



The Well-Automated Arduino Library

Created by lady ada



<https://learn.adafruit.com/the-well-automated-arduino-library>

Last updated on 2024-06-03 02:17:24 PM EDT

Table of Contents

Overview	3
Repository	4
<ul style="list-style-type: none">• Overall Structure• Create Empty gh-pages Branch• Enable GitHub Pages on gh-pages• Optional: .gitignore• Optional (For Adafruit Only!) Add Adafruit Librarians group	
Doxygen	9
<ul style="list-style-type: none">• About Doxygen• Step 1) Install Doxygen• Step 2) Grab our Doxyfile• Step 3) Run doxygen• Preview Documentation• Commit Documented Code	
Doxygen Tips	15
<ul style="list-style-type: none">• Main .cpp File Header• Main .h File Header• Documenting Functions• Parameters• Macros and Enums• Classes & Objects• Optional: Code Snippets and Fixed Width Text Blocks	
Travis CI	20
<ul style="list-style-type: none">• Sign up for Travis• Travis YML• Add travis.yml To Library• Checking Status• Failure/Passes• Adding Write Permissions• Final Run and Check!	
Formatting with clang-format	29
<ul style="list-style-type: none">• About ClangFormat	

Overview



Writing software is like gardening - the flowers are beautiful but you're going to spend a lot of time weeding! Except instead of weeding, it's keeping up to date with new frameworks, operating systems and dependencies. It's a ton of work! So why not make it easy on your self with automation?

In this guide we'll go through how to use [GitHub](https://adafru.it/d6C) (a free service for storing source code) + [Doxygen](https://adafru.it/BRL) (a free documentation standard/parser) + [TravisCI.org](https://adafru.it/BRM) (a free service for continuous integration/testing) To automatically document and test your code.

In specific, we'll be showing how to automate wide-range compilation tests for an Arduino library to make sure that updates to the Arduino IDE, language, and board support packages don't end up breaking builds.

In general, the same steps can be used for any codebase, including every-day source code projects - Arduino and otherwise! (But, it's really good for Arduino because we have scripts that do a lot of work for you already written)

This guide doesn't teach you how to write code, or how to write a library. But it will show you how to make your existing libraries live long fruitful lives as they age!

Hey, there's paid services that do this for you!

Yep, this guide is how to do it using only free services for open-source software

Hey, there's other *better* documentation systems and CI services!

Yep, this guide is how to do it using only the techniques I know how to use

Hey, you can do this yourself using hand-built artisanal servers and crontabs!

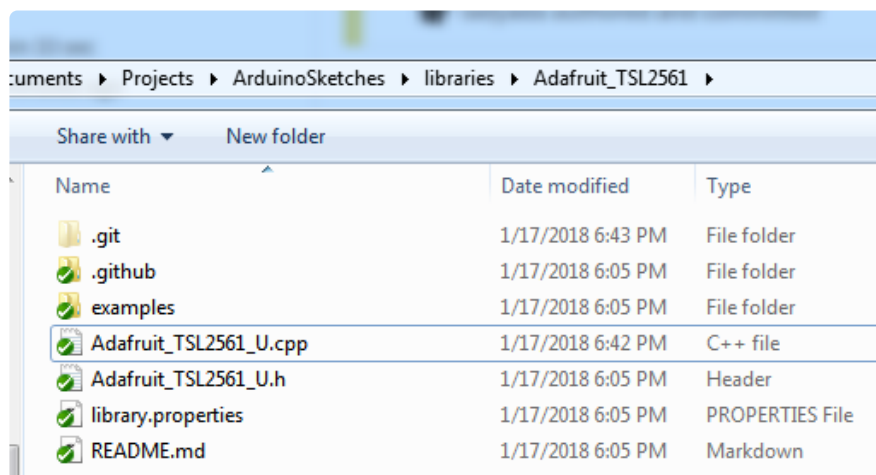
Yep, this guide is how to do it without having to host anything on your own server

Repository

Overall Structure

In this guide we'll watch as I transform [Adafruit_TSL2561 \(https://adafru.it/aZ9\)](https://adafru.it/aZ9), one of our first libraries, spruce it up and automate it. This guide doesn't talk about how to write code - just how to make it fancy!

We'll be using the 'classic' Arduino library structure. In this case, we make a new repo, with no spaces or dashes, on GitHub: **Adafruit_TSL2561** and `git clone` it into our **ArduinoSketches/libraries** folder. This makes it easy to test in Arduino and commit in the same folder



Inside the folder you'll need these 4 parts:

- `Adafruit_TSL2561.cpp` - the main cpp file that contains your code
- `Adafruit_TSL2561.h` - the main header file that has macros, enums, classes, etc!

- **library.properties** - the file that lets Arduino IDE and library manager know what's inside your library
- **README.md** - Appears in the front of the GitHub repo, a good place to put descriptions, images, instructions and a Travis tag

You may also have optional parts like

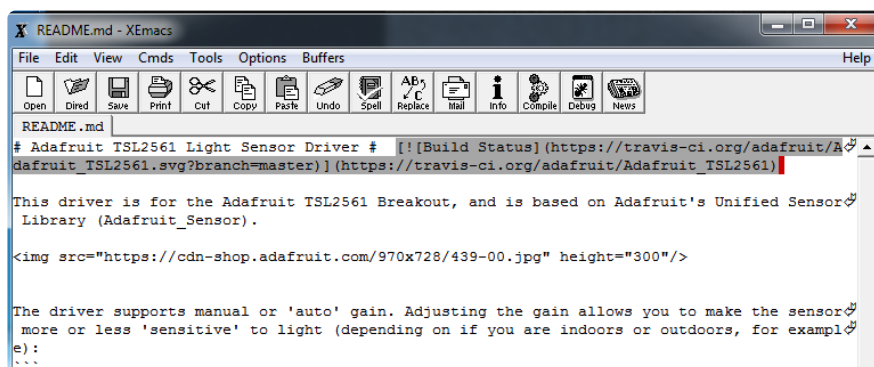
- **.gitignore** (see below) - helps keep your git commits from accidentally including temporary files
- **.github folder** with ISSUE_TEMPLATE.md and PULL_REQUEST_TEMPLATE.md - [see our examples here \(https://adafru.it/BRN\)](https://adafru.it/BRN) if you like you can adapt and reuse them.

library.properties is very particular about its formatting, [see here for a guide on how to make sure you get this in the right formatting \(https://adafru.it/ehu\)](https://adafru.it/ehu)! We won't cover it in detail here

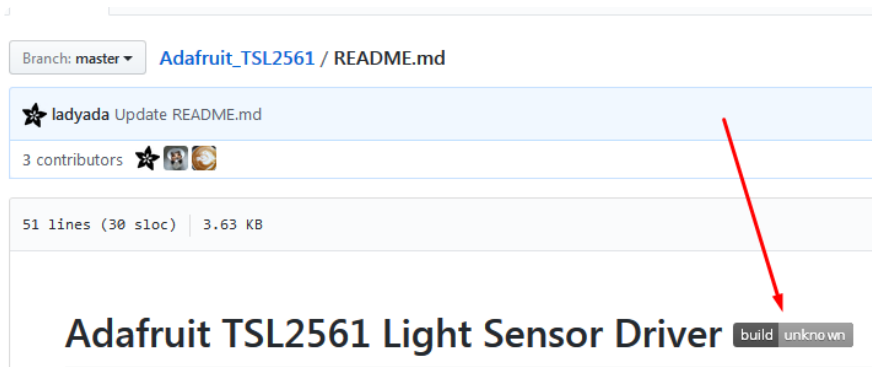
For **README.md** we do recommend adding a Travis tag and you might as well do that now. Open it up in a text editor and after the first heading add text like this:

```
[[Build Status](https://travis-ci.com/adafruit/Adafruit_TSL2561.svg?branch=master)](https://travis-ci.com/adafruit/Adafruit_TSL2561)
```

Except change the two sub-URLs to match your repository



When it renders for now it will say **Build Unknown** - we'll get that going later!



Create Empty gh-pages Branch

Rather than storing the documentation in your master branch, cluttering your repo, we'll have TravisCI put the files into the **gh-pages** branch. GitHub can be informed that documentation is there and it will host the webpages for you, it's very slick!

Make sure you've committed and pushed all your code before you continue, since you will be deleting from a branch and it can be scary!

```
cd /path/to/repository
git checkout --orphan gh-pages && \
rm -rf * .github .travis.yml .gitignore && \
echo "My gh-pages branch" > README.md && \
git add . && \
git commit -a -m "Clean gh-pages branch" && \
git push origin gh-pages && \
git checkout main
```

```
ladyada@VARICK MINGW32 ~/Documents/Projects/ArduinoSketches/libraries/Adafruit_TSL2561
$ git checkout --orphan gh-pages
Switched to a new branch 'gh-pages'

ladyada@VARICK MINGW32 ~/Documents/Projects/ArduinoSketches/libraries/Adafruit_TSL2561
$ ls
Adafruit_TSL2561_U.cpp  Adafruit_TSL2561_U.h  examples/  library.properties  pgmspace.h

ladyada@VARICK MINGW32 ~/Documents/Projects/ArduinoSketches/libraries/Adafruit_TSL2561
$ rm -rf *

ladyada@VARICK MINGW32 ~/Documents/Projects/ArduinoSketches/libraries/Adafruit_TSL2561
$ echo "My gh-pages branch" > README.md

ladyada@VARICK MINGW32 ~/Documents/Projects/ArduinoSketches/libraries/Adafruit_TSL2561
$ git add .

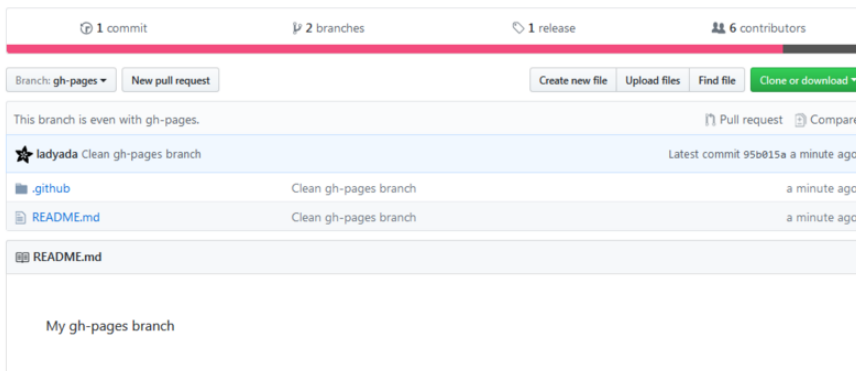
ladyada@VARICK MINGW32 ~/Documents/Projects/ArduinoSketches/libraries/Adafruit_TSL2561
$ git commit -a -m "Clean gh-pages branch"
[gh-pages (root-commit) 95b015a] Clean gh-pages branch
 3 files changed, 73 insertions(+)
 create mode 100644 .github/ISSUE_TEMPLATE.md
 create mode 100644 .github/PULL_REQUEST_TEMPLATE.md
 create mode 100644 README.md

ladyada@VARICK MINGW32 ~/Documents/Projects/ArduinoSketches/libraries/Adafruit_TSL2561
$ git push origin gh-pages
Counting objects: 6, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (5/5), done.
Writing objects: 100% (6/6), 2.30 KiB | 783.00 KiB/s, done.
Total 6 (delta 0), reused 0 (delta 0)
To github.com:adafruit/Adafruit_TSL2561.git
 * [new branch]      gh-pages -> gh-pages

ladyada@VARICK MINGW32 ~/Documents/Projects/ArduinoSketches/libraries/Adafruit_TSL2561
$ git checkout master
Switched to branch 'master'
Your branch is up to date with 'origin/master'.

ladyada@VARICK MINGW32 ~/Documents/Projects/ArduinoSketches/libraries/Adafruit_TSL2561
$
```

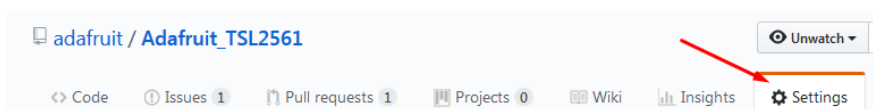
On GitHub you can verify the **gh-pages** branch is empty



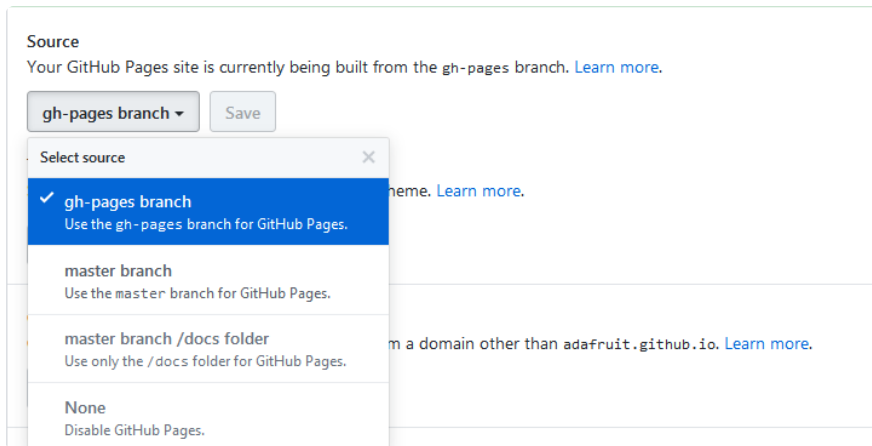
Enable GitHub Pages on gh-pages

Now you've created **gh-pages** you can set it up so GitHub will automatically create a website with our auto-generated documentation

Go to the **Settings** for the repository



And scroll down to **GitHub Pages**. From the drop-down select **gh-pages** as the source and hit Save



Save and you'll get notification of your new github.io URL. Note that nothing will be there at this time - that's OK! We'll get that going soon.

GitHub Pages

GitHub Pages is designed to host your personal, organization, or project pages from a GitHub repository.

✓ Your site is published at http://adafruit.github.io/Adafruit_TSL2561/

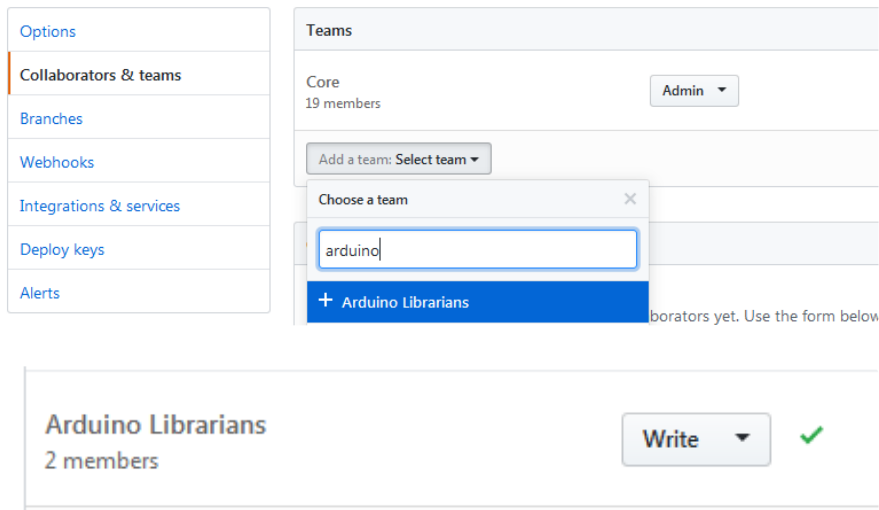
Optional: .gitignore

Put the following into your **master** branch's **.gitignore**, it will help you avoid committing doxygen files later

```
# Our handy .gitignore for automation ease
Doxyfile*
doxygen_sqlite3.db
html
```

Optional (For Adafruit Only!) Add Adafruit Librarians group

If you're working on an official Adafruit library, add the **Arduino Librarians** group to your repository with **write** access, which will let [adafruit-adabot](https://adafru.it/BRO) (<https://adafru.it/BRO>) push pages on your behalf!



Doxygen



About Doxygen

Doxygen is a somewhat ancient but popular method for creating documentation for C and C++ project (it also supports other languages but its great at C/C++). What's great about Doxygen is that the documentation lives with the source code in the same file so you are less likely to get out of sync

From [doxygen.org \(https://adafru.it/BRL\)](https://adafru.it/BRL)'s website:

Generate documentation from source codeDoxygen is the de facto standard tool for generating documentation from annotated C++ sources, but it also supports other popular programming languages such as C, Objective-C, C#, PHP, Java, Python, IDL (Corba, Microsoft, and UNO/OpenOffice flavors), Fortran, VHDL, Tcl, and to some extent D.Doxygen can help you in three ways:

1. It can generate an on-line documentation browser (in HTML) and/or an off-line reference manual (in \LaTeX) from a set of documented source files. There is also support for generating output in RTF (MS-Word), PostScript, hyperlinked PDF, compressed HTML, and Unix man pages. The documentation is extracted directly from the sources, which makes it much easier to keep the documentation consistent with the source code.

2. You can [configure \(https://adafru.it/BRP\)](https://adafru.it/BRP) doxygen to extract the code structure from undocumented source files. This is very useful to quickly find your way in large source distributions. Doxygen can also visualize the relations between the various elements by means of include dependency graphs, inheritance diagrams, and collaboration diagrams, which are all generated automatically.
3. You can also use doxygen for creating normal documentation (as I did for the doxygen user manual and web-site).

Doxygen is developed under Mac OS X and Linux, but is set-up to be highly portable. As a result, it runs on most other Unix flavors as well. Furthermore, executables for Windows are available.

Doxygen runs on the source code itself, and create HTML output that we can then point people to when they want detailed information about the library.

To keep our lives easier, instead of running Doxygen every commit, we'll use **Travis CI** to keep the documentation up to date. Every commit on GitHub will trigger TravisCI to call Doxygen on the source code, verify that documentation exists for each function, and will refresh the website pages.

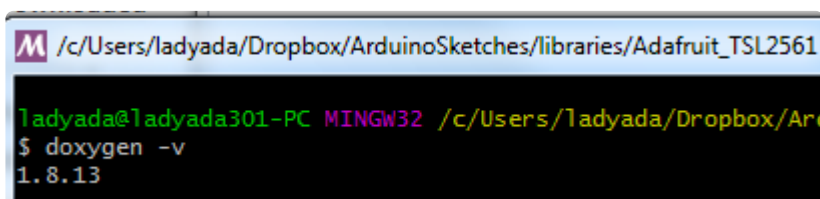
However, to get started, we'll want to run Doxygen locally so we can make sure our files are in good shape before committing them (otherwise we'll have to run Travis over and over until we get it right, which takes a lot longer than fixing it all locally)

Step 1) Install Doxygen

Doxygen runs in the command line, and is available for Mac, Win and Linux. On Mac/Linux we suggest using **brew** or **apt-get** or whatever package manager you have

For Windows, [visit the downloads page \(https://adafru.it/BRQ\)](https://adafru.it/BRQ) and scroll down to download the installer, then run it to install. Or if you have something like MSYS2, use the package manager.

In this guide we'll be using version 1.8.13 - if its something much older you may get errors

A screenshot of a terminal window. The top line shows the path `/c/Users/ladyada/Dropbox/ArduinoSketches/libraries/Adafruit_TSL2561`. The second line shows the prompt `ladyada@ladyada301-PC MINGW32 /c/Users/ladyada/Dropbox/Ar`. The third line shows the command `$ doxygen -v`. The fourth line shows the output `1.8.13`.

Step 2) Grab our Doxyfile

Doxygen needs a 'setup' file called Doxyfile. There's two ways to go about this step.

If you are not going to be using our Travis CI auto-doxy-runner or if you have needs for a custom Doxyfile you should run `doxygen -g` from within the library folder, which will create a Doxyfile. Then edit it to your hearts desire and commit it to your GitHub repo

```
$ doxygen -g

Configuration file 'Doxyfile' created.

Now edit the configuration file and enter

doxygen Doxyfile

to generate the documentation for your project
```

However, if you are planning to use TravisCI to auto-generate Doxygen and you don't have any big changes you want to make to our setup we recommend using our default Doxyfile, and not committing it. Our TravisCI will automatically grab the Doxyfile for you, which means we can update it from time to time as doxygen updates and changes the config file. **This is what we'll assume you're doing and continue as-is!**

In which case, download or `wget` our example Doxyfile from <https://raw.githubusercontent.com/adafruit/travis-ci-arduino/master/Doxyfile.default> (<https://adafru.it/BRR>), place it in the base of the library, and rename it `Doxyfile`

```
/c/Users/ladyada/Dropbox/ArduinoSketches/libraries/Adafruit_TSL2561

ladyada@ladyada301-PC MINGW32 /c/Users/ladyada/Dropbox/ArduinoSketches/libraries/Adafruit_TSL2561
$ wget https://raw.githubusercontent.com/adafruit/travis-ci-arduino/master/Doxyfile.default
--2018-01-17 00:15:45-- https://raw.githubusercontent.com/adafruit/travis-ci-arduino/master/Doxyfile.default
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 151.101.208.133
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)[151.101.208.133]:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 108369 (106K) [text/plain]
Saving to: 'DoxyFile.default'

DoxyFile.default      100%[=====] 105.83K  --.-KB/s  in 0.02s
2018-01-17 00:15:45 (4.54 MB/s) - 'DoxyFile.default' saved [108369/108369]

ladyada@ladyada301-PC MINGW32 /c/Users/ladyada/Dropbox/ArduinoSketches/libraries/Adafruit_TSL2561
$ mv DoxyFile.default Doxyfile
```

Step 3) Run doxygen

Now you can try running `doxygen Doxyfile` and see the output

```

is not documented
/c/Users/ladyada/Dropbox/ArduinoSketches/libraries/Adafruit_TSL2561/Adafruit_TSL
2561_U.h:181: warning: parameters of member Adafruit_TSL2561_Unified::Adafruit_T
SL2561_Unified are not (all) documented
/c/Users/ladyada/Dropbox/ArduinoSketches/libraries/Adafruit_TSL2561/Adafruit_TSL
2561_U.h:182: warning: return type of member Adafruit_TSL2561_Unified::begin is
not documented
/c/Users/ladyada/Dropbox/ArduinoSketches/libraries/Adafruit_TSL2561/Adafruit_TSL
2561_U.h:188: warning: parameters of member Adafruit_TSL2561_Unified::setIntegra
tionTime are not (all) documented
/c/Users/ladyada/Dropbox/ArduinoSketches/libraries/Adafruit_TSL2561/Adafruit_TSL
2561_U.h:189: warning: parameters of member Adafruit_TSL2561_Unified::setGain ar
e not (all) documented
/c/Users/ladyada/Dropbox/ArduinoSketches/libraries/Adafruit_TSL2561/Adafruit_TSL
2561_U.h:191: warning: parameters of member Adafruit_TSL2561_Unified::calculateL
ux are not (all) documented
/c/Users/ladyada/Dropbox/ArduinoSketches/libraries/Adafruit_TSL2561/Adafruit_TSL
2561_U.h:191: warning: return type of member Adafruit_TSL2561_Unified::calculat
eLux is not documented

ladyada@ladyada301-PC MINGW32 /c/Users/ladyada/Dropbox/ArduinoSketches/libraries
/Adafruit_TSL2561
$ |

```

It's normal to get a massive number of complaints. Now it's time to document and get those complaints down to zero!

- [Read the Doxygen Manual \(https://adafru.it/GJa\)](https://adafru.it/GJa)! Its detailed and very informative
- [Check out our Doxygen tips page \(https://adafru.it/BRT\)](https://adafru.it/BRT), which is basically my cheat-sheet since I often forget the @ tags and formatting
- Look online or at other Doxygen'd code for inspiration

OK its a little annoying to do but eventually you'll get to zero output from Doxygen. Actually, because my compiled doxygen binary didn't come with CLANG support, I do get two warnings, but you can ignore these.

```

ladyada@ladyada301-PC MINGW32 /c/Users/ladyada/Dropbox/ArduinoSketches/libraries/Adafruit_TSL2561
$ doxygen Doxyfile
warning: Tag 'CLANG_ASSISTED_PARSING' at line 1082 of file 'Doxyfile' belongs to an option that was not enabled at compile time.
To avoid this warning please remove this line from your configuration file or upgrade it using "doxygen -u", or recompile
doxygen with this feature enabled.
warning: Tag 'CLANG_OPTIONS' at line 1090 of file 'Doxyfile' belongs to an option that was not enabled at compile time.
To avoid this warning please remove this line from your configuration file or upgrade it using "doxygen -u", or recompile
doxygen with this feature enabled.

```

Preview Documentation

OK now you can look in the folder and you'll see that in addition to the **Doxyfile**, you also have a **sqlite3.db** and an **html** folder.

Name	Date modified	Type	Size
.github	10/16/2016 11:31 ...	File folder	
examples	9/12/2017 6:39 PM	File folder	
html	1/17/2018 1:14 AM	File folder	
Adafruit_TSL2561_U.cpp	1/17/2018 12:55 AM	C++ File	19 KB
Adafruit_TSL2561_U.h	1/17/2018 1:08 AM	H File	10 KB
Doxyfile	1/17/2018 12:15 AM	File	106 KB
doxygen_sqlite3.db	1/17/2018 1:14 AM	Data Base File	660 KB
library.properties	1/17/2018 12:09 AM	PROPERTIES File	1 KB
README.md	1/17/2018 12:56 AM	markdown	4 KB

Don't commit these to your git repo! Instead, open up the html folder and double-click the `index.html`

Name	Date modified	Type	Size
folderclosed.png	1/17/2018 1:14 AM	PNG image	1 KB
folderopen.png	1/17/2018 1:14 AM	PNG image	1 KB
functions.html	1/17/2018 1:14 AM	Firefox HTML Doc...	5 KB
functions_func.html	1/17/2018 1:14 AM	Firefox HTML Doc...	4 KB
globals.html	1/17/2018 1:14 AM	Firefox HTML Doc...	14 KB
globals_defs.html	1/17/2018 1:14 AM	Firefox HTML Doc...	14 KB
globals_enum.html	1/17/2018 1:14 AM	Firefox HTML Doc...	3 KB
hierarchy.html	1/17/2018 1:14 AM	Firefox HTML Doc...	4 KB
index.html	1/17/2018 1:14 AM	Firefox HTML Doc...	4 KB
jquery.js	1/17/2018 1:14 AM	JScript Script File	169 KB
md_r_e_a_d_m_e.html	1/17/2018 1:14 AM	Firefox HTML Doc...	7 KB
menu.js	1/17/2018 1:14 AM	JScript Script File	2 KB
menudata.js	1/17/2018 1:14 AM	JScript Script File	1 KB
nav_f.png	1/17/2018 1:14 AM	PNG image	1 KB

Adafruit Library

Main Page Related Pages Classes Files

Adafruit TSL2561 Light/Lux sensor driver

Introduction

This is the documentation for Adafruit's TSL2561 driver for the Arduino platform. It is designed specifically to work with the Adafruit TSL2561 breakout. These sensors use I2C to communicate. 2 pins (SCL+SDA) are required to interface with the breakout.

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

Dependencies

This library depends on `Adafruit_Sensor` being present on your system. Please make sure you have installed the latest version before using this library.

Author

Written by Kevin "KTOWN" Townsend for Adafruit Industries.

License

BSD license, all text here must be included in any redistribution.

HISTORY

v2.0 - Rerrote driver for `Adafruit_Sensor` and Auto-Gain support, and added lux clipping check (returns 0 lux on sensor saturation) v1.0 - First

You will now be able to browse the documentation. You can see the nice front page you set up in the `.cpp` file

Click the Classes button

Adafruit Library

Main Page Related Pages **Classes** Files

Adafruit TSL2561 Light/Lux sensor driver

Class List

And you can locate your object, click on that...

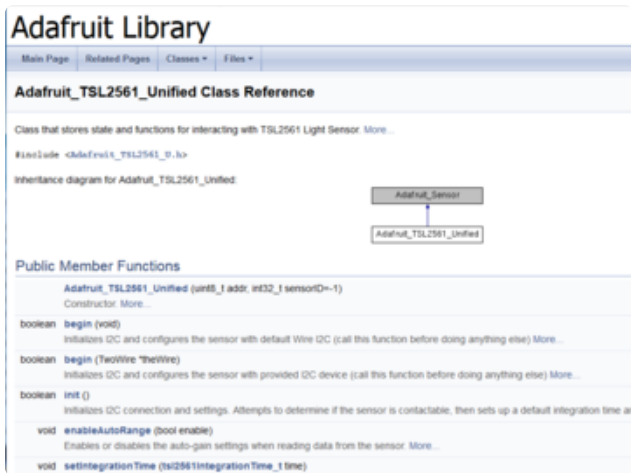
Adafruit Library

Main Page Related Pages Classes Files

Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Adafruit_TSL2561_Unified	Class that stores state and functions for interacting with TSL2561 Light Sensor
---------------------------------	---



Now you can see the details of your class documentation! Check it over for typos, corrections, or other stuff you want to add. Re-run `doxygen` as desired until you get it to a good spot

Commit Documented Code

You are very nearly done!

Commit and push your updated code to GitHub

Do not commit the Doxygen file (unless you changed it), the html folder, or the sqlite db. Only your fully updated and documented files!

```
GNU nano 2.8.7 File: /c/Users/ladyada/Dropbox/ArduinoSketches
Wrote some awwwwwesome documentation!
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# On branch master
# Your branch is up to date with 'origin/master'.
#
# Changes to be committed:
#   modified:   Adafruit_TSL2561_U.cpp
#   modified:   Adafruit_TSL2561_U.h
#   modified:   README.md
#
# Untracked files:
#   Doxyfile
#   doxygen_sqlite3.db
#   html/
#
```

```
[master 02f2582] Wrote some awwwwwesome documentation!
3 files changed, 414 insertions(+), 385 deletions(-)
rewrite Adafruit_TSL2561_U.h (80%)

Tadyada@ladyada301-PC MINGW32 /c/Users/ladyada/Dropbox/ArduinoSketches/libraries/Adafruit_TSL2561
$ git push
Counting objects: 5, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (5/5), done.
Writing objects: 100% (5/5), 3.93 KiB | 804.00 KiB/s, done.
Total 5 (delta 3), reused 0 (delta 0)
remote: Resolving deltas: 100% (3/3), completed with 3 local objects.
To github.com:adafruit/Adafruit_TSL2561.git
 48be091..02f2582 master -> master

Tadyada@ladyada301-PC MINGW32 /c/Users/ladyada/Dropbox/ArduinoSketches/libraries/Adafruit_TSL2561
$ |
```

OK you are ready for the next, amazing step...**automation!**

Doxygen Tips

Main .cpp File Header

This fancy file header from KTown has multiple @ tags that will make a very nice 'front page' for your documentation and contains useful information! Copy and paste the whole chunk, then edit as necessary:

- `@file` - change this to the name of the file the header is at the top of!
- `@mainpage` - This will be the name of the documentation page, so make it short but descriptive
- `@section intro_sec Introduction` - Your intro. Longer than the header
- `@section dependencies Dependencies` - What other libraries or hardware are required
- `@section author Author` - You and others!
- `@section license License` - How you want others to use this code

```
/*!
 * @file Adafruit_FX0S8700.cpp
 *
 * @mainpage Adafruit FX0S8700 accel/mag sensor driver
 *
 * @section intro_sec Introduction
 *
 * This is the documentation for Adafruit's FX0S8700 driver for the
 * Arduino platform. It is designed specifically to work with the
 * Adafruit FX0S8700 breakout: https://www.adafruit.com/products/3463
 *
 * These sensors use I2C to communicate, 2 pins (SCL+SDA) are required
 * to interface with the breakout.
 *
 * Adafruit invests time and resources providing this open source code,
 * please support Adafruit and open-source hardware by purchasing
 * products from Adafruit!
 *
 * @section dependencies Dependencies
 *
 * This library depends on https://github.com/adafruit/Adafruit\_Sensor
 * Adafruit_Sensor being present on your system. Please make sure you have
 * installed the latest version before using this library.
 *
 * @section author Author
 *
 * Written by Kevin "KTOWN" Townsend for Adafruit Industries.
 *
 * @section license License
 *
 * BSD license, all text here must be included in any redistribution.
 */
```

NOTE: You can insert simple HTML style tags in your text, and they will be rendered properly in the documentation, included 'a href=' type links.

Main .h File Header

The .h is shorter, and doesn't have the special sections but you **do** need to add a `@file` tag!

```
/*!
 * @file Adafruit_FXOS8700.h
 *
 * This is part of Adafruit's FXOS8700 driver for the Arduino platform. It is
 * designed specifically to work with the Adafruit FXOS8700 breakout:
 * https://www.adafruit.com/products/3463
 *
 * These sensors use I2C to communicate, 2 pins (SCL+SDA) are required
 * to interface with the breakout.
 *
 * Adafruit invests time and resources providing this open source code,
 * please support Adafruit and open-source hardware by purchasing
 * products from Adafruit!
 *
 * Written by Kevin "KTOWN" Townsend for Adafruit Industries.
 *
 * BSD license, all text here must be included in any redistribution.
 */
```

If you don't see any global variables, enums or typedefs in your documentation, you probably forgot to add the `@file` tag to your file. Without this tag, it won't parse global values properly.

Documenting Functions

The most work will be in documenting functions. Here's another lovely example from KTown we can reference:

```
/*!
*****
@brief Gets the most recent sensor events.
       This function reads from both the accelerometer and the
       magnetometer in one call, and is a deviation from the standard
       Adafruit_Sensor API, but is provided as a convenience since most
       AHRS algorithms require sensor samples to be as close in time as
       possible.
@param accelEvent
       A reference to the sensors_event_t instances where the
       accelerometer data should be written.
@param magEvent
       A reference to the sensors_event_t instances where the
       magnetometer data should be written.
@return True if the event read was successful, otherwise false.
*/
```



```
*/
/*****
bool Adafruit_FX0S8700::getEvent(sensors_event_t* accelEvent, sensors_event_t*
magEvent)
```

The first and last `/*`

```
*****
```

`*****/` are not required but we think it looks good.

Brief & Description

Start with `@brief` and add a short sentence, ending with a `.` After that you can write as much as you like and it goes into the longer description of the function. KTown did some fun text-justification to make it all line up but if your editor doesn't do that, you can make it one long sentence like so:

```
@brief Gets the most recent sensor events. This function reads from
both the accelerometer and the magnetometer in one call, and is a
deviation from the standard Adafruit_Sensor API, but is provided as a
convenience since most AHRS algorithms require sensor samples to be
as close in time as possible.
```

Parameters

Each parameter needs a `@param` line. If you have no parameters to the function, skip this part. The word right after `@param` must be the same as the name of the parameter (make sure the parameter name in the `.cpp` file matches the one in `.h`) If you want you can make the name and the text on separate lines, or just put on one line like:

```
@param magEvent A reference to the sensors_event_t instances where
the magnetometer data should be written.
```

Return Value

Lastly, we need to put in the return value details. You can skip this if your function returns void or is a constructor. Otherwise, put in a `@returns` with a line explaining what it returns

Macros and Enums

After functions, you'll need to document all your enumerations and macros

(`#define`s)

For enums, you only need a comment beforehand like so, but I like to add a comment per line also:

```
/** TSL2561 offers 2 gain settings */
typedef enum
{
    TSL2561_GAIN_1X          = 0x00,    // No gain
    TSL2561_GAIN_16X       = 0x10,    // 16x gain
}
tsl2561Gain_t;
```

```
/** TSL2561 I2C Registers */
enum
{
    TSL2561_REGISTER_CONTROL      = 0x00, // Control/power register
    TSL2561_REGISTER_TIMING       = 0x01, // Set integration time register
    TSL2561_REGISTER_THRESHHOLDL_LOW = 0x02, // Interrupt low threshold low-byte
    TSL2561_REGISTER_THRESHHOLDL_HIGH = 0x03, // Interrupt low threshold high-byte
    TSL2561_REGISTER_THRESHHOLDH_LOW = 0x04, // Interrupt high threshold low-byte
    TSL2561_REGISTER_THRESHHOLDH_HIGH = 0x05, // Interrupt high threshold high-byte
    TSL2561_REGISTER_INTERRUPT    = 0x06, // Interrupt settings
    TSL2561_REGISTER_CRC          = 0x08, // Factory use only
    TSL2561_REGISTER_ID           = 0x0A, // TSL2561 identification setting
    TSL2561_REGISTER_CHAN0_LOW    = 0x0C, // Light data channel 0, low byte
    TSL2561_REGISTER_CHAN0_HIGH  = 0x0D, // Light data channel 0, high byte
    TSL2561_REGISTER_CHAN1_LOW    = 0x0E, // Light data channel 1, low byte
    TSL2561_REGISTER_CHAN1_HIGH  = 0x0F, // Light data channel 1, high byte
};
```

For `#define`'s you'll need to add a proper one-line comment. the `///
comment is easy to put after each line`

```
#define TSL2561_LUX_LUXSCALE      (14)    ///  
Scale by 2^14  
#define TSL2561_LUX_RATIOSCALE  (9)     ///  
Scale ratio by 2^9  
#define TSL2561_LUX_CHSCALE     (10)    ///  
Scale channel values by 2^10  
#define TSL2561_LUX_CHSCALE_TINT0 (0x7517) ///  
322/11 * 2^TSL2561_LUX_CHSCALE  
#define TSL2561_LUX_CHSCALE_TINT1 (0x0FE7) ///  
322/81 * 2^TSL2561_LUX_CHSCALE
```

Classes & Objects

Don't forget classes/objects also need a description!

```
/**
 *
 * @brief Class that stores state and functions for interacting with TSL2561
 * Light Sensor
 */
```

```
/*  
class Adafruit_TSL2561_Unified : public Adafruit_Sensor {
```

Optional: Code Snippets and Fixed Width Text Blocks

While you might want to avoid inserting complex code snippets since it becomes a maintenance risk, if you do need to include a code snippet or a block of 'fixed width' text, you can use the `@code` and `@endcode` tags, as shown below:

Make sure all of your comments use `/**` (and not `/* .. */`) inside the code block!

@section example_getevent Example

The following loop implementation shows how you can use the `.getEvent` function to continuously read data from the sensor, and display it on the Serial Monitor:

```
@code  
void loop(void)  
{  
  sensors_event_t aevent, mevent;  
  
  // Get a new sensor event from the accelerometer and magnetometer  
  accelmag.getEvent(&aevent, &mevent);  
  
  // Display the accel results (acceleration is measured in m/s^2)  
  Serial.print("A ");  
  Serial.print("X: "); Serial.print(aevent.acceleration.x, 4); Serial.print(" ");  
  Serial.print("Y: "); Serial.print(aevent.acceleration.y, 4); Serial.print(" ");  
  Serial.print("Z: "); Serial.print(aevent.acceleration.z, 4); Serial.print(" ");  
  Serial.println("m/s^2");  
  
  // Display the mag results (mag data is in uTesla)  
  Serial.print("M ");  
  Serial.print("X: "); Serial.print(mevent.magnetic.x, 1); Serial.print(" ");  
  Serial.print("Y: "); Serial.print(mevent.magnetic.y, 1); Serial.print(" ");  
  Serial.print("Z: "); Serial.print(mevent.magnetic.z, 1); Serial.print(" ");  
  Serial.println("uT");  
  
  Serial.println("");  
  
  delay(500);  
}  
@endcode
```

This will result in something resembling the following output:

```
• getEvent() (2/2)
bool Adafruit_FXOS8700::getEvent ( sensors_event_t* accelEvent,
                                sensors_event_t* magEvent
                              )

Gets the most recent sensor events.
This function reads from both the accelerometer and the magnetometer in one call, and is a deviation from the standard Adafruit_Sensor API, but is provided as a convenience since most AHRS algorithms require sensor samples to be as close in time as possible.

Parameters
accelEvent A reference to the sensors_event_t instances where the accelerometer data should be written.
magEvent A reference to the sensors_event_t instances where the magnetometer data should be written.

Returns
True if the event read was successful, otherwise false.

Example
The following loop implementation shows how you can use the .getEvent function to continuously read data from the sensor, and display it on the Serial Monitor:

void loop(void)
{
  sensors_event_t accel;
  sensors_event_t mag;
  bool success = true;

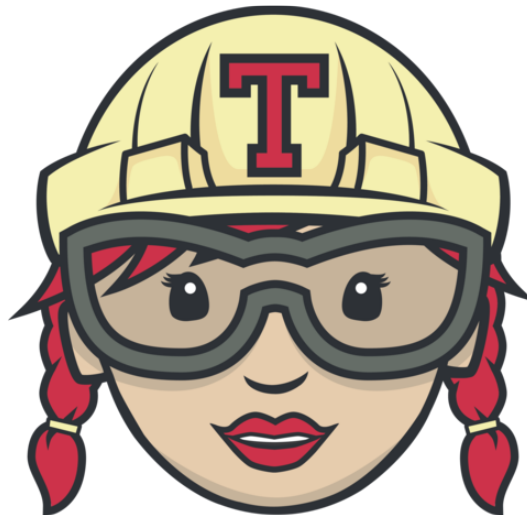
  // Get a new sensor event:
  success = getEvent(&accel, &mag);

  // Display the accel results (acceleration is measured in m/s^2)
  Serial.println(" ");
  Serial.print("X: "); Serial.print(accel.acceleration.x); Serial.print(" ");
  Serial.print("Y: "); Serial.print(accel.acceleration.y); Serial.print(" ");
  Serial.print("Z: "); Serial.print(accel.acceleration.z); Serial.print(" ");
  Serial.println(" ");

  // Display the mag results (mag data is in uTesla)
  Serial.println(" ");
  Serial.print("X: "); Serial.print(accel.magnetic.x); Serial.print(" ");
  Serial.print("Y: "); Serial.print(accel.magnetic.y); Serial.print(" ");
  Serial.print("Z: "); Serial.print(accel.magnetic.z); Serial.print(" ");
  Serial.println(" ");
  Serial.println(" ");
  delay(500);
}
```

Travis CI

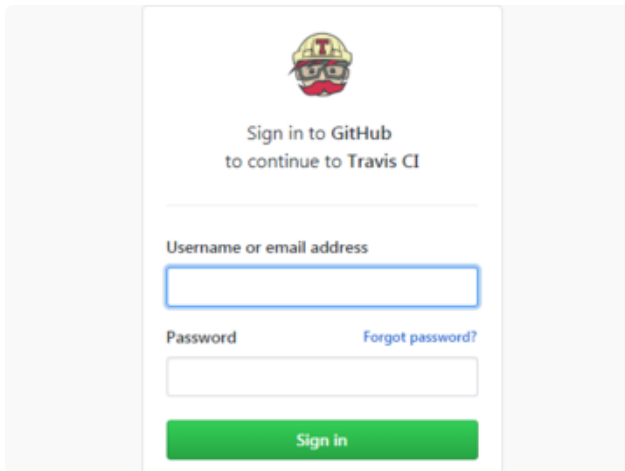
This process is now deprecated. CI has been moved to GitHub Actions.



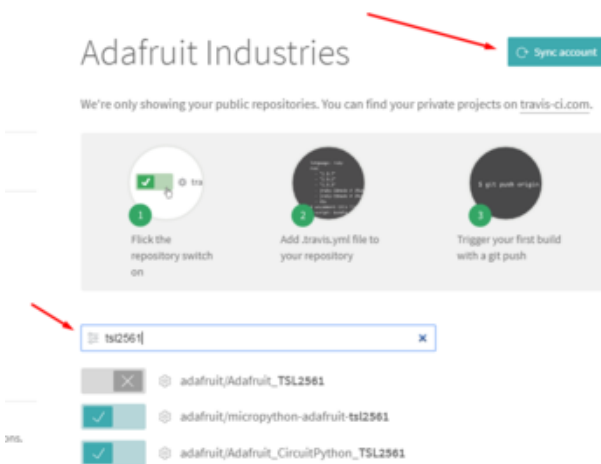
Now that we have fully documented the library, we are ready to add automation. Automation means we don't have to remember to run doxygen, or compile every platform every time. Automation will do this for you after every commit or pull request! You'll make fewer mistakes and get better pulls from the community. We'll be using **Travis CI**, a very popular and well run **Continuous Integration** service. You might be wondering - how much is the monthly subscription? **Nothing!** For open source libraries, you can run travis free of cost at travis-ci.org (<https://adafru.it/fuN>)

(Adafruit pays for the .com version of the service which we use a lot for internal repositories, and we recommend paying for it if you're building anything commercial because it's soooo worth it.)

Sign up for Travis

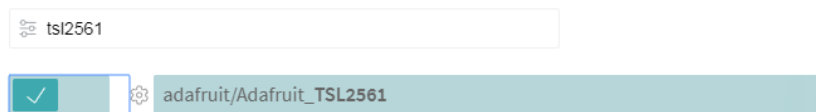


Use your GitHub login to register for Travis at travis-ci.org (<https://adafru.it/KFL>). This is required



Once logged in, you can **Sync** your account and **Search** for the repository you want to automate

Flick the switch to turn on Travis for that repository



Nothing will happen quite yet. That's OK! We have to add our Travis config file to the repository

Travis YML

Travis will create a fully new computer image every time and run your shell script instructions to compile the code and document it.

Here is our demo configuration file:

```

language: c
sudo: false
cache:
  directories:
    - ~/arduino_ide
    - ~/.arduino15/packages/
git:
  depth: false
  quiet: true
addons:
  apt:
    sources:
      - llvm-toolchain-trusty-5.0
      - key_url: 'http://apt.llvm.org/llvm-snapshot.gpg.key'
    packages:
      - python3-pip
      - python3-wheel
      - clang-format-5.0
env:
  global:
#   - ARDUINO_IDE_VERSION="1.8.10"
#   - PRETTYNAME="Adafruit FT6206 Arduino Library"
# Optional, will default to "$TRAVIS_BUILD_DIR/Doxyfile"
#   - DOXYFILE: $TRAVIS_BUILD_DIR/Doxyfile

before_install:
  - source <(curl -SLs https://raw.githubusercontent.com/adafruit/travis-ci-arduino/master/install.sh)
  - curl -SLs https://raw.githubusercontent.com/adafruit/travis-ci-arduino/master/run-clang-format.py > run-clang-format.py

install:
  - arduino --install-library "Adafruit ILI9341","Adafruit GFX Library"

script:
  - python run-clang-format.py -r .
  - build_main_platforms

# Generate and deploy documentation
after_success:
  - source <(curl -SLs https://raw.githubusercontent.com/adafruit/travis-ci-arduino/master/library_check.sh)
  - source <(curl -SLs https://raw.githubusercontent.com/adafruit/travis-ci-arduino/master/doxy_gen_and_deploy.sh)

```

And here's what it all means:

language: c This part is easy, we're going to be compiling C / C++ code.

sudo: false means We don't need root on the container (if you don't need it, don't use it!)

```

# Blacklist
branches:
  except:
    - gh-pages

```

This means we will not perform the full compile/documentation step on when we push to the **gh-pages** branch. This is so we don't accidentally trigger Travis to run twice after we push the documentation

```
env:
  global:
  - PRETTYNAME="Adafruit FT6206 Arduino Library"
```

This is the 'nice pretty' name of the library, which will be displayed at the header of the documentation. **You must change this to avoid confusion!**

```
before_install:
- source &lt;(curl -SLs https://raw.githubusercontent.com/adafruit/travis-ci-arduino/master/install.sh)
```

This step will install Arduino on your virtual computer and set up the board types as well, such as Uno, ESP8266, Zero, Due, etc! It's just a shell script so you can check it out at the URL to see what its' doing

```
install:
- arduino --install-library "Adafruit ILLI9341","Adafruit GFX Library"
```

This line adds more configuration setup. The text after the - is literally a bash shell command. We're [calling the Arduino binary on the command line and instructing it to do stuff like install some libraries \(https://adafru.it/fuR\)](https://adafru.it/fuR). If you do not need to install any libraries or any other configuration steps, just remove these two lines.

```
script:
- build_main_platforms
```

This calls a function we created in the **before_install** step, when we ran that shell script from **adafruit/travis-ci-arduino**

We have a few different functions, but this one will build for the main platforms we aim to support: Arduino UNO (ATmega328P), Leonardo (ATmega32u4), Zero (ATSAMD21), Due (ATSAMX), and ESP8266

```
# Generate and deploy documentation
after_success:
- source &lt;(curl -SLs https://raw.githubusercontent.com/adafruit/travis-ci-arduino/master/library_check.sh)
- source &lt;(curl -SLs https://raw.githubusercontent.com/adafruit/travis-ci-arduino/master/doxy_gen_and_deploy.sh)
```

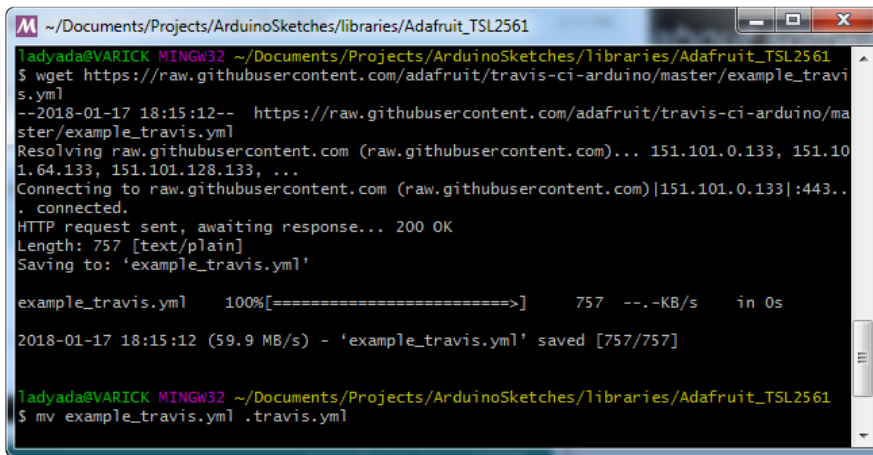
Lastly we have two more scripts we run. **library_check.sh**, at this time, is an empty hook but we hope to add some things like checking format of the library and looking for missing or misformatted lines. You can remove this.

`doxy_gen_and_deploy.sh` Is the script that will do the doxygen generation and then push the pages to your gh-pages branch

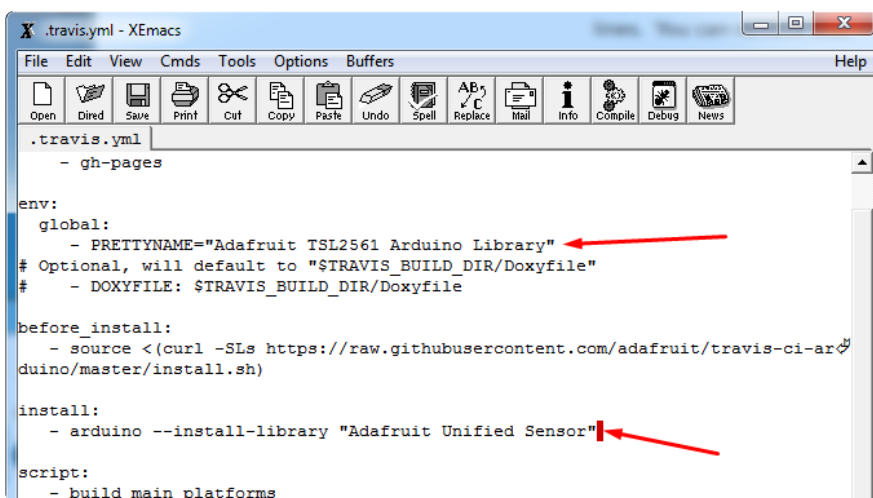
Don't forget you'll need that empty gh-page branch if you want to have the doxygen pages pushed for you, see earlier in the guide!

Add travis.yml To Library

OK now you are ready! Grab our example travis.yml file from https://raw.githubusercontent.com/adafruit/travis-ci-arduino/master/example_travis.yml (<https://adafru.it/BRU>), rename it `.travis.yml` and place in your library repository.



Edit it as we explain above, in particular edit the `PRETTYNAME` variable and change the extra-library-installation line as necessary!

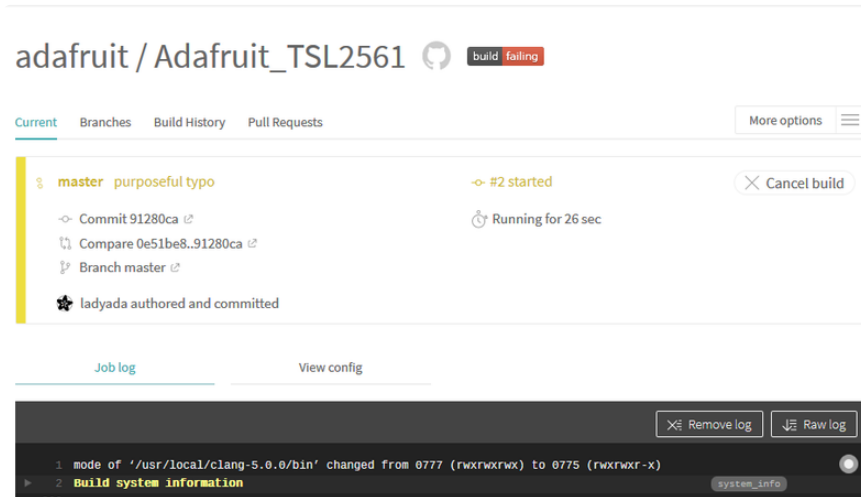


Save, commit and push the new `.travis.yml` file to your `master` branch

Checking Status

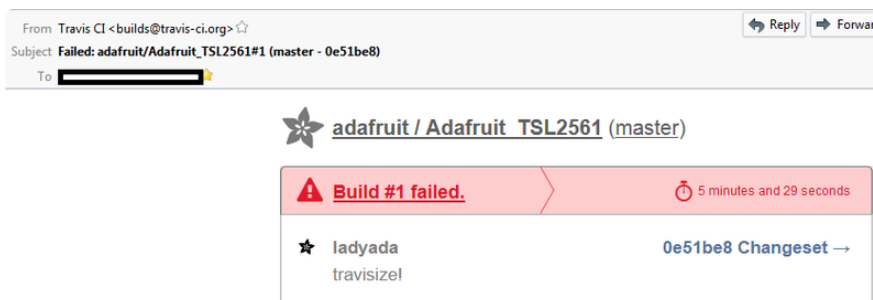
You can now check out the status of your build by visiting <http://travis-ci.org>

If you have multiple repositories, they'll appear on the left hand side. Yellow means working, green means pass and red means fail.



Failure/Passes

After every commit you may get an email or notification that Travis failed, or if it succeeded after failing you can get one too



Lets go and check out in the build log avialable at <https://travis-ci.org/yourgithubaccount/repositoryname>

In this case, you can see we failed to compile the `sensorapi.ino` demo when running the zero test (SAMD21) due to an `#include "avr/delay.h"` now we know we need to either `#ifdef` it out or restructure our code.

```

500 $ source <(curl -sL https://raw.githubusercontent.com/adafruit/travis-ci-arduino/master/install.sh)
501 $ arduino --install-library "Adafruit Unified Sensor"
502 $ build_main_platforms
503
504 #####
505 SWITCHING TO leonardo: ✓
506 #####
507 sensorapi.ino: ✓
508 #####
509 SWITCHING TO zero: ✓
510 #####
511 sensorapi.ino: ✗
512 #####
513 ----- DEBUG OUTPUT -----
514
515 Picked up JAVA_TOOL_OPTIONS:
516 Picked up JAVA_OPTIONS: -Xmx2048m -Xms512m
517 Loading configuration...
518 Initializing packages...
519 Preparing boards...
520 Verifying...
521 /home/travis/arduino_ide/libraries/Adafruit_Test_Library/Adafruit_TSL2561_U.cpp:42:23: fatal error: avr/delay.h: No such file or directory
522 #include "avr/delay.h"
523 ^
524 compilation terminated.
525 exit status 1
526
527 -----

```

Travis builds take 5 minutes or so, so you're best off doing fast iterative tests locally. After you've fixed the bug and tested it locally, commit and push!

This time, with any luck, you'll pass the main compilation tests!

```

527 #####
528 SWITCHING TO due: ✓
529 #####
530 sensorapi.ino: ✓
531 #####
532
533 #####
534 JSMN STATUS:
535 [{"repo": "https://github.com/adafruit/Adafruit_TSL2561.git", "status": 1, "passed": 5, "skipped": 0, "failed": 0, "platforms": {"leonardo": {"status": 1, "builds": {"sensorapi.ino": 1}}, "zero": {"status": 1, "builds": {"sensorapi.ino": 1}}, "esp266": {"status": 1, "builds": {"sensorapi.ino": 1}}, "uno": {"status": 1, "builds": {"sensorapi.ino": 1}}, "due": {"status": 1, "builds": {"sensorapi.ino": 1}}}]
536 #####
537 JSMN STATUS:
538
539
540 The command "build_main_platforms" exited with 0.
541
542 $ source <(curl -sL https://raw.githubusercontent.com/adafruit/travis-ci-arduino/master/library_check.sh)
543 $ source <(curl -sL https://raw.githubusercontent.com/adafruit/travis-ci-arduino/master/doxy_gen_and_deploy.sh)
544 Setting up the script...
545 Cloning into 'Adafruit_TSL2561'...
546 remote: Counting objects: 142, done.

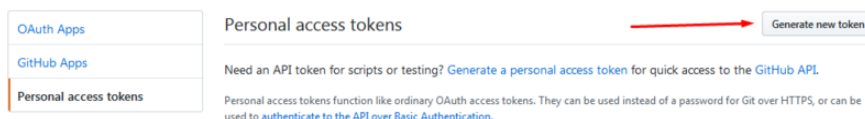
```

However your CI will probably still fail because we cannot 'push' the documentation to **gh-pages** so the last thing we have to do is give Travis CI permission to push to our repository

Adding Write Permissions

Log in to GitHub again and visit <https://github.com/settings/tokens> (<https://adafru.it/BRV>)

Click on **Generate New Token**



You'll need to set up the scope for this token, give it a nice name (you can have one token per repository or one token for all your repos)

And, if it's a public repo, click **public_repo** (otherwise you'll need to click all of **repo** for the access)

New personal access token

Personal access tokens function like ordinary OAuth access tokens. They can be used for HTTPS, or can be used to [authenticate to the API over Basic Authentication](#).

Token description

Adafruit_TSL2561 Travis CI Token

What's this token for?

Select scopes

Scopes define the access for personal tokens. [Read more about OAuth scopes](#).

<input type="checkbox"/>	repo	Full control of private repositories
<input type="checkbox"/>	repo:status	Access commit status
<input type="checkbox"/>	repo_deployment	Access deployment status
<input checked="" type="checkbox"/>	public_repo	Access public repositories
<input type="checkbox"/>	repo:invite	Access repository invitations

Now click **Generate Token** to get your key, which will be a bunch of numbers and letters. **Keep this token safe! It's the same as your GitHub login!**

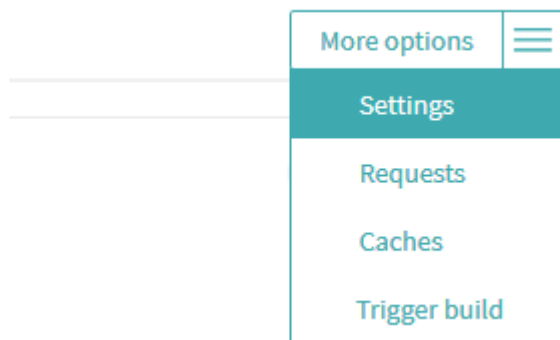
Tokens you have generated that can be used to access the [GitHub API](#).

Make sure to copy your new personal access token now. You won't be able to see it again!

✓ [Copy](#) [Edit](#) [Delete](#)

Personal access tokens function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to [authenticate to the API over Basic Authentication](#).

Back in your Travis CI page, go to the **Settings** for your repository



Scroll down to the **Environment Variables** section and create a **new** variable called `GH_REPO_TOKEN` then paste in the generated token in the box

Environment Variables

Notice that the values are not escaped when your builds are executed. Special characters (for bash) should be escaped accordingly.

GH_REPO_TOKEN OFF
Display value in build log

Make sure that "Display value in build log" is OFF!!!

Save it, your token is now stored safely and securely

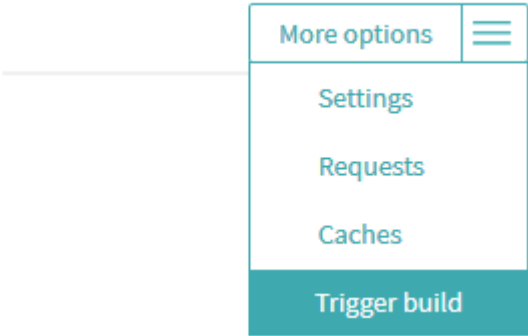
Environment Variables

Notice that the values are not escaped when your builds are executed. Special characters (for bash) should be escaped accordingly.

GH_REPO_TOKEN

Final Run and Check!

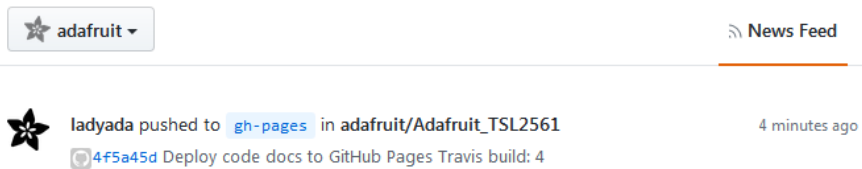
Now you can re-trigger a Travis build from the menu



This time, success!



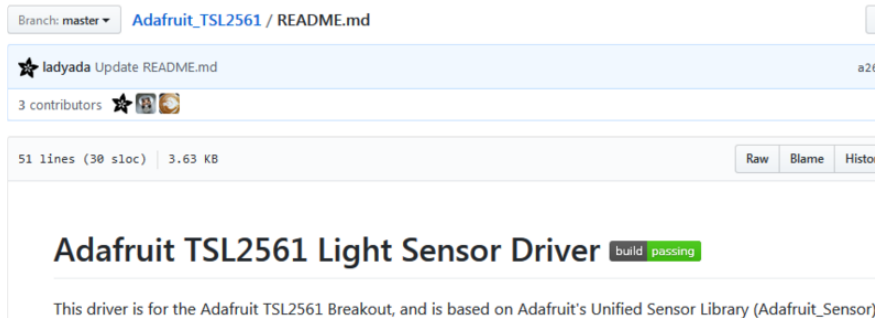
And if you look at your GitHub activity history you will see that 'you' pushed the code documentation in a Travis build



And at github.io you'll see your newly updated documentation!



And in the main repo, our Travis tag says its passing! Congrats, its a little tough the first time to do all these steps but once you get the hang of it, you'll be glad you did it later



Formatting with clang-format

About ClangFormat

ClangFormat (often called Clang) is a tool that allows you to automatically format C, C++, and Objective-C files. It makes your code more readable and saves your time and the time of anyone reviewing your code for a pull request by making it so neither of you has to worry about formatting very much. On Adafruit repositories, clang-format is run automatically on every commit and pull request, but you still have to run it locally since when it is run through a CI, it just tells you what needs to be reformatted without actually reformatting it.

Step 1) Install clang-format

For mac/linux, you can install with a package manager

macOS:

```
brew install clang-format
```

If you don't already have Homebrew installed, you can do it [here](https://adafru.it/OLE) (<https://adafru.it/OLE>)

Linux:

```
sudo apt install clang-format
```

```
cherrada@dh:~$ sudo apt install clang-format
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  clang-format-10 libclang-cpp10
The following NEW packages will be installed:
  clang-format clang-format-10 libclang-cpp10
0 upgraded, 3 newly installed, 0 to remove and 267 not upgraded.
Need to get 9,988 kB of archives.
After this operation, 44.9 MB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://us.archive.ubuntu.com/ubuntu focal/universe amd64 libclang-cpp10 amd64 1:10.0.0-4ubuntu1 [9,944 kB]
Get:2 http://us.archive.ubuntu.com/ubuntu focal/universe amd64 clang-format-10 amd64 1:10.0.0-4ubuntu1 [40.9 kB]
Get:3 http://us.archive.ubuntu.com/ubuntu focal/universe amd64 clang-format amd64 1:10.0-50-exp1 [3,272 B]
Fetched 9,988 kB in 0s (29.6 MB/s)
Selecting previously unselected package libclang-cpp10.
(Reading database ... 242305 files and directories currently installed.)
Preparing to unpack .../libclang-cpp10_1%3a10.0.0-4ubuntu1_amd64.deb ...
Unpacking libclang-cpp10 (1:10.0.0-4ubuntu1) ...
Selecting previously unselected package clang-format-10.
Preparing to unpack .../clang-format-10_1%3a10.0.0-4ubuntu1_amd64.deb ...
Unpacking clang-format-10 (1:10.0.0-4ubuntu1) ...
Selecting previously unselected package clang-format.
Preparing to unpack .../clang-format_1%3a10.0-50-exp1_amd64.deb ...
Unpacking clang-format (1:10.0-50-exp1) ...
Setting up libclang-cpp10 (1:10.0.0-4ubuntu1) ...
Setting up clang-format-10 (1:10.0.0-4ubuntu1) ...
Setting up clang-format (1:10.0-50-exp1) ...
Processing triggers for man-db (2.9.1-1) ...
Processing triggers for libc-bin (2.31-0ubuntu9) ...
cherrada@dh:~$
```

Windows:

Download the "Windows Installer" from the "Windows Snapshot Builds" section from the link below.

[LLVM Snapshot Builds](https://adafru.it/OLF)

<https://adafru.it/OLF>

Step 2) Run it

Navigate to the folder you'd like to run clang-format in and then run the following command, replacing **File_To_Format.cpp** with the filename of the file you'd like to format:

```
clang-format -i File_To_Format.cpp
```

Step 3) Add an alias to your .bashrc (optional)

I've found it can be really useful to have one simpler command that runs clang on all the pertinent files in a directory, and I modified a command from our Arduino CI repository to do that. Here's how you can use that too.

Linux:

1. Open a terminal
2. Type `nano .bashrc` and hit enter.
3. Paste the following near the bottom of the file:
 1. `alias format='find . -name "*.cpp" -o -name "*.c" -o -name "*.h"|xargs -I {} clang-format -i {}'`
4. Close the file with Shift+X and then enter Y when prompted if you would like to save the file.
5. Type `source .bashrc`
6. If there are no errors, then go back into the directory you'd like to run clang-format in, type `format`, and hit enter.
7. The command should look through the directory you're in and run clang-format on all .cpp, .c, and .h files.

These instructions may work on mac, but you will have to replace `.bashrc` with `.bash_profile`.