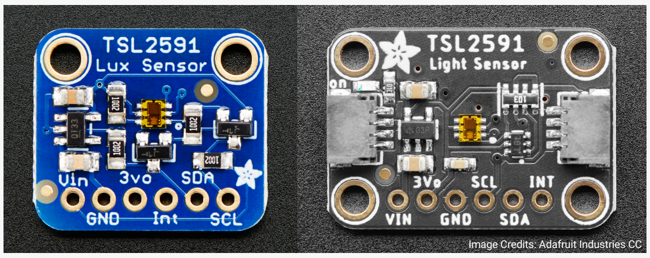
Converting Arduino Libraries to XOD

Why Convert Arduino Libraries?

Whilst the XOD community is growing fast and new libraries are being added all the time, you may find that certain hardware and components do not yet have compatible XOD nodes. In this case, there is almost always an Arduino IDE library that can be used. Arduino libraries exist for a huge range of breakout boards and other devices (see www.arduinolibraries.info). If you have a little C++ experience, it is easy to incorporate these libraries into XOD.

If you cannot find a XOD library for a 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 you will typically find links to code repositories. For more unusual devices a web search will often find libraries developed by hobbyists.



The Adafruit Industries TSL2591 Lux Sensor Breakout Boards. Left to right: solderable and STEMMA-QT versions.

In this tutorial we will create a XOD library for the TSL2591 high dynamic range digital light sensor. Adafruit produce breakout boards for this sensor, available as either a solderable version or with a STEMMA-QT socket (Qwiic-compatible, not Grove-compatible). You can learn more about these breakout boards at learn.adafruit.com/adafruit-tsl2591.

Adafruit's code repository for their TSL2591 library is available on github at www.github.com/adafruit/Adafruit_TSL2591_Library. You should download this library before starting this tutorial.

Please note that this tutorial assumes a basic knowledge of the XOD IDE and C++. For our Beginner's Guide to XOD see www.www.biomaker.org/nocode-programming-for-biology-handbook. For an excellent beginner's short course in C++ see www.codecademy.com/learn/learn-c-plus-plus.

Creating a XOD Library for the TSL2591 Lux Sensor

Requirements

- Computer running MacOS, Windows or Linux with XOD software and USB driver installed (required)
- Adafruit TSL2591 library (required - download from **www.github.com/adafruit/Adafruit_TSL2591_Library**)
- Arduino IDE (for testing - download from **www.arduino.cc/en/Main/Software**)
- Arduino board and USB connector cable (for testing)
- Adafruit TSL2591 Lux Sensor and connectors (for testing)

When presented with a new device the first thing you should do is check if it is already supported in XOD. There is a searchable database of XOD libraries at **www.xod.io/libs**.

If you search for “light sensor” or “TSL2591” you will find that a library already exists for this device (**www.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, and will instead convert the Adafruit C++ library for use in XOD.

It is a good idea to test libraries you find 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.

In this tutorial we will first test the Adafruit TSL2591 library in the Arduino IDE, then ‘wrap’ this library for use in XOD. First we will create a device node to represent the TSL2591 sensor, then we will create action nodes to represent each of the library’s member functions.

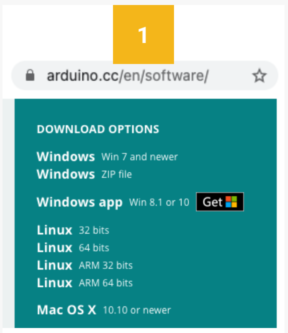
Connecting the TSL2591 Sensor to your Arduino

CONNECTING THE SOLDERABLE SENSOR TO A GROVE BOARD:
- Solder a six pin header set to the breakout board.
- Plug a 4-pin Grove-to-female connector into an I2C socket on the Grove board.
- Fit the male header pins on the breakout board into the female connector ends.
- Make sure the wire colours match the pins as follows: black to GND, red to Vin, white to SDA, yellow to SCL.

CONNECTING THE SOLDERABLE SENSOR TO A DIFFERENT ARDUINO BOARD:
- Solder a six pin header set to the breakout board.
- Use male-to-female wires to connect the pins on the breakout board to the header sockets on the board.
- Make sure the wires are connected as follows: GND to GND, Vin to VIN, SDA to SDA and SCL to SCL.

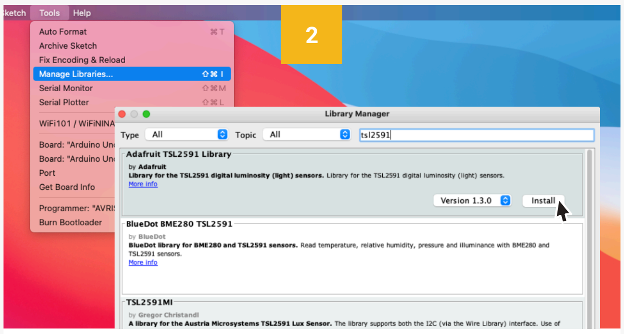
CONNECTING THE STEMMA-QT SENSOR:
- Use a SparkFun Qwiic Arduino board and connect with Qwiic cables.
- Or connect a SparkFun Qwiic shield to any other Arduino board and connect with Qwiic cables.

Testing the Arduino Library



INSTALL ARDUINO IDE

Download the Arduino IDE from the Arduino website and install it on your computer.



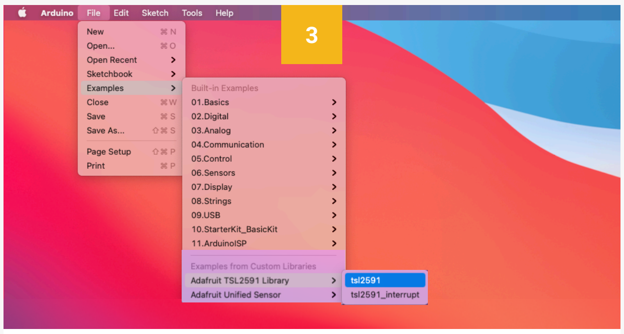
Adafruit TSL2591 Library
by Adafruit
Library for the TSL2591 digital luminosity (light) sensors. Library for the TSL2591 digital luminosity (light) sensors.
More info

BlueDot BME280 TSL2591
by BlueDot
BlueDot library for BME280 and TSL2591 sensors. Read temperature, relative humidity, pressure and illuminance with BME280 and TSL2591 sensors.
More info

TSL2591MI
by Gregor Christandl
A library for the Austria Microsystems TSL2591 Lux Sensor. The library supports both the I2C (via the Wire Library) interface. Use of

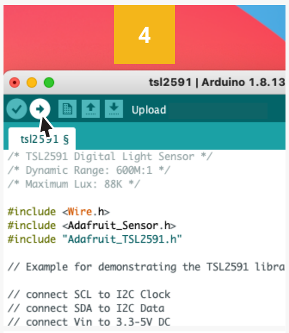
ADD LIBRARY TO IDE

From the 'Tools' menu select 'Manage Libraries'. In the Library Manager search for 'tsl2591'. Select the most recent version of the Adafruit TSL2591 Library and click 'Install'. If you receive a prompt informing you that the library is dependent on other libraries, click 'Install all'.



RUN AN EXAMPLE SKETCH

Running an example sketch is a good way of checking that the device is wired correctly to the Arduino board, that the device is working, and that the library is working. Open an example sketch by navigating to 'File' > 'Examples' > 'Adafruit TSL2591 Library' > 'tsl2591'.

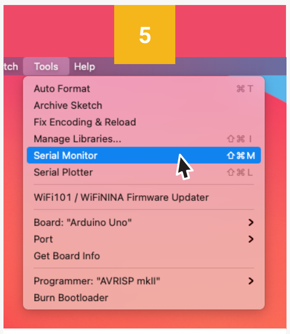


```
/* TSL2591 Digital Light Sensor */
/* Dynamic Range: 600M:1 */
/* Maximum Lux: 88K */

#include <Wire.h>
#include <Adafruit_Sensor.h>
#include "Adafruit_TSL2591.h"

// Example for demonstrating the TSL2591 libra

// connect SCL to I2C Clock
// connect SDA to I2C Data
// connect Vin to 3.3-5V DC
```

UPLOAD

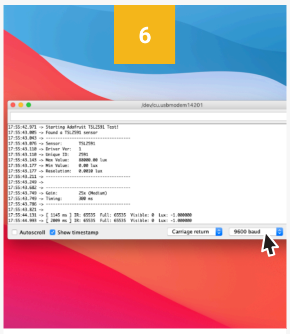
Click on the Upload button. This is the button on the top left of the screen that looks like an arrow.

Testing the Arduino Library



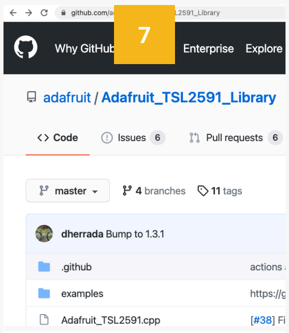
OPEN THE SERIAL MONITOR

Once the program is running, open the serial monitor by navigating to 'Tools' > 'Serial Monitor'.



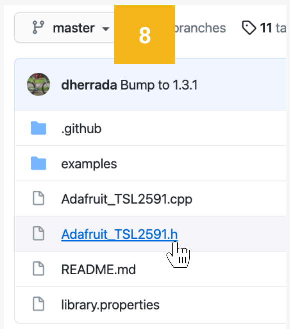
TEST THE SKETCH

Make sure the speed is set to 9600 baud. If everything is working data will be printed to the serial monitor.



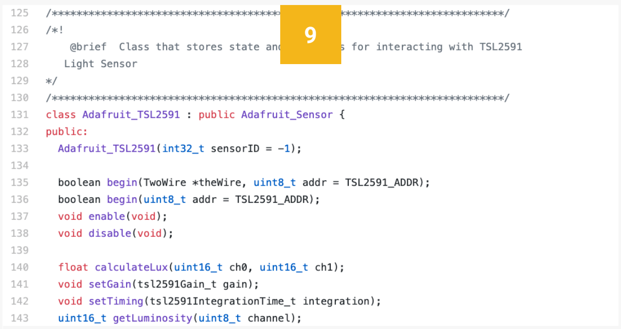
GO TO LIBRARY GITHUB PAGE

In an internet browser, navigate to www.github.com/adafruit/Adafruit_TSL2591_Library.



OPEN THE .H FILE

On Github you can browser the .cpp and .h files which contain the code behind the library. Click on the .h file to open it.

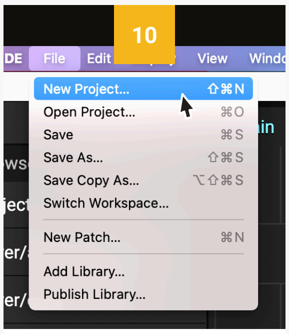


```
/*************************************          **************************/
/*!
    @brief  Class that stores state an[illegible]s for interacting with TSL2591
   Light Sensor
*/
/**************************************************************************/
class Adafruit_TSL2591 : public Adafruit_Sensor {
public:
  Adafruit_TSL2591(int32_t sensorID = -1);

  boolean begin(TwoWire *theWire, uint8_t addr = TSL2591_ADDR);
  boolean begin(uint8_t addr = TSL2591_ADDR);
  void enable(void);
  void disable(void);

  float calculateLux(uint16_t ch0, uint16_t ch1);
  void setGain(tsl2591Gain_t gain);
  void setTiming(tsl2591IntegrationTime_t integration);
  uint16_t getLuminosity(uint8_t channel);
```

EXPLORE THE CODE

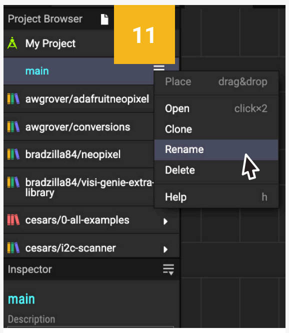
In the .h file find the public interface to the class. This provides the class constructor and various member functions. In XOD we will create a node for the device and then an action node for each of the member functions we want to use. Continue to browse the .cpp and .h files to get a better idea of how the library works.

Creating a TSL2591-Device Node



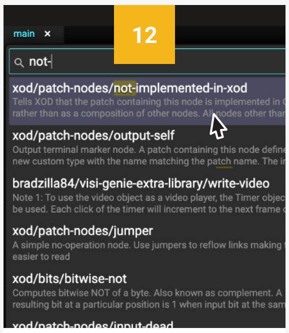
CREATE A NEW PROJECT IN XOD

Open the XOD software and start a new project by navigating to ‘File’ > ‘New Project...’.



RENAME YOUR PATCH

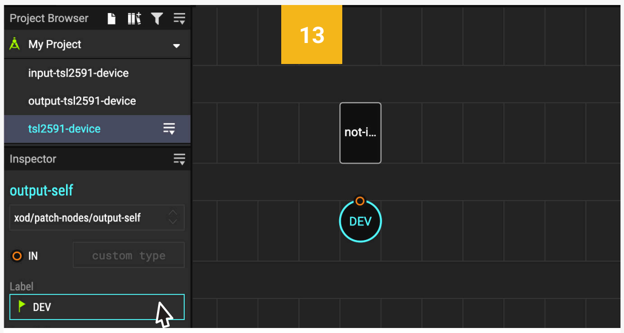
Right-click on the *main* patch in the Project Browser and select ‘Rename’. Name the patch ‘tsl2591-device’.



xod/patch-nodes/not-implemented-in-xod
Tells XOD that the patch containing this node is implemented in C
rather than as a composition of other nodes. All nodes other than
xod/patch-nodes/output-self
Output terminal marker node. A patch containing this node define
new custom type with the name matching the patch name. The i
bradzilla84/visi-genie-extra-library/write-video
Note 1: To use the video object as a video player, the Timer objec
be used. Each click of the timer will increment to the next frame o
xod/patch-nodes/jumper
A simple no-operation node. Use jumpers to reflow links making t
easier to read
xod/bits/bitwise-not
Computes bitwise NOT of a byte. Also known as complement. A
resulting bit at a particular position is 1 when input bit at the sam
xod/patch-nodes/input-dead

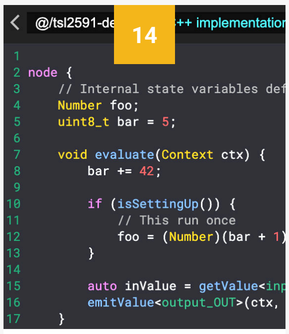
NOT-IMPLEMENTED-IN-XOD NODE

Double click on the patch and type ‘not-implemented-in-xod’. When the node appears, click to add it to the patch.



ADD AN OUTPUT-SELF NODE

Add an *output-self* node (*xod/patch-nodes*) in the same way. Use the ‘Label’ box of the Inspector Pane to name it ‘DEV’. When you do this, you should notice two new patches will automatically appear in the Project Browser: *input-tsl2591-device* and *output-tsl2591-device*.

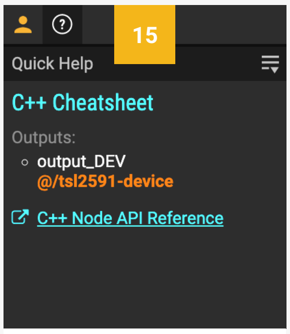


```

node {
    // Internal state variables def
    Number foo;
    uint8_t bar = 5;

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

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

        auto inValue = getValue<inp
        emitValue<output_OUT>(ctx,
    }
```

OPEN C++ CODE EDITOR

Double-click on the *not-implemented-in-xod* node to open the C++ code editor. You will see some template code in the editor.

Creating a TSL2591-Device Node



C++ Cheatsheet

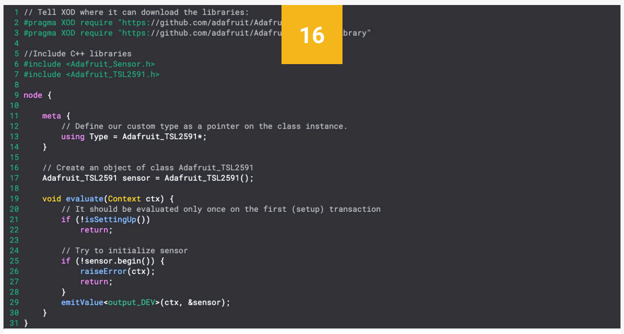
Outputs:

- output_DEV
  @/tsl2591-device

C++ Node API Reference

QUICK HELP

If you look at the Quick Help pane there is a C++ Cheatsheet listing terminal nodes in the patch. Here there is a single output node.



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

//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);
    }
}
```

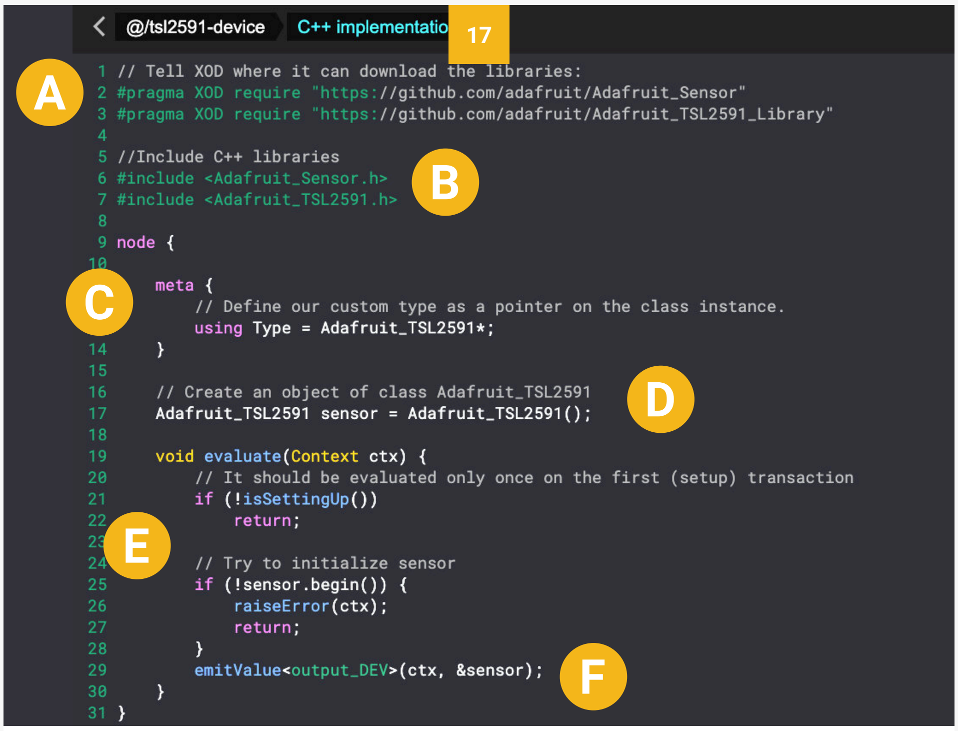
ADD CODE

Delete the template code and add the prepared C++ code (download from the Biomaker website, or copy from below).

C++ CODE FOR XOD TSL2591 DEVICE

```
// 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);
  }
}
```

The TSL2591-Device C++ Code

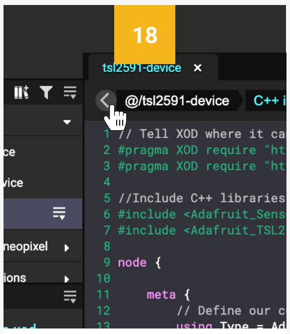


```cpp
// Tell XOD where it can download the libraries:
#pragma XOD require "https://github.com/adafruit/Adafruit_Sensor"
#pragma XOD require "https://github.com/adafruit/Adafruit_TSL2591_Library"

//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);
    }
}
```

UNDERSTAND EACH SECTION OF THE C++ CODE

A	Declare dependencies on the Arduino libraries so that XOD can automatically download and install them.
B	Include the header files of the Arduino libraries.
C	Declare a custom type which describes the hardware module.
D	Create an instance of the custom type.
E	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 initialisation code runs once only. The begin function of the Adafruit_TSL2591 class is called to initialise the sensor; if initialisation fails an error is raised.
F	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.

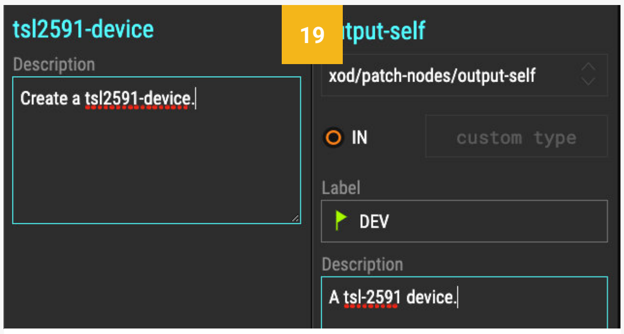
Documenting Your Device Node and Introduction to Action Nodes


18

```
// Tell XOD where it ca
#pragma XOD require "ht
#pragma XOD require "ht

//Include C++ libraries
#include <Adafruit_Sens
#include <Adafruit_TSL2

node {

    meta {
        // Define our c
        using Type = Ad
```

RETURN TO THE XOD PATCH

You can return to the XOD patch at ant time by clicking the back arrow in the top left of the patch.


19

tsl2591-device

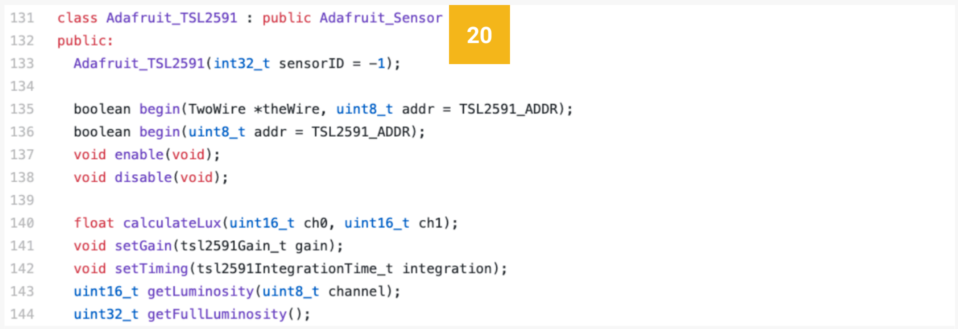
Description: Create a tsl2591-device.

Label: DEV

Description: A tsl-2591 device.

DESCRIBE YOUR NODE

Click on an empty space in the XOD patch, then use the description box in the Inspector pane to write a short description of your new node. You can also add descriptions for each node within your patch by clicking on them. This documentation is important as it will help others to understand and use your libraries.

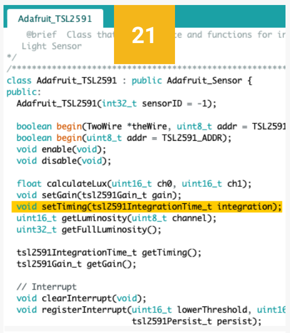

20

```
class Adafruit_TSL2591 : public Adafruit_Sensor
public:
  Adafruit_TSL2591(int32_t sensorID = -1);

  boolean begin(TwoWire *theWire, uint8_t addr = TSL2591_ADDR);
  boolean begin(uint8_t addr = TSL2591_ADDR);
  void enable(void);
  void disable(void);

  float calculateLux(uint16_t ch0, uint16_t ch1);
  void setGain(tsl2591Gain_t gain);
  void setTiming(tsl2591IntegrationTime_t integration);
  uint16_t getLuminosity(uint8_t channel);
  uint32_t getFullLuminosity();
```

ACTION NODES

Now that we have a node to represent our device, we also need action nodes to initiate actions or sequences from the Arduino library. In the header file, you can see that 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. As an example we'll use the function to set the integration time (the length of time the sensing element is collecting charge) of the device.

Creating a Set-Timing Node

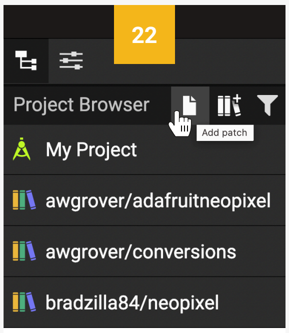


```
Adafruit_TSL2591
    @brief  Class that    e and functions for ir
   Light Sensor
*/
/*******************************************************
class Adafruit_TSL2591 : public Adafruit_Sensor {
public:
  Adafruit_TSL2591(int32_t sensorID = -1);

  boolean begin(TwoWire *theWire, uint8_t addr = TSL2591
  boolean begin(uint8_t addr = TSL2591_ADDR);
  void enable(void);
  void disable(void);

  float calculateLux(uint16_t ch0, uint16_t ch1);
  void setGain(tsl2591Gain_t gain);
  void setTiming(tsl2591IntegrationTime_t integration);
  uint16_t getLuminosity(uint8_t channel);
  uint32_t getFullLuminosity();

  tsl2591IntegrationTime_t getTiming();
  tsl2591Gain_t getGain();

  // Interrupt
  void clearInterrupt(void);
  void registerInterrupt(uint16_t lowerThreshold, uint16
                         tsl2591Persist_t persist);
```

21

THE INTEGRATION TIME FUNCTION

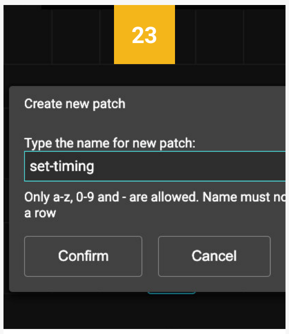
This function is called *setTiming* and takes one argument: *tsl2591IntegrationTime_t*, which is an enumerated type.



22

MAKE A NEW PATCH

Add a new patch to XOD. Click the 'Add patch' button in the Project Browser or select 'File > New Patch...' in the menu.



Create new patch

Type the name for new patch:

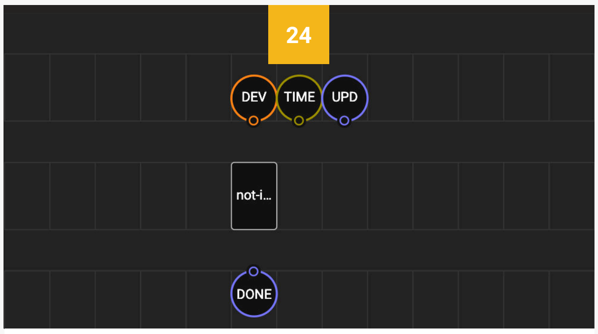
set-timing

Only a-z, 0-9 and - are allowed. Name must no
a row

23

NAME THE PATCH

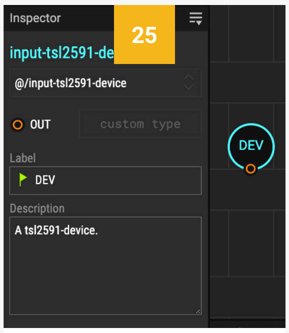
Following the convention of starting the names of action nodes with a verb. We'll name this one 'set-timing'.



24

ADD NODES

Add the following nodes to your *set-timing* patch: *input-tsl2591-device* (your patch), *input-byte*, *input-pulse*, *output-pulse*, *not-implemented-in-xod* (*xod/patch-nodes*). We will give them names, labels, and explain their purposes in the following steps.



Inspector

input-tsl2591-de

@/input-tsl2591-device

OUT custom type

Label: DEV

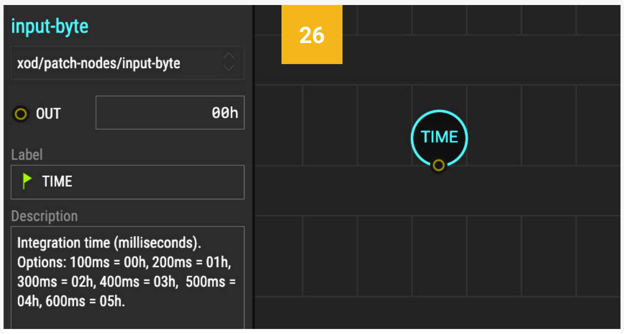
Description: A tsl2591-device.

25

INPUT-TSL2591-DEVICE

Name this node 'DEV' with description 'A tsl2591-device'. This is a tsl2591-device created using our tsl2591-device node.

Creating a Set-Timing Node


26

input-byte

xod/patch-nodes/input-byte

OUT 00h

Label

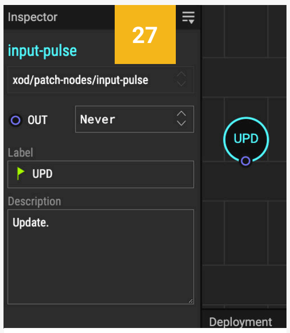
TIME

Description

Integration time (milliseconds). Options: 100ms = 00h, 200ms = 01h, 300ms = 02h, 400ms = 03h, 500ms = 04h, 600ms = 05h.

INPUT-BYTE

Name this node ‘TIME’ with description ‘Integration time (milliseconds). Options: 100ms = 00h, 200ms = 01h, 300ms = 02h, 400ms = 03h, 500ms = 04h, 600ms = 05h’. There’s no enum data type in XOD, so we’ll use a byte to specify TIME and list the available integration times and their byte values in the description.


27

Inspector

input-pulse

xod/patch-nodes/input-pulse

OUT Never

Label

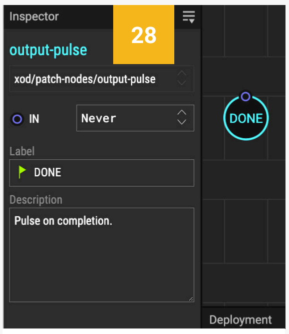
UPD

Description

Update.

INPUT-PULSE

Name this node ‘UPD’ with description ‘Update’. Pulses received by UPD will trigger the action of the node.


28

Inspector

output-pulse

xod/patch-nodes/output-pulse

IN Never

Label

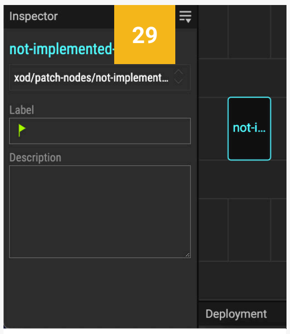
DONE

Description

Pulse on completion.

OUTPUT-PULSE

Name this ‘DONE’ with description ‘Pulse on completion’. This node will output a pulse when the integration time has been set.


29

Inspector

not-implemented-

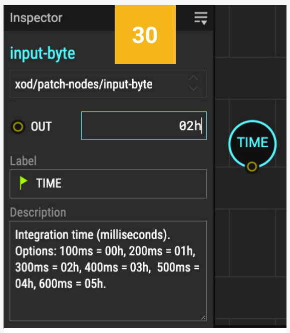
xod/patch-nodes/not-implement...

Label

Description

NOT-IMPLEMENTED-IN-XOD

We will use this node to add C++ code linking the XOD patch to the Arduino library.


30

Inspector

input-byte

xod/patch-nodes/input-byte

OUT 02h

Label

TIME

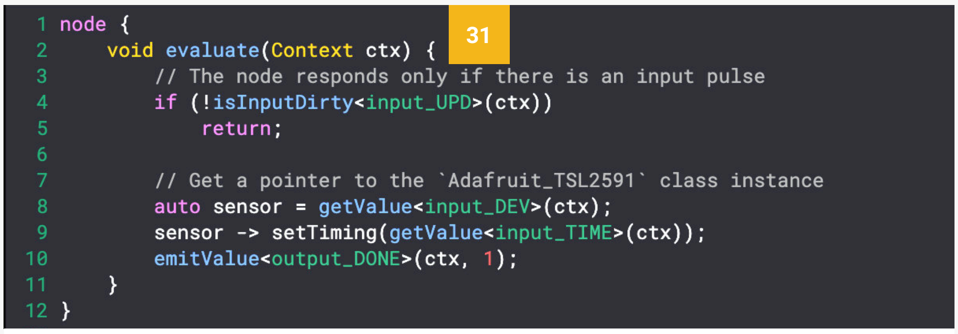
Description

Integration time (milliseconds). Options: 100ms = 00h, 200ms = 01h, 300ms = 02h, 400ms = 03h, 500ms = 04h, 600ms = 05h.

DEFAULT VALUES

We can set default values for inputs. E.g. set default integration time to 300ms using 02h in the OUT field of the TIME input.

The Set-Timing C++ Code



31

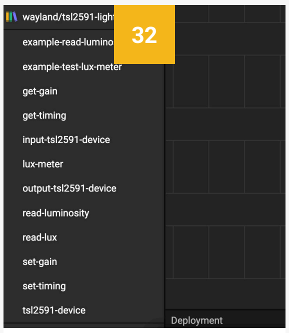
```
node {
    void evaluate(Context ctx) {
        // The node responds only if there is an input pulse
        if (!isInputDirty<input_UPD>(ctx))
            return;

        // Get a pointer to the `Adafruit_TSL2591` class instance
        auto sensor = getValue<input_DEV>(ctx);
        sensor -> setTiming(getValue<input_TIME>(ctx));
        emitValue<output_DONE>(ctx, 1);
    }
}
```

REPLACE THE TEMPLATE WITH CODE

Double-click on the not-implemented-in-xod node to open the C++ editor. Replace the template with the code below (download from the Biomaker website or copy from below). Read the comments for an explanation of each line.

C++ CODE FOR XOD SET-TIMING NODE

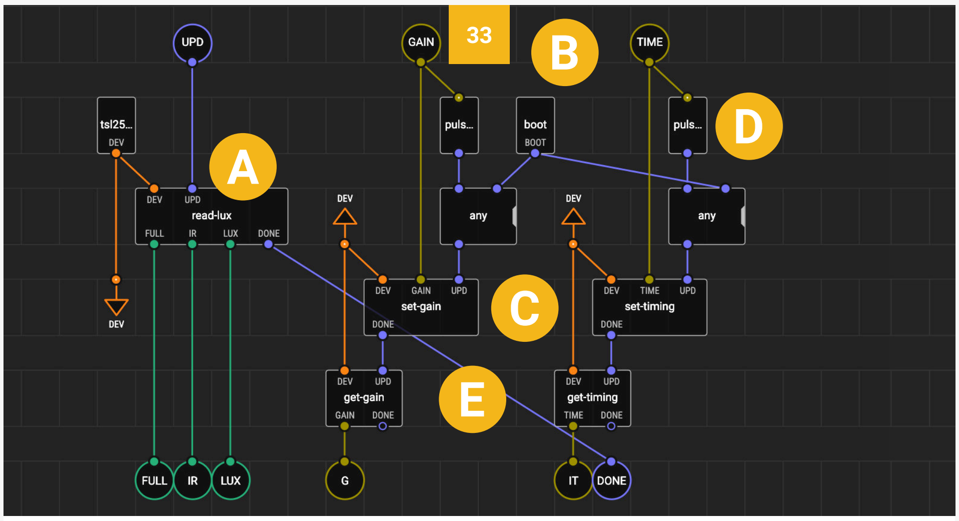
```
node {
   void evaluate(Context ctx) {
      // The node responds only if there is an input pulse
      if (!isInputDirty<input_UPD>(ctx))
         return;

      // Get a pointer to the `Adafruit_TSL2591` class
instance
      auto sensor = getValue<input_DEV>(ctx);
      sensor -> setTiming(getValue<input_TIME>(ctx));
      emitValue<output_DONE>(ctx, 1);
   }
}
```



REPEAT FOR EACH FUNCTION

Repeat the process for each of the functions in the Arduino library. Use the *wayland/tsl2591-light-sensor* library as a reference.

Creating a Quick-Start Node



CREATE A QUICK-START NODE

Let's simplify use of our library by creating a single node with all the functionality a typical user requires. For the TSL2591 sensor, we will assemble a lux meter.

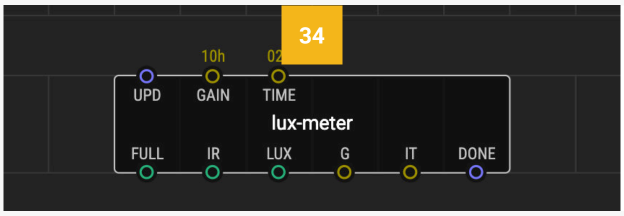
A The *read-lux* action node is triggered by a pulse to UPD and outputs total luminosity (FULL), infrared luminosity (IR) and lux (LUX).

B The inputs GAIN and TIME are used to set sensor gain and integration time respectively.

C The *set-gain* and *set-timing* action nodes are triggered on the initial boot and also whenever the input values change.

D The *pulse-on-change* nodes (*xod/core*) emit a pulse when the values of their inputs change.

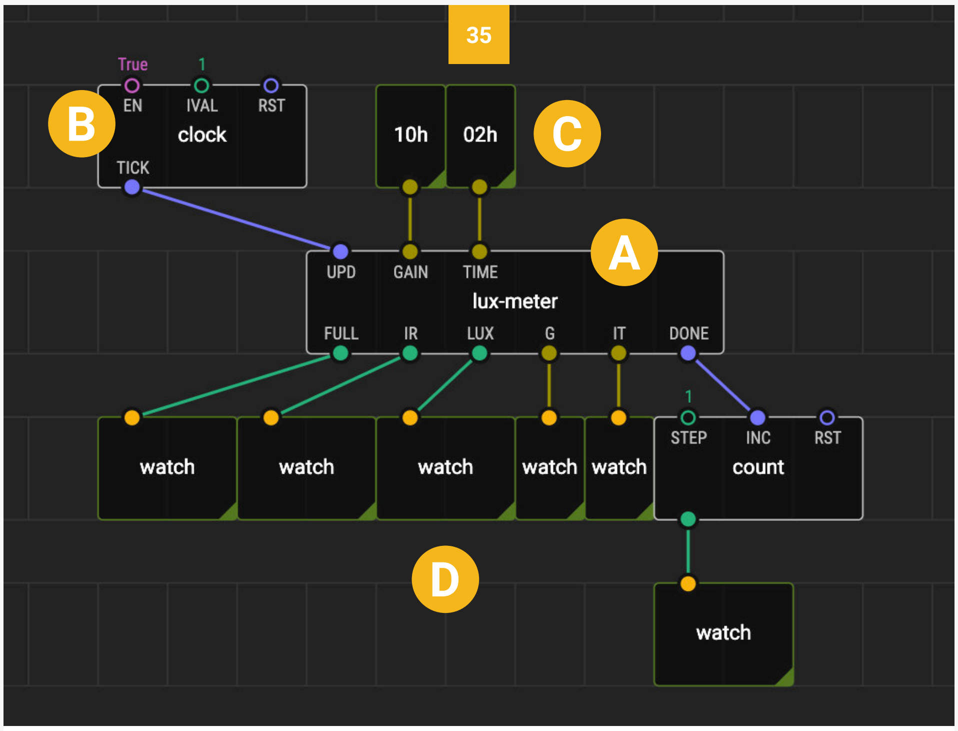
E The *get-gain* and *get-timing* action nodes report the current sensor gain and integration time respectively.



LUX-METER NODE

The finished lux-meter node will look like this.

Creating Example Patches and Testing



MAKE AN EXAMPLE PATCH AND TEST YOUR PATCH

Example patches demonstrate how to use your library and are also invaluable for testing. This example patch shows how our newly-created *lux-meter* node can be used.

A The *lux-meter* is our quick-start node which encompasses our device node, as well as several action nodes, to take readings from the sensor.

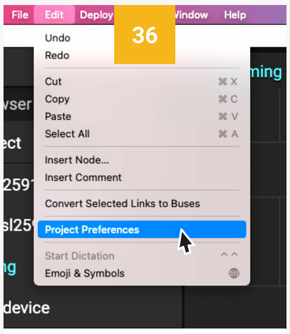
B A clock node is used to initiate a reading from the sensor every second.

C Tweak nodes allow the user to adjust the gain and integration time at runtime.

D Watch nodes display the values output from the lux-meter.

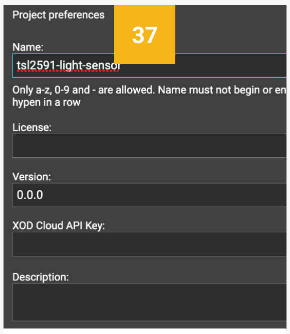
Once finished you should use your example patch to test your nodes. Use 'Upload and Debug' to upload the patch to your Arduino, installing dependencies if you need to. Once running you should see output to all watch nodes. Check whether the values being reported by watch nodes are sensible, and try adjusting the gain and integration time.

Publishing Your Library



OPEN PROJECT PREFERENCES

The first step to publish your library is to set the metadata. Go to 'Edit' > 'Project Preferences...' in the menu bar.



Project preferences

Name:
tsl2591-light-sensor

Only a-z, 0-9 and - are allowed. Name must not begin or en
hypen in a row

License:

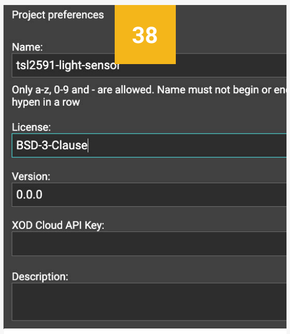
Version:
0.0.0

XOD Cloud API Key:

Description:

SET METADATA: NAME

Use this window to set your library's metadata. Under 'Name' add a short, but descriptive name (max 20 characters).



Project preferences

Name:
tsl2591-light-sensor

Only a-z, 0-9 and - are allowed. Name must not begin or en
hypen in a row

License:
BSD-3-Clause

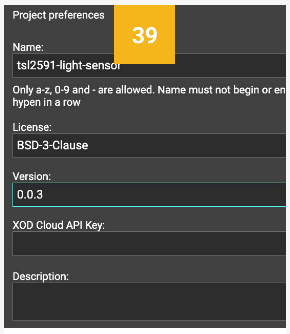
Version:
0.0.0

XOD Cloud API Key:

Description:

SET METADATA: LICENCE

Under 'Licence' choose an open source software license (see **www.opensource.org/licenses**).



Project preferences

Name:
tsl2591-light-sensor

Only a-z, 0-9 and - are allowed. Name must not begin or en
hypen in a row

License:
BSD-3-Clause

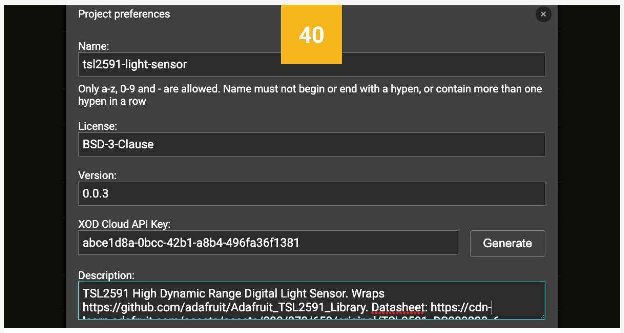
Version:
0.0.3

XOD Cloud API Key:

Description:

SET METADATA: VERSION

Under 'Version' set the version number using Semver notation, i.e. major.minor.patch.



Project preferences

Name:
tsl2591-light-sensor

Only a-z, 0-9 and - are allowed. Name must not begin or end with a hypen, or contain more than one hypen in a row

License:
BSD-3-Clause

Version:
0.0.3

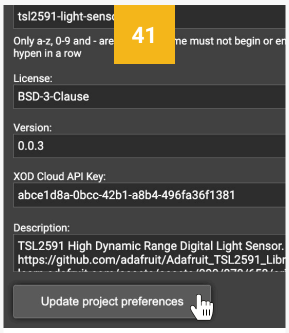
XOD Cloud API Key:
abce1d8a-0bcc-42b1-a8b4-496fa36f1381

Description:
TSL2591 High Dynamic Range Digital Light Sensor. Wraps https://github.com/adafruit/Adafruit_TSL2591_Library. Datasheet: https://cdn-

SET METADATA: DESCRIPTION

Briefly describe the purpose of the library. You may wish to include a link to the underlying Arduino library and the data sheet for the device. This information will be useful for anyone wishing to use your library.

Publishing Your Library



tsl2591-light-sens
Only a-z, 0-9 and - are ne must not begin or en
hypen in a row

License:
BSD-3-Clause

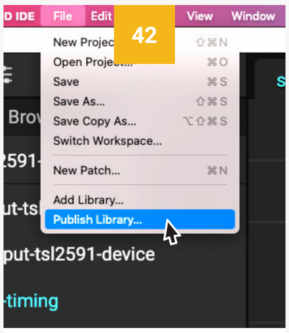
Version:
0.0.3

XOD Cloud API Key:
abce1d8a-0bcc-42b1-a8b4-496fa36f1381

Description:
TSL2591 High Dynamic Range Digital Light Sensor.
https://github.com/adafruit/Adafruit_TSL2591_Libr

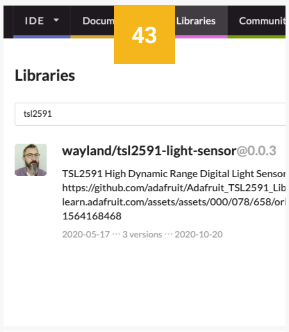
UPDATE PROJECT PREFERENCES

Click the 'Update Project Preferences' button to save your changes.



PUBLISH LIBRARY

When ready to publish, go to 'File' > 'Publish Library...'. A window will summarise the metadata. Click 'Publish' to finalise.



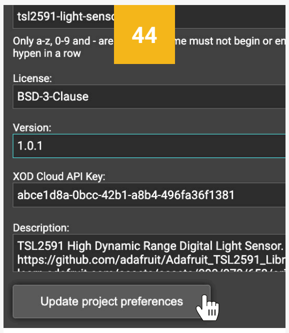
Libraries

tsl2591

wayland/tsl2591-light-sensor@0.0.3
TSL2591 High Dynamic Range Digital Light Sensor
https://github.com/adafruit/Adafruit_TSL2591_Lib
learn.adafruit.com/assets/assets/000/078/658/ori
1564168468
2020-05-17 ··· 3 versions ··· 2020-10-20

XOD LIBRARY DATABASE

Your library will now appear in the XOD database (www.xod.io/libs), available for other users to download.



tsl2591-light-sens
Only a-z, 0-9 and - are ne must not begin or en
hypen in a row

License:
BSD-3-Clause

Version:
1.0.1

XOD Cloud API Key:
abce1d8a-0bcc-42b1-a8b4-496fa36f1381

Description:
TSL2591 High Dynamic Range Digital Light Sensor.
https://github.com/adafruit/Adafruit_TSL2591_Libr

UPDATES

To update your library:
- Open the project.
- Make changes.
- Update metadata.
- Publish again.

Summary

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

1. Find Arduino library for device
2. Test Arduino library
3. Familiarise 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 quick-start node
8. Create one or more example patches
9. Test library
10. Share library with XOD community

Useful Resources

XOD Resources

XOD DOCUMENTATION
XOD has good quality documentation for a range of projects available at **www.xod.io/docs**. The following guides are particularly relevant for this tutorial:

- Wrapping class-based Arduino libraries: **www.xod.io/docs/guide/wrapping-arduino-libraries**
- C++ API: **www.xod.io/docs/reference/node-cpp-api**
- Error handling: **www.xod.io/docs/guide/errors**
- Dealing with state: **www.xod.io/docs/guide/cpp-state**
- Dealing with time: **www.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 at **www.forum.xod.io**

XOD LIBRARIES
You can learn a lot from looking at existing libraries at **www.xod.io/libs**. However, you should be aware that many use an older style of C++ syntax. See **www.xod.io/docs/guide/migrating-to-v035** for more details.

Arduino Libraries

The following are good locations to search for relevant class-based Arduino libraries:

- Arduino: **www.arduinolibraries.info**
- Adafruit: **www.adafruit.com**
- Pololu: **www.pololu.com**
- Sparkfun: **www.sparkfun.com**

ACKNOWLEDGEMENTS

Author: Matt Wayland
Images: Stephanie Norwood, Adafruit Industries, LLC
Design: Stephanie Norwood
Hardware: Arduino S.r.l., Adafruit Industries, LLC
Software: Arduino S.r.l., XOD, Inc
XOD library creators: Matt Wayland
Funders and Sponsors: BBSRC, EPSRC, NERC, OpenPlant, SynBio IRC, NSF
Contacts: Stephanie Norwood (**synbio@hermes.cam.ac.uk**), Jim Haseloff (**jh295@cam.ac.uk**)