationTime_t integration);
143    uint16_t getLuminosity(uint8_t channel);
144    uint32_t getFullLuminosity();
145
146    tsl2591IntegrationTime_t getTiming();
147    tsl2591Gain_t getGain();
```

025_tsl2591_class.png

Start a new XOD project

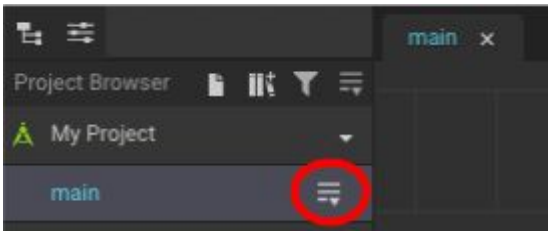
There is no technical difference between a project and a library. To start a new library click: **File → New Project...**



011_new_xod_project.png

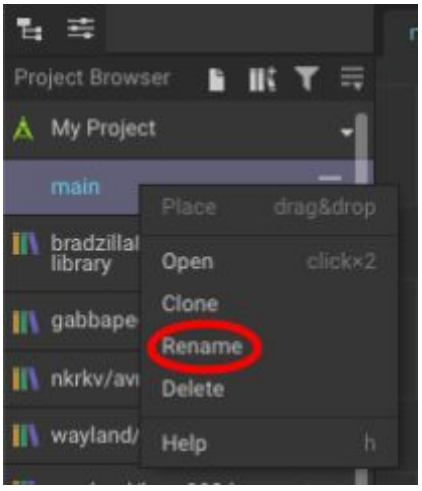
Create a new device

We need to declare a new custom type to represent our hardware device. New XOD projects start with a single patch called **main**. We will rename this patch **tsl2591-device**. By convention, nodes that create a new type to work with hardware are given the suffix **-device**. Go to the **Project Browser** and either left-click on the menu icon or right-click on **main**.



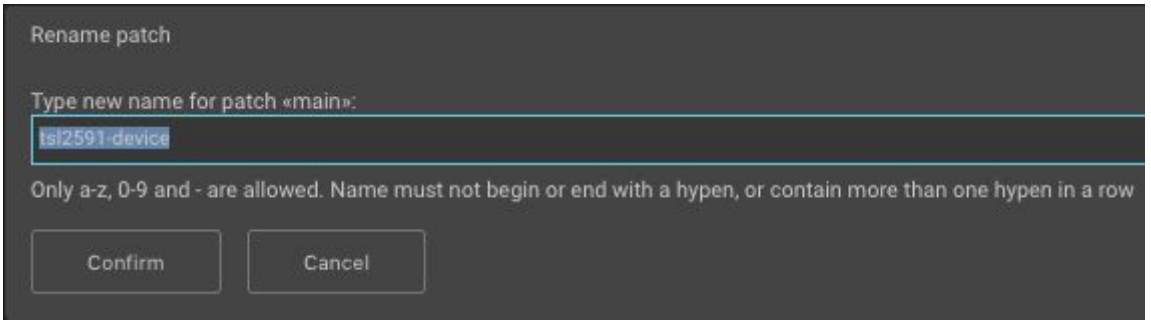
012_patch_name_main.png

This will bring up a context menu with various options including **Rename**.



014_context_menu.png

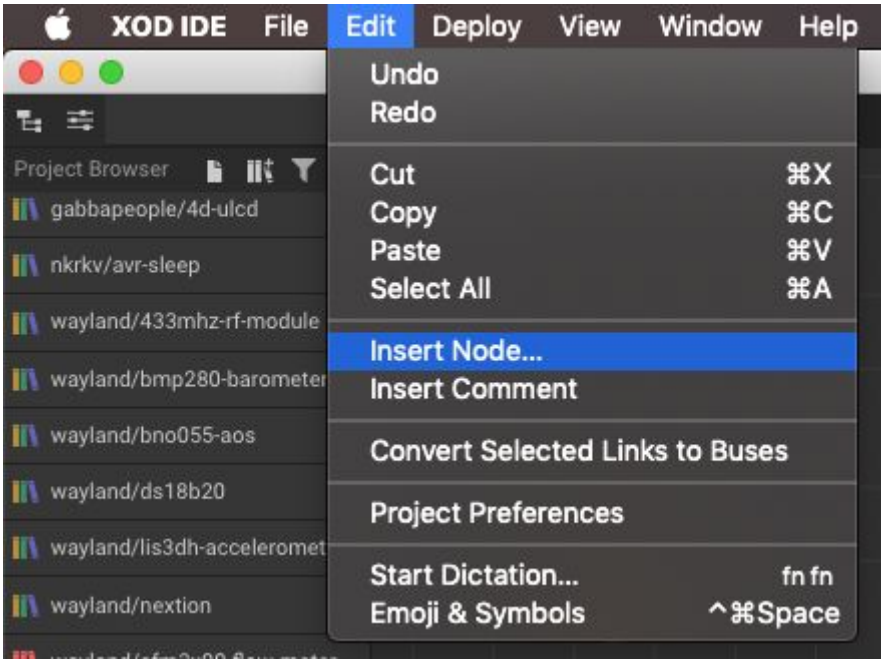
Enter the new name for the patch and hit **Confirm**.



015_rename_node.png

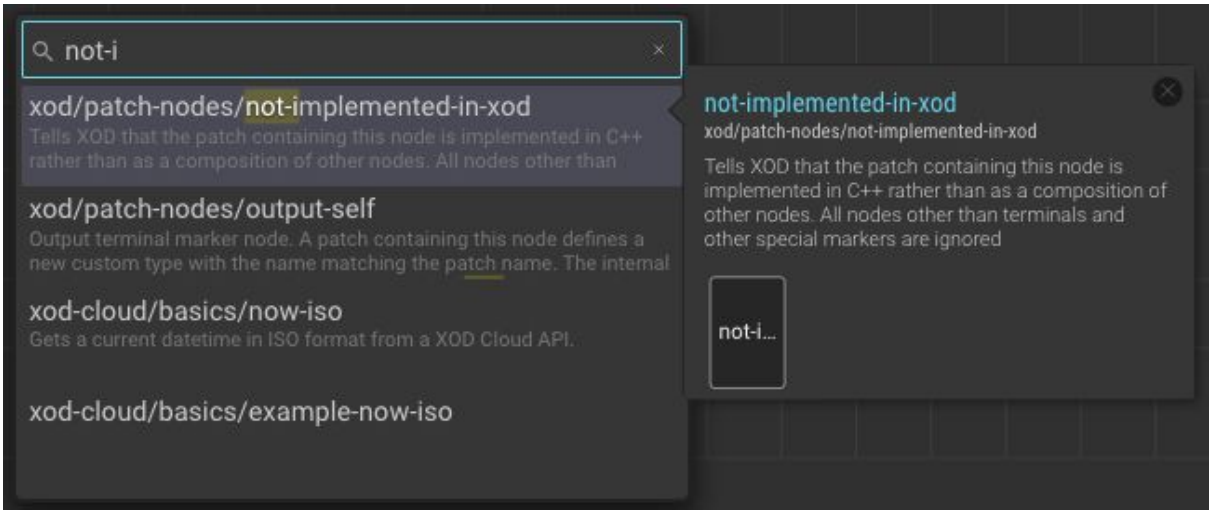
Insert nodes

We are now ready to start adding nodes to our device patch.
Hit **I** or choose:
Edit → Insert Node...



016_insert_node.png

The first node we will add is **not-implemented-in-xod** which will allow us to incorporate C++ code. Start typing the name of this node in the search box and a number of suggestions will appear. Select **xod/patch-nodes/not-implemented-in-xod**.

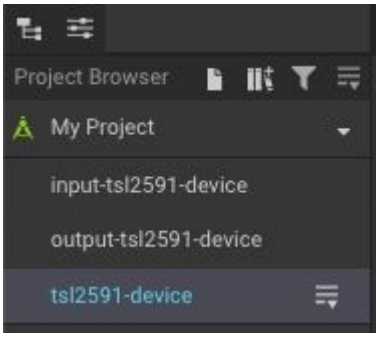


017_not_implemented_in_XOD.png

Next add a **xod/patch-nodes/output-self** node and rename it **DEV**. The name isn't important, but **DEV** is the convention for devices.

018_device_patch.png

After adding the **output-self** node, two new terminal nodes will automatically appear in the **Project Browser**: **input-tsl2591-device** and **output-tsl2591-device**.



019_dev_input_output_nodes.png

Open C++ code editor

Double-click on the **not-implemented-in-XOD** node to open the C++ code editor which contains template code.

tsl2591-device ×

@/tsl2591-device | C++ implementation

```
node {
    // Internal state variables defined at this level persists across evaluations
    Number foo;
    uint8_t bar = 5;

    void evaluate(Context ctx) {
        bar += 42;

        if (isSettingUp()) {
            // This run once
            foo = (Number)(bar + 1);
        }

        auto inValue = getValue<input_IN>(ctx);
        emitValue<output_OUT>(ctx, inValue);
    }
}
```

020_default_code.png

Quick Help provides a C++ Cheatsheet listing the terminal nodes on the patch. In this case there is a single output node. Note that the **output-self** node we named **DEV** on the patch is called **output_DEV** in the C++ code.

021_cpp_cheatsheet.png

Replace the template with the following code:


```
// Tell XOD where it can download the libraries:
#pragma XOD require "https://github.com/adafruit/Adafruit_Sensor"
#pragma XOD require "https://github.com/adafruit/Adafruit_TSL2591_Library"

//Include C++ libraries
#include <Adafruit_Sensor.h>
#include <Adafruit_TSL2591.h>

node {

    meta {
        // Define our custom type as a pointer on the class instance.
        using Type = Adafruit_TSL2591*;
    }

    // Create an object of class Adafruit_TSL2591
    Adafruit_TSL2591 sensor = Adafruit_TSL2591();

    void evaluate(Context ctx) {
        // It should be evaluated only once on the first (setup) transaction
        if (!isSettingUp())
            return;

        // Try to initialize sensor
        if (!sensor.begin()) {
            raiseError(ctx);
            return;
        }
        emitValue<output_DEV>(ctx, &sensor);
    }
}
```

022_device_cpp.png

1. Declare dependencies on the Arduino libraries so that XOD can automatically download and install them.
2. Include the header files of the Arduino libraries.
3. Declare a custom type which describes the hardware module.
4. Create an instance of the custom type.
5. The **evaluate** function is called whenever the node requires updating. The **isSettingUp** function returns true on the first transaction. It is used here to ensure that the initialization code runs once only. The begin function of the Adafruit_TSL2591 class is called to initialize the sensor; if initialization fails an error is raised.
6. Finally an instance of type tsl2591-device is emitted via the patch terminal node **DEV**. N.B. The custom type takes its name from the patch.

Document the device

Document the patch-node and terminal output using the **Description** field on the **Inspector** tab. These descriptions will be made available to users of your library via **Quick Help**.

027_tsl2591-device_description_field.png

028_tsl2591-device_output_description_field.png

Action nodes

The Adafruit_TSL2591 class has several member functions for configuring and reading data from the sensor. We can make these functions available to XOD by wrapping them inside nodes.

Function to be wrapped

Let’s take as an example the function used to set the integration time (the length of time during which the sensing element is collecting charge) of the device. The function is called **setTiming** and takes a single argument, an enumerated type named **tsl2591IntegrationTime_t**.

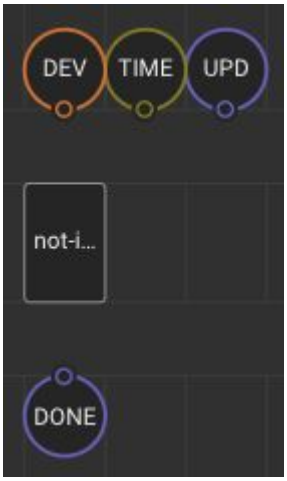
```
86  /// Enumeration for the sensor integration timing
87  typedef enum {
88    TSL2591_INTEGRATIONTIME_100MS = 0x00, // 100 millis
89    TSL2591_INTEGRATIONTIME_200MS = 0x01, // 200 millis
90    TSL2591_INTEGRATIONTIME_300MS = 0x02, // 300 millis
91    TSL2591_INTEGRATIONTIME_400MS = 0x03, // 400 millis
92    TSL2591_INTEGRATIONTIME_500MS = 0x04, // 500 millis
93    TSL2591_INTEGRATIONTIME_600MS = 0x05, // 600 millis
94  } tsl2591IntegrationTime_t;
```

029_integration_time_enum.png

Add a new patch

Follow the convention of starting the names of action nodes with a verb. We'll name this one **set-timing.** Add the following nodes to the patch:

Node	Label	Description
input-tsl2591-device	DEV	A tsl2591-device.
xod/patch-nodes/input-byte	TIME	Integration time (milliseconds). Options: 100ms = 00h, 200ms = 01h, 300ms = 02h, 400ms = 03h, 500ms = 04h, 600ms = 05h.
xod/patch-nodes/input-pulse	UPD	Update
xod/patch-nodes/output-pulse	DONE	Pulse on completion.



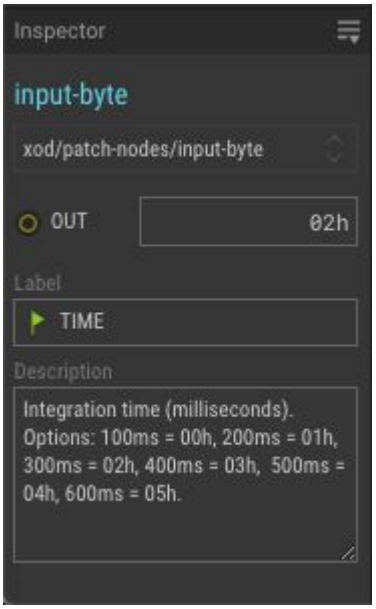
030_set-timing_patch.png

- The input to the **DEV** terminal is a **tsl2591-device** created using our **tsl2591-device** node.
- XOD doesn't have an enum data type, so we'll use a **byte** to specify **TIME** and list the available integration times and their corresponding byte values in the description.

- Pulses received by **UPD** will trigger the action of the node.
- The node will output a pulse from **DONE** when the integration time has been set.

Default values for inputs

We can set default values for node inputs. For example we can set the default integration time to **300ms** by entering **02h** in the **OUT** field of the **TIME** input.



040_default_value_input.png

C++ code

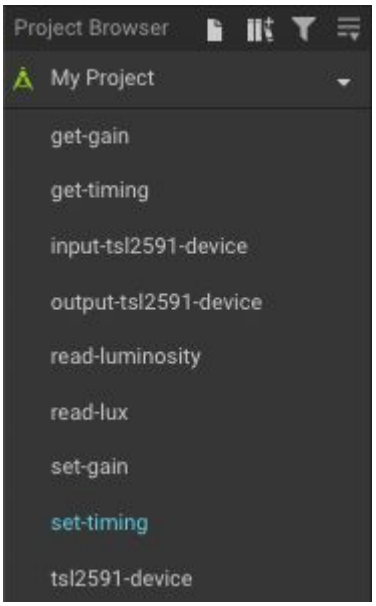
Double-click on the not-implemented-in-xod node to open the C++ editor. Replace the template with the following code. Read comments for an explanation of each line.

```
node {
    void evaluate(Context ctx) {
        // The node responds only if there is an input pulse
        if (!isInputDirty<input_UPD>(ctx))
            return;

        // Get a pointer to the `Adafruit_TSL2591` class instance
        auto sensor = getValue<input_DEV>(ctx);
        // Call the setTiming function passing the input to the
        // TIME terminal as the argument
        sensor -> setTiming(getValue<input_TIME>(ctx));
        // Send a pulse from the DONE terminal
        emitValue<output_DONE>(ctx, 1);
    }
}
```

031_set-timing_cpp_code.png

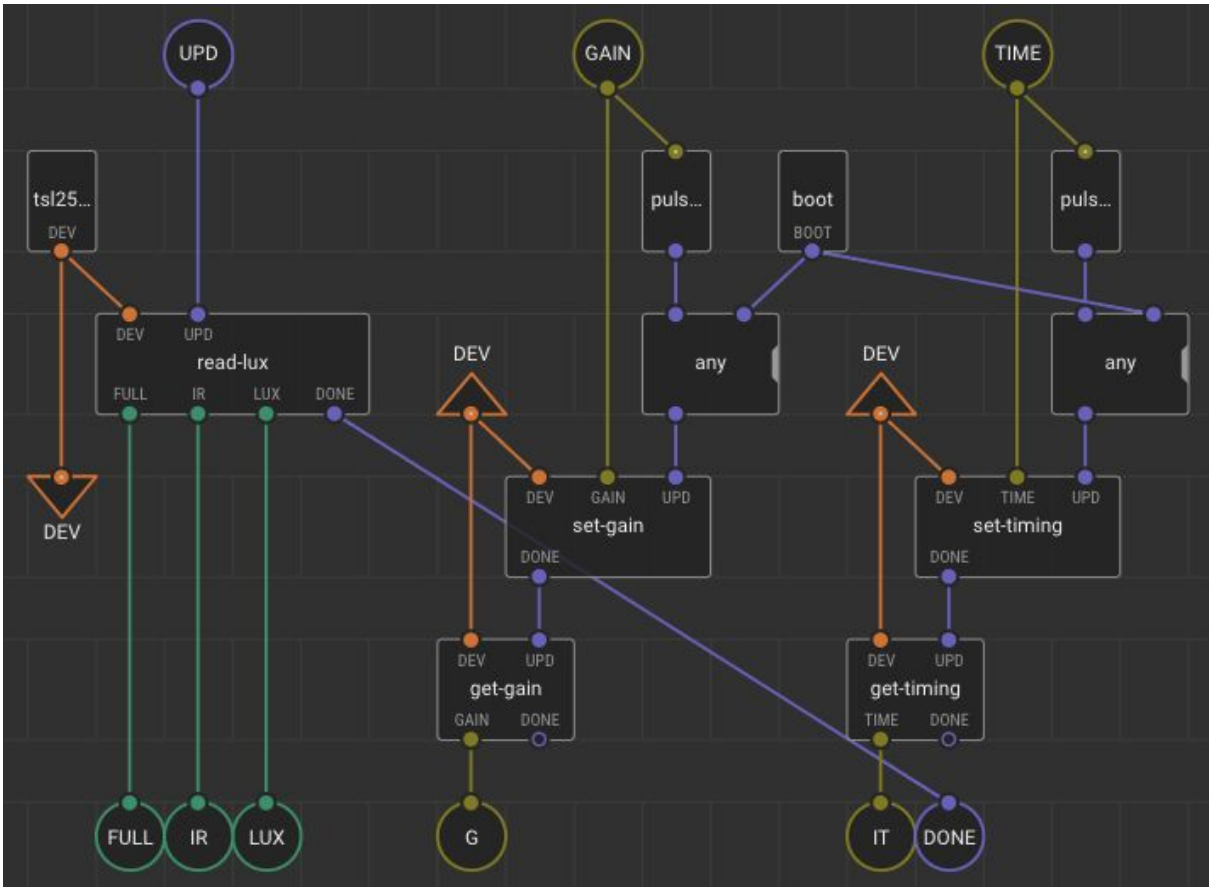
Repeat the process to generate an action node for each of the functions in the Arduino library. If you are unsure how to implement any of the action nodes, please refer to https://xod.io/libs/wayland/tsl2591-light-sensor/.



032_action_nodes.png

Quickstart node

Let’s simplify use of our library by creating a single node that provides all of the functionality a typical user will require. For the TSL2591 sensor, we will assemble a lux meter.



033_lux-meter_patch.png

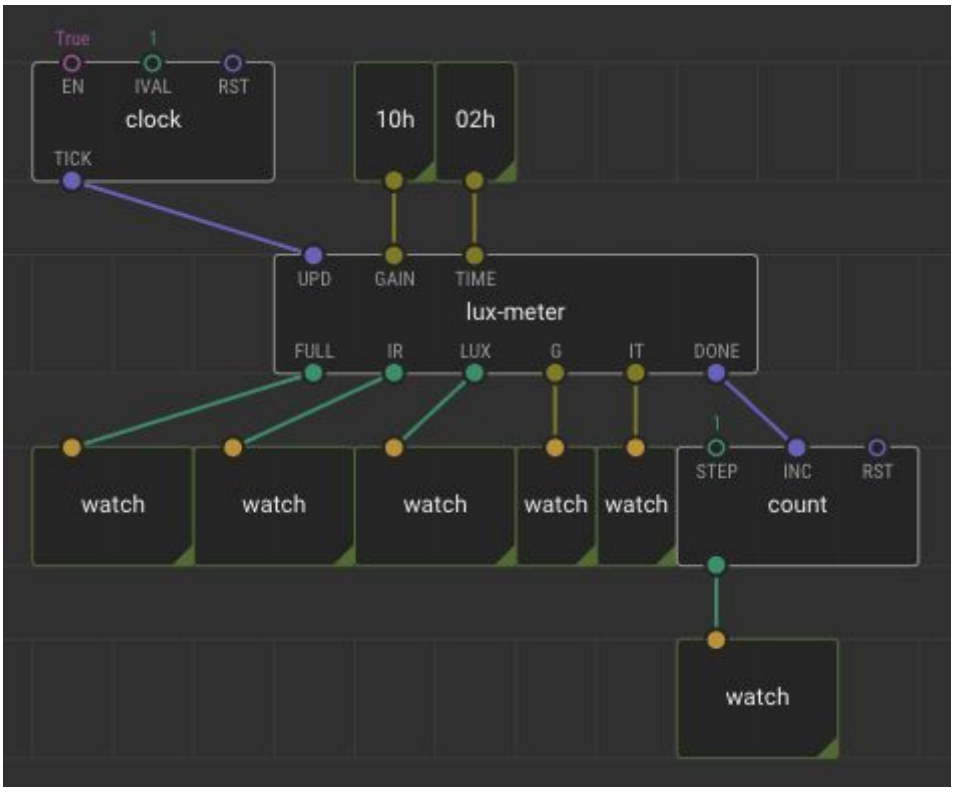
The **read-lux** action node is triggered by a pulse to **UPD** and outputs total luminosity (**FULL**), infrared luminosity (**IR**) and lux (**LUX**). The inputs **GAIN** and **TIME** are used to set sensor gain and integration time respectively. The set-gain and set-timing action nodes are triggered on the initial boot and also whenever the input values change. **Pulse-on-change** nodes (xod/core/pulse-on-change) emit a pulse when the values of their inputs change. The **get-gain** and **get-timing** action nodes report the current sensor gain and integration time respectively.

The finished lux-meter node will look like this:

033_lux-meter_node.png

Example patches

Example patches demonstrate how to use your library and are also invaluable for testing. Here a **clock** node is used to initiate a reading from the sensor every second. **Tweak** nodes allow the user to adjust the gain and integration time at runtime. **Watch** nodes display the values output from the **lux-meter**.

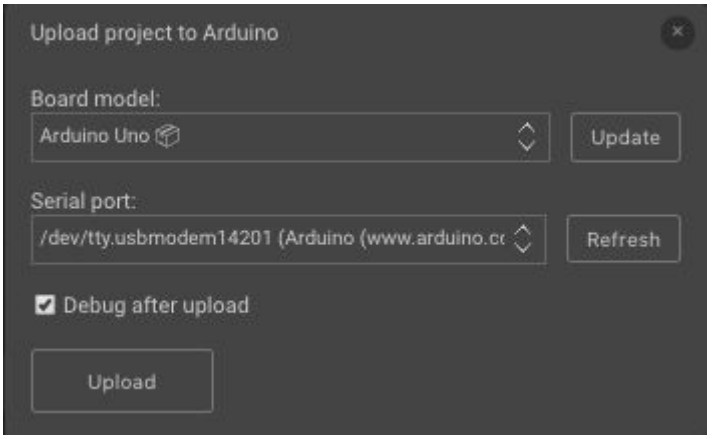


034-example-patch.png

Testing

Upload example patch to Arduino

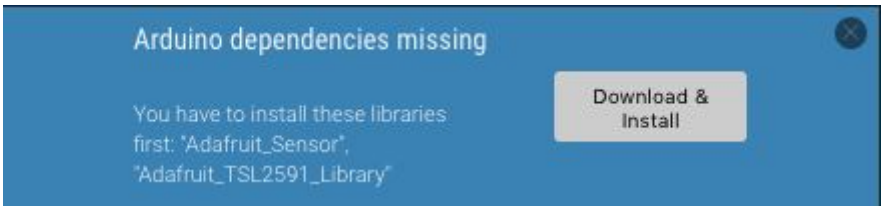
Deploy → Upload to Arduino…



035_upload_project_to_Arduino.png
Since we have **tweak** and **watch** nodes on the example patch, ensure that the **Debug after upload** checkbox is ticked.

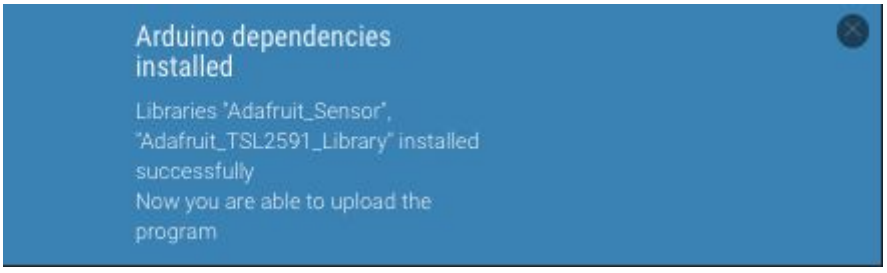
Install dependencies

You will be prompted to install dependencies:



036_arduino_dependencies_missing.png

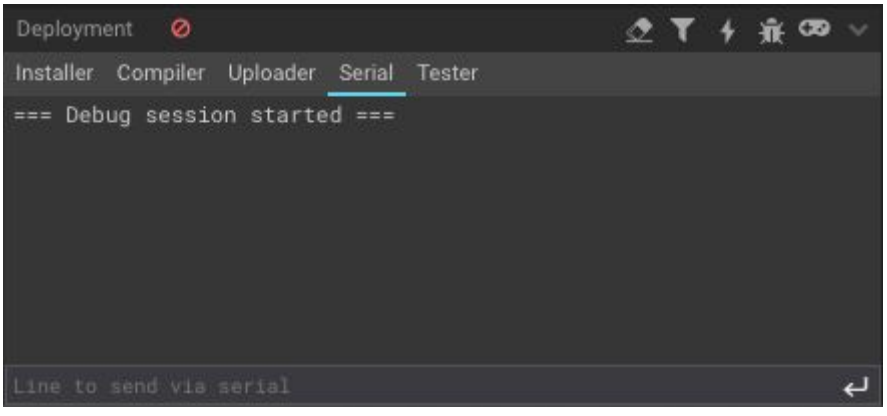
On successful installation you will recipe this message.



037_arduino_dependencies_installed.png

Debugging

Upload the example patch to the Arduino again. Compilation errors will be output on the **Deployment** panel.

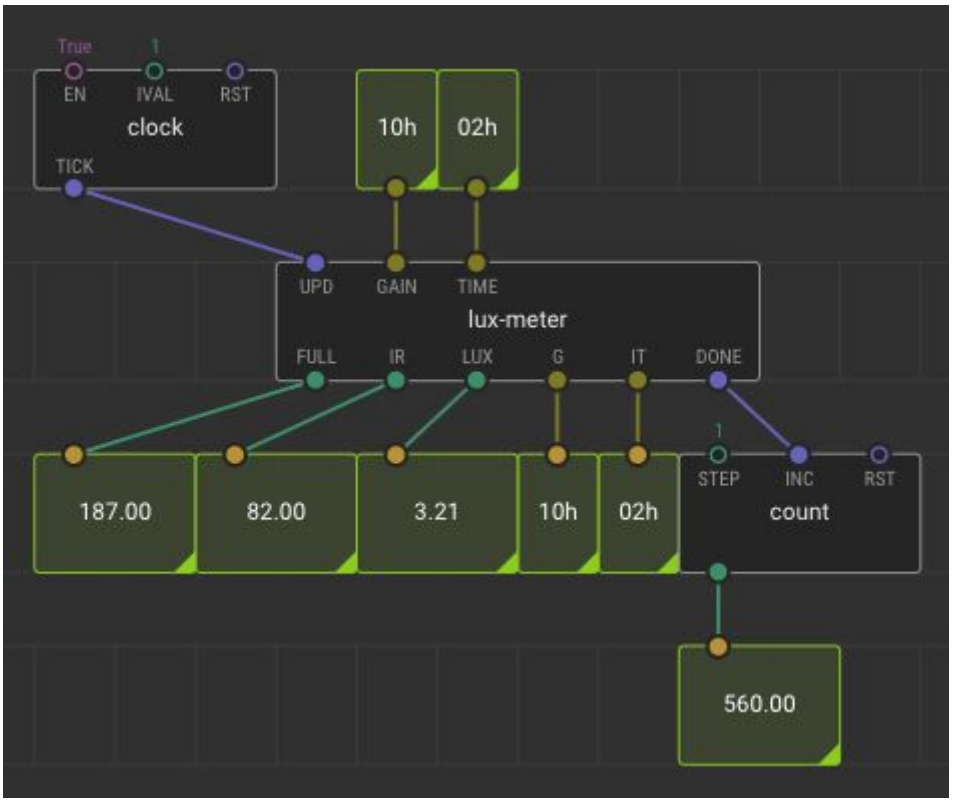


038_deployment_panel.png

Check output

Once the program is running you should see output to all of the watch nodes.

- Are sensible values being reported by all **watch** nodes?
- Try adjusting the gain and integration time of the sensor using the **tweak** nodes.



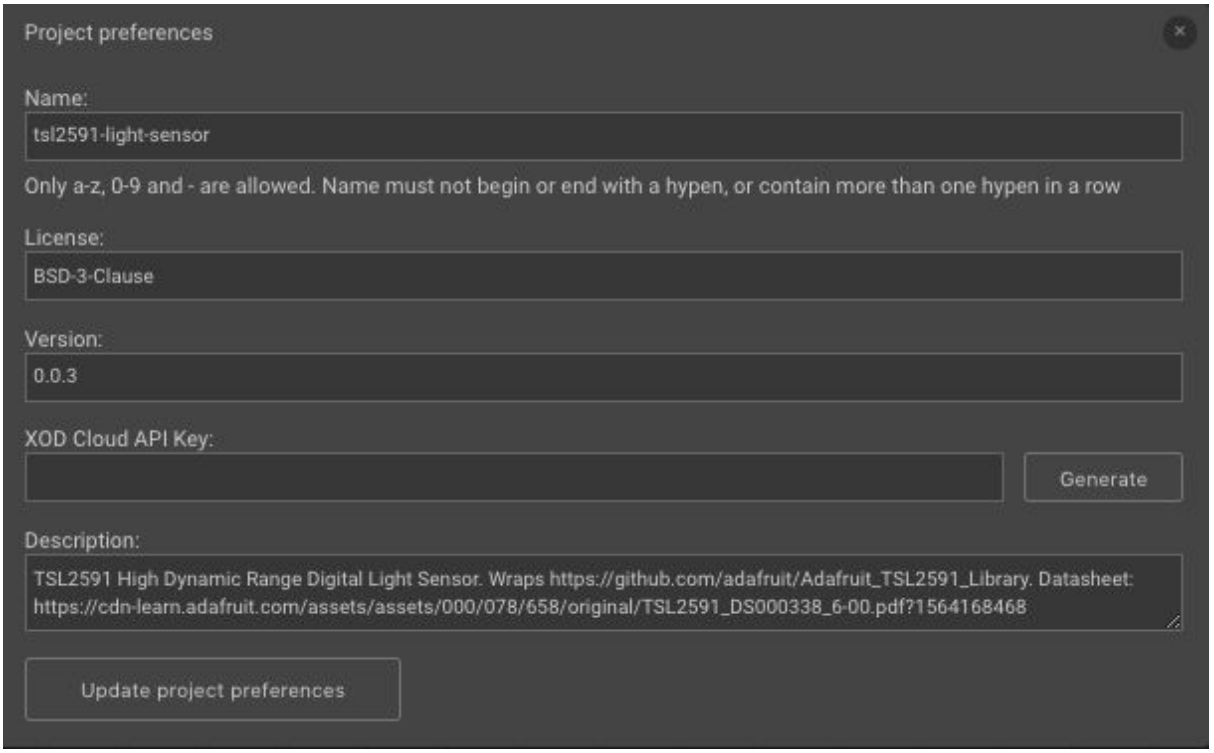
039_running_example_patch.png

Sharing libraries

The process of sharing your library with other xoders is very simple and the XOD IDE provides you with the tools needed.

Set metadata

The first step is to set the metadata for your library.
Edit → Project Preferences

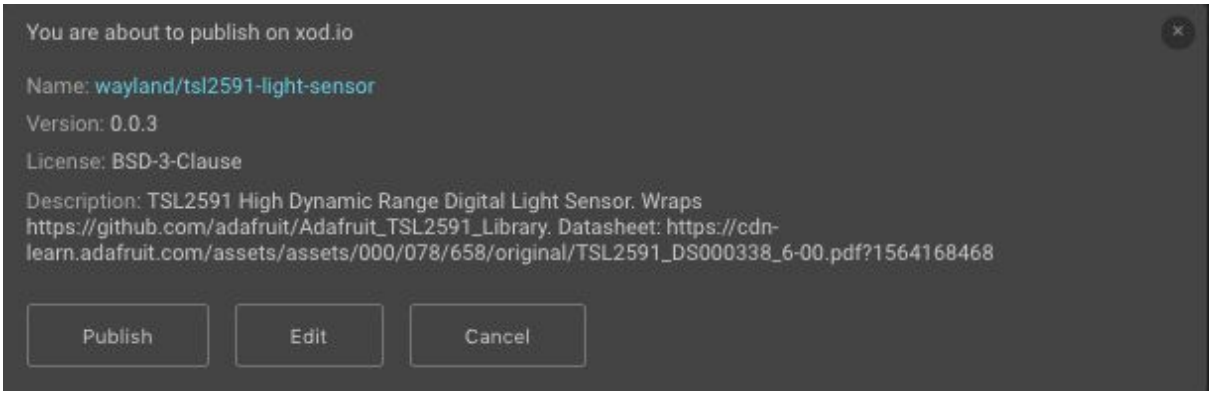


040_default_value_input.png

Name	Short, but descriptive name (max 20 characters).
License	Choose an open source software license.
Version	Semver notation: major.minor.patch
XOD Cloud API Key	Used only for the feeds service provide by **XOD Cloud**
Description	Briefly describe the purpose of the library. You may wish to include a link to the underlying Arduino library and the datasheet for the device.

Publish

When ready to publish, hit:
File → Publish Library…



042_publish.png

Updates

To update your library:

1. Open the library project.
2. Make the required changes.
3. Update the metadata.
4. Publish again.

Summary

The process of wrapping class-based Arduino libraries can be summarized as follows:

1. Find Arduino library for device
2. Test Arduino library
3. Familiarize yourself with the class defined by the library
4. Start a new XOD project
5. Create a new device
6. Wrap class member functions in action nodes
7. Create a quickstart node
8. Create one or more example patches
9. Test library
10. Share library with XOD community

Resources

XOD documentation

XOD has good quality documentation (https://xod.io/docs/). The following guides are particularly relevant:

- Wrapping class-based Arduino libraries: https://xod.io/docs/guide/wrapping-arduino-libraries/

- C++ API: https://xod.io/docs/reference/node-cpp-api/
- Error handling: https://xod.io/docs/guide/errors/
- Dealing with state: https://xod.io/docs/guide/cpp-state/
- Dealing with time: https://xod.io/docs/guide/cpp-time/

XOD forum

XOD has a friendly and helpful community. Don't be afraid to ask for help on the forum: https://forum.xod.io/

Existing XOD libraries

You can learn a lot from looking at existing libraries (https://xod.io/libs/), but be aware that many use an older style of C++ syntax (see https://xod.io/docs/guide/migrating-to-v035/).

Arduino libraries

- https://www.arduinolibraries.info
- https://adafruit.com
- https://www.pololu.com
- https://www.sparkfun.com