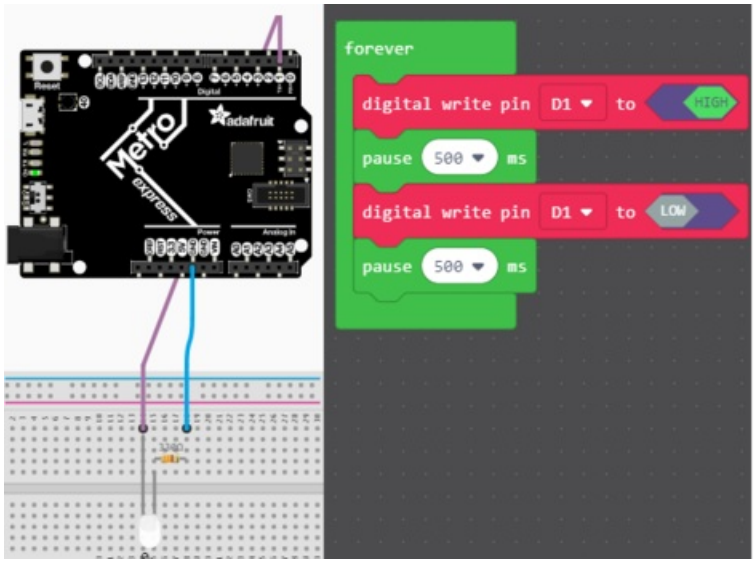
adafruit learning system

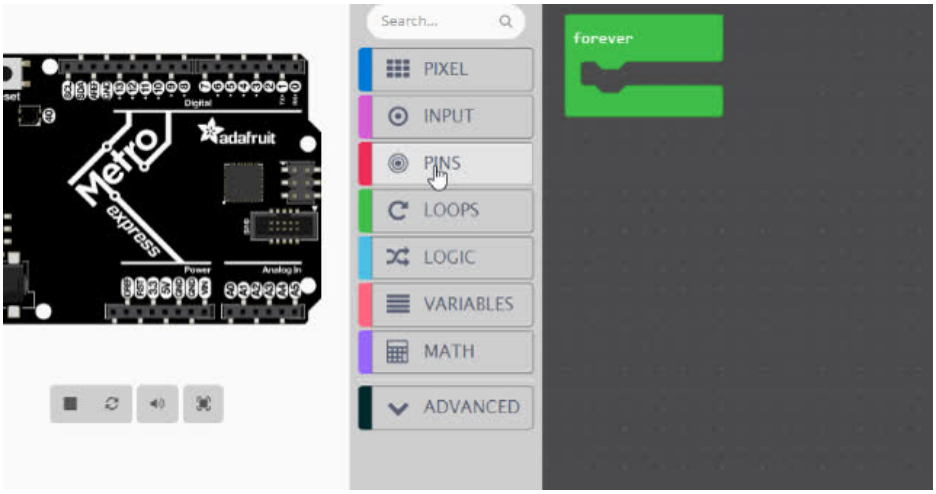
MakeCode Maker

Created by Peli de Halleux



Last updated on 2019-09-04 12:58:09 PM UTC

What is MakeCode Maker?



MakeCode Maker, https://maker.makecode.com, is a web-based code editor for physical computing. It provides a block editor, similar to Scratch or Code.org, and also a JavaScript editor for more advanced users.

Some of the key features of MakeCode are:

- **web based editor:** nothing to install
- **cross platform:** works in most modern browsers from tiny phone to giant touch screens
- **compilation in the browser:** the compiler runs in your browser, it's fast and works offline
- **blocks + JavaScript:** drag and drop blocks or type JavaScript, MakeCode let's you go back and forth between the two.
- **works offline:** once you've loaded the editor, it stays cached in your browser.
- **event based runtime:** easily respond to button clicks, shake gestures and more

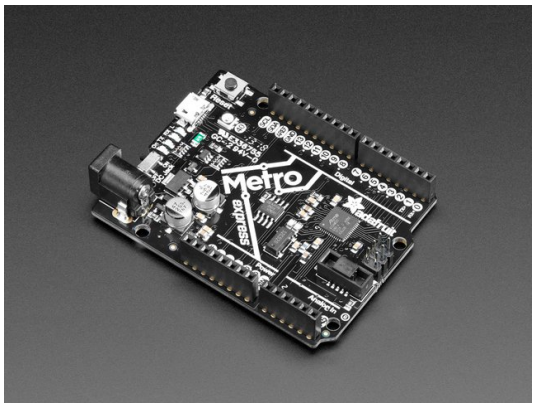
How is it related to makecode.adafruit.com ?

makecode.adafruit.com and maker.makecode.com are editors built using the MakeCode project. In both editors, one can use drag-and-drop blocks or JavaScript to program micro-controllers.

- makecode.adafruit.com specifically applies to the **Adafruit Circuit Playground Express only**
- maker.makecode.com aims at supporting the **Adafruit Express** boards (and more boards from different manufacturers), with an emphasis on breadboarding support.

Is it open source?

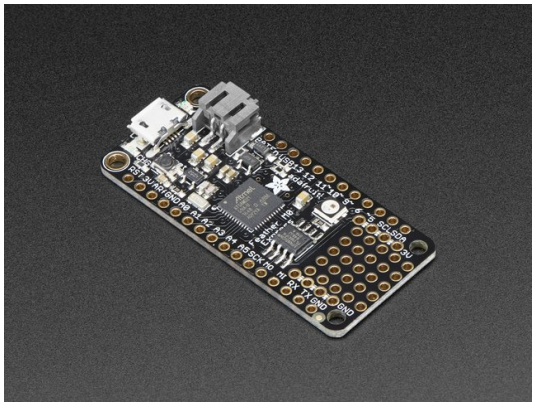
Yes, Maker is open source under MIT at https://github.com/Microsoft/pxt-maker.



Adafruit METRO M0 Express - designed for CircuitPython

$24.95
IN STOCK

Add To Cart



Adafruit Feather M0 Express - Designed for CircuitPython

$19.95
IN STOCK

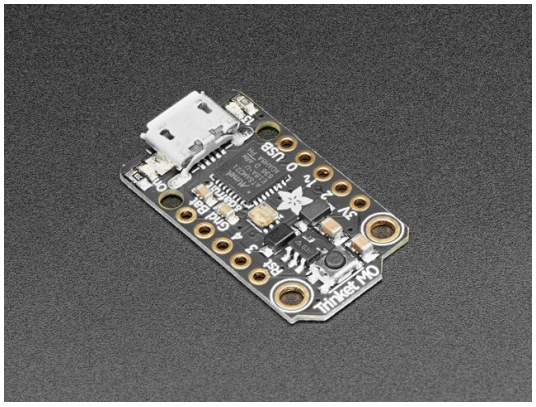
Add To Cart

Your browser does not support the video tag.

Adafruit GEMMA M0 - Miniature wearable electronic platform

OUT OF STOCK

Out Of Stock



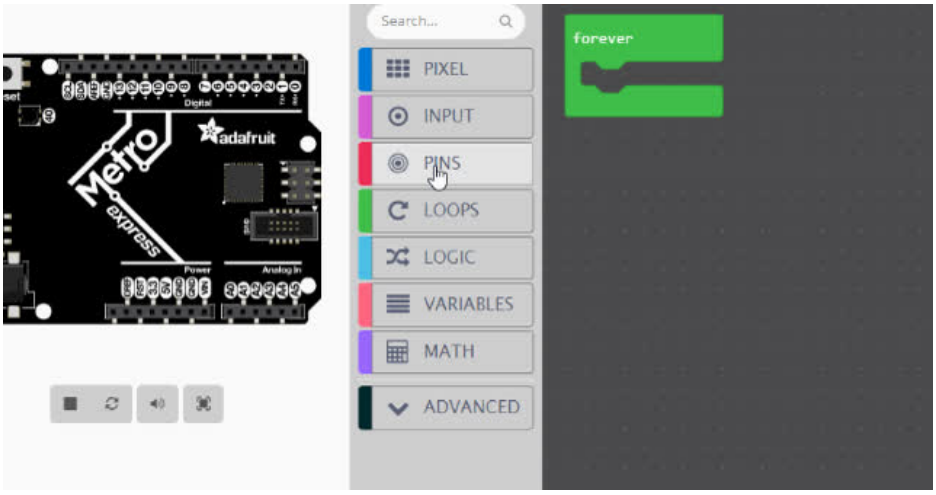
Adafruit Trinket M0 - for use with CircuitPython & Arduino IDE

$8.95
IN STOCK

Add To Cart

Editing Blocks

The block editor is the easiest way to get started with MakeCode Maker. You can drag and drop blocks from the category list. Each time you make a change to the blocks, the simulator will automatically restart and run the code. You can test your program in the browser! The simulator will also generate the wiring for your breadboard for simple programs.



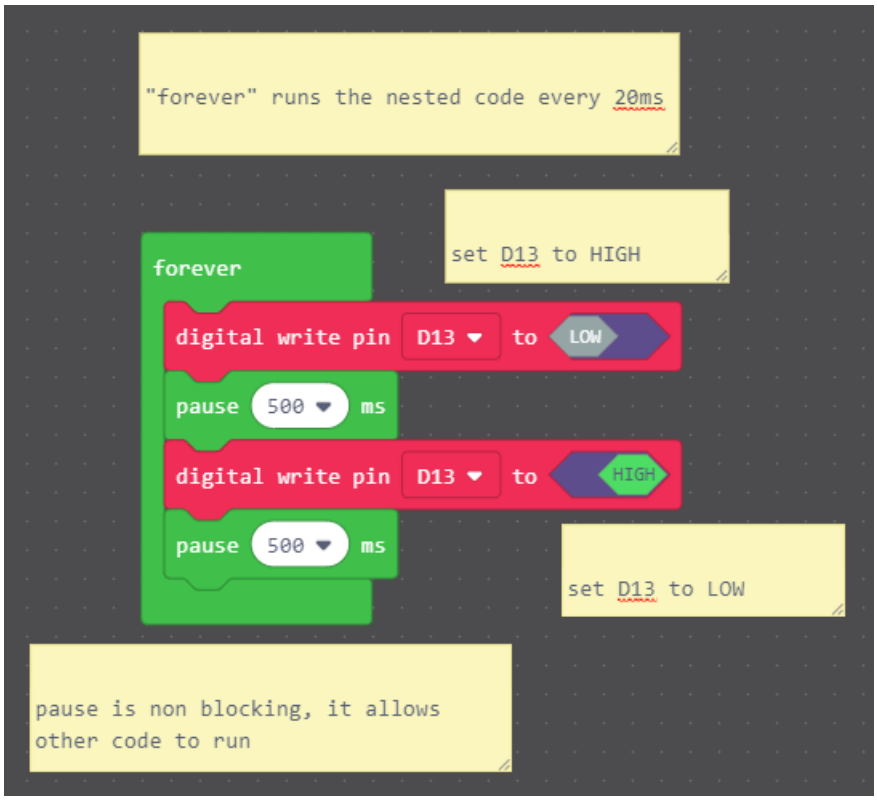
On the maker home screen, click on "New Project", then select which board you want to use (you can change board later too).

Blinky!

The animation above shows to use the blocks to create a program that blinks an LED.

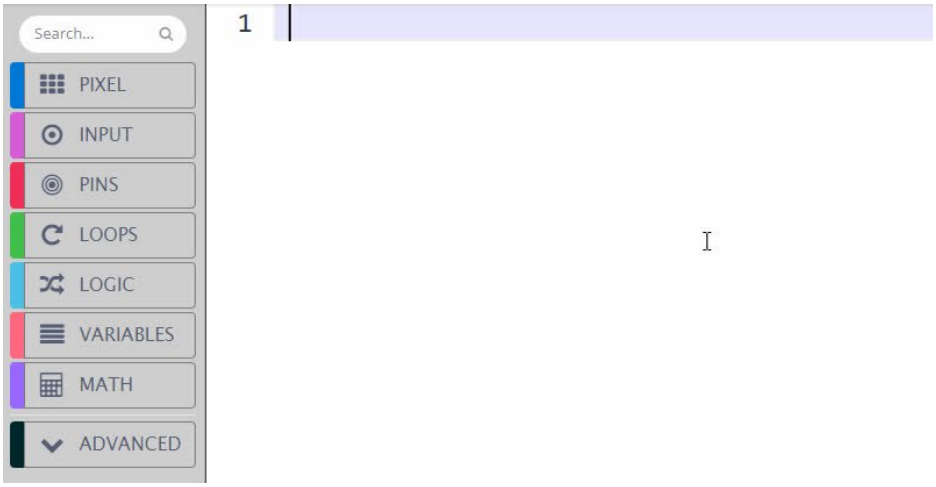
Creating a blink effect is done by setting the pin HIGH, **pause** for a little, then set the pin LOW, pause for a little, then repeat **forever**.

- **forever** runs blocks in a loop with a 20ms pause in between (it is similar to Arduino**loop**).
- **digital write pin** sets the pin to high or low
- **pause** blocks the current thread for 100ms. If other events or forever loops are running, they have the opporunity to run in parallel.


"forever" runs the nested code every 20ms
set D13 to HIGH
forever
digital write pin D13 to LOW
pause 500 ms
digital write pin D13 to HIGH
pause 500 ms
set D13 to LOW
pause is non blocking, it allows other code to run

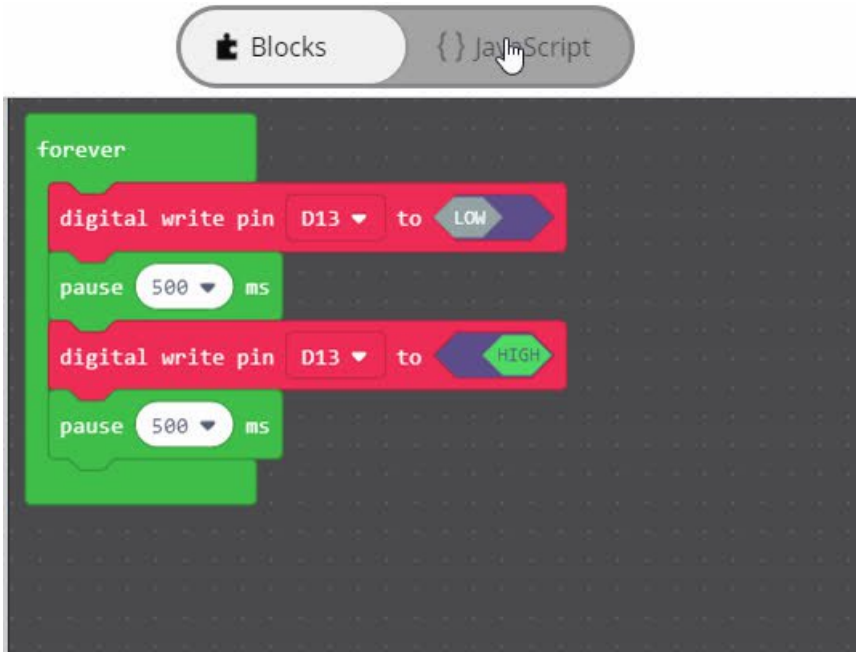
Editing JavaScript

MakeCode allows you to author your programs in a flavor of JavaScript optimized for micro-controllers. The code editor comes with error highlighting, auto-completion and other goodies. It is the same code editor that powers Visual Studio Code.



Blocks to JavaScript

Click on the "Blocks / JavaScript" toggle on top of the editor to enter the JavaScript mode. Your blocks will automatically be converted to JavaScript.



Downloading and Flashing

Getting your code into your device is very easy with MakeCode. You do not need to install any software on your machine and the process takes two steps:

- Step 1: Connect your board via USB
- Step 2: **Compile and Download** the .uf2 file into your board drive

We are going to go through these two steps in detail.

Step 1: Connect your board via USB

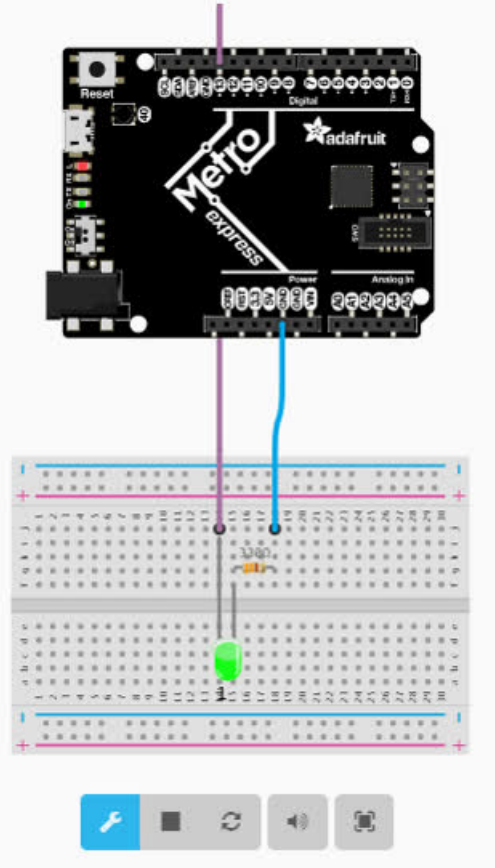
Connect your board to your computer via a USB cable. You should see a **MAKECODE** drive appear in your file explorer/finder. If your board is in bootloader mode, you will see drive names like **METROBOOT** or **GEMMABOOT**. We will call those **boardnameBOOT**.

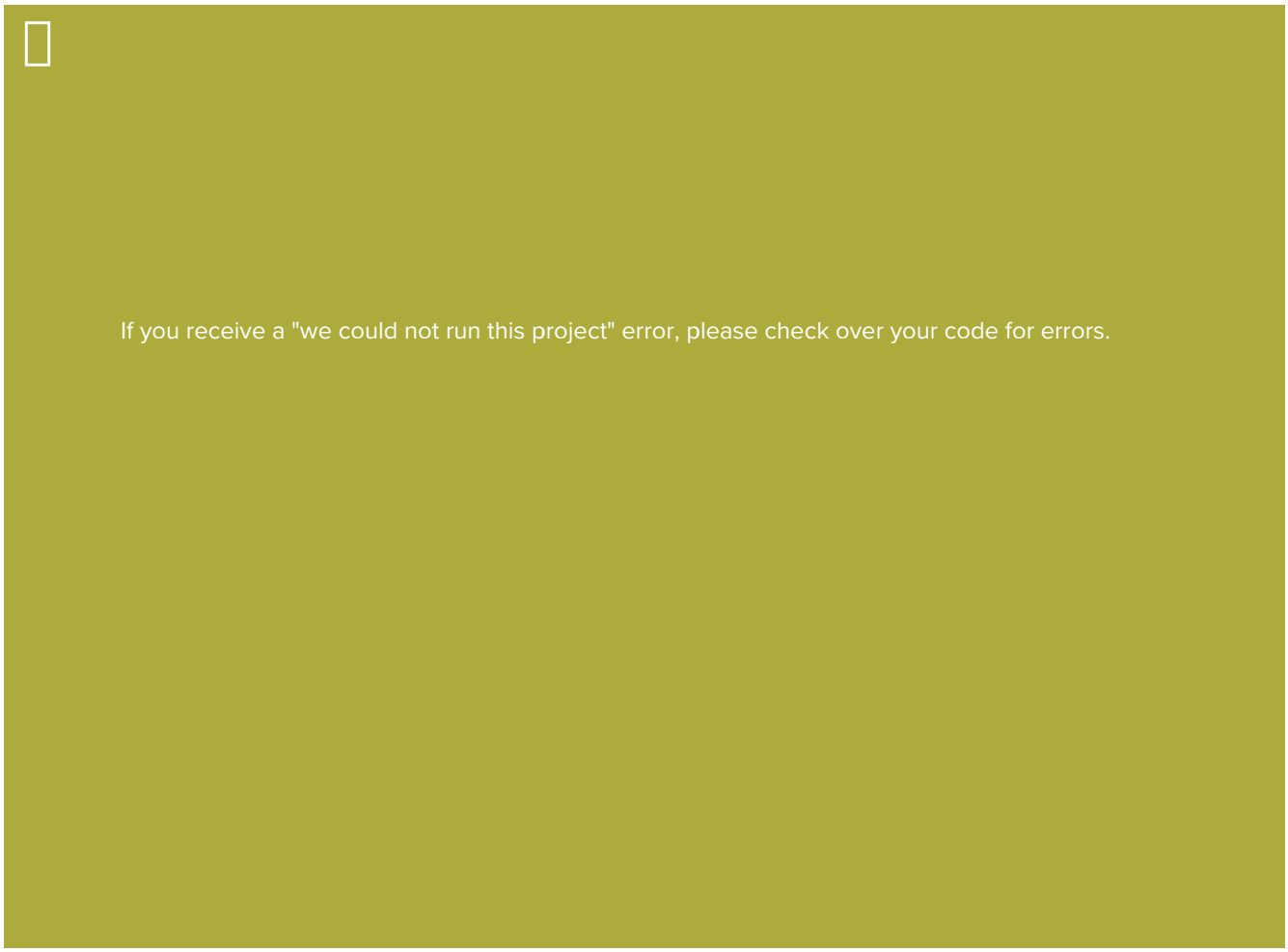
If it is your first time running MakeCode or if you have previously installed Arduino or CircuitPython, you may need to double press the reset button to get your board into bootloader mode.

Step 2: Test your code in the simulator

Let's first verify that our code compiles properly in MakeCode.

MakeCode has a built-in simulator that re-loads and re-runs code when restarted. This is an easy way to both ensure that our code compiles and simulate it before moving it onto the board. The refresh button re-loads the simulator with your latest version of block code.


Reset
Digital
adafruit
Metro
express
Power
Analog In

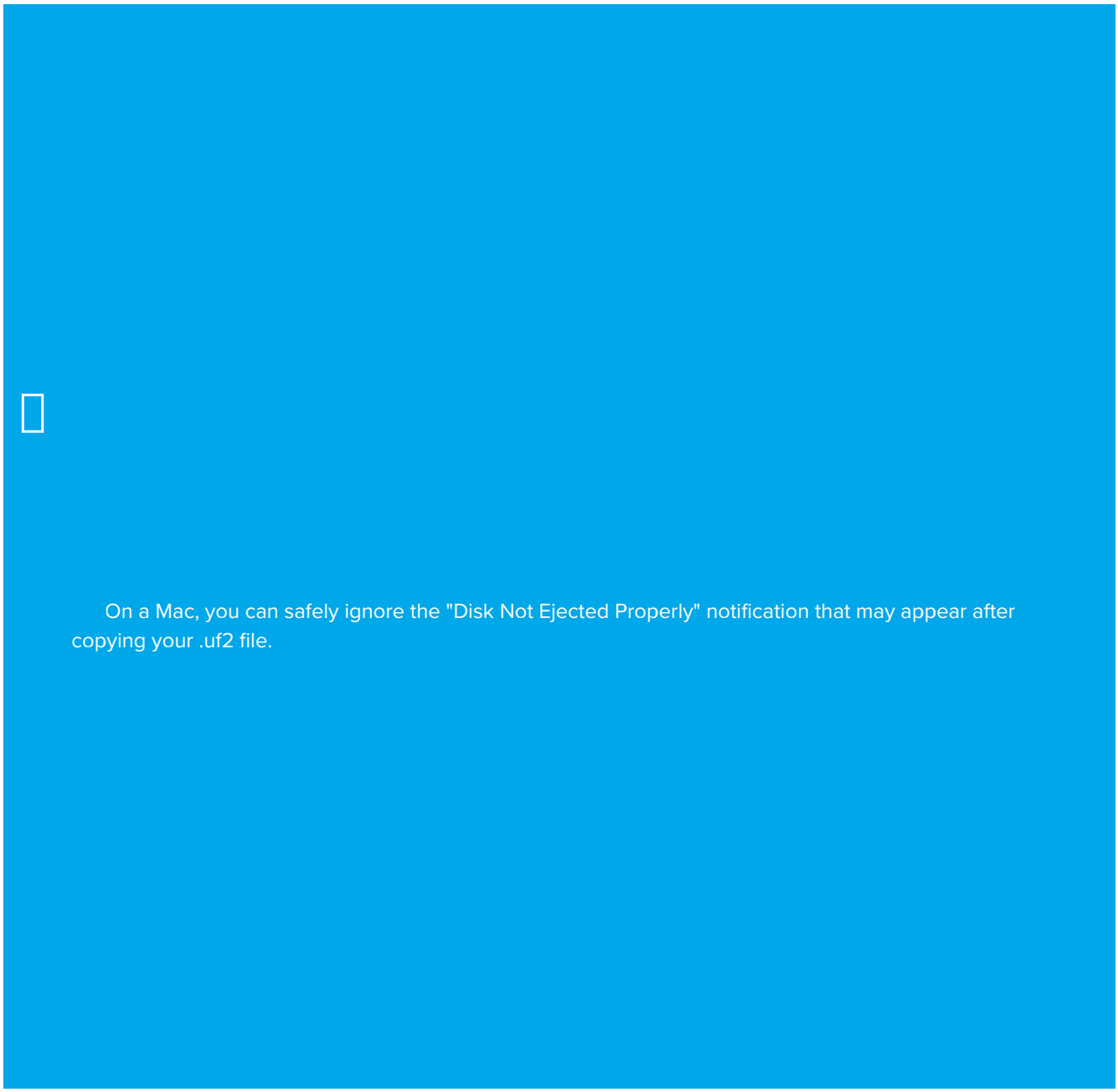


Step 3: Download and flash your code

If your board is working in the simulator, it's time to download it to your actual board! Click the **Download** button. It will generate a .uf2 file and download it to your computer. UF2 (https://adafru.it/vPE) is a file format designed by Microsoft to flash microcontrollers over USB.

General Steps to copy over your program (not specific to any Operating system)

* Ensure your board is connected via USB.
* Find the .uf2 file generated by MakeCode in your file explorer. Copy it to the **MAKECODE** or **boardnameBOOT** volume.
* The status LED on the board will blink while the file is transferring. Once it's done transferring your file, the board will automatically reset and start running your code (just like in the simulator!)



On a Mac, you can safely ignore the "Disk Not Ejected Properly" notification that may appear after copying your .uf2 file.

Saving and Sharing

Extracting your code from the board

The .uf2 file you created by clicking on the Compile button in MakeCode also contains the source code of your program!

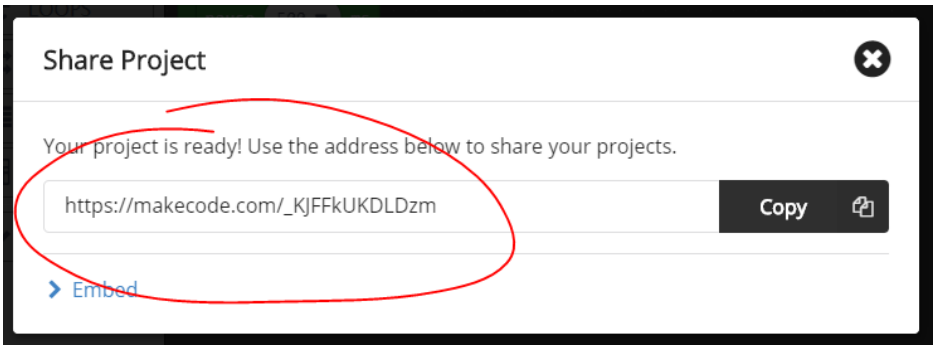
You can open this file in MakeCode by dragging and dropping it into the browser to edit it.

You can also find the current .uf2 file running on the **MAKECODE** or **boardnameBOOT** drive.

Sharing

You can share your code by clicking on the share button. After confirmation, MakeCode will create a short unique URL for your code. Anyone with that URL will be able to reload the code.

These URLs can also be used to embed the editor your blog or web pages! Just copy paste the URL in your text editor and (if it supports oEmbed) it will automatically load it in your page.



Custom Extensions

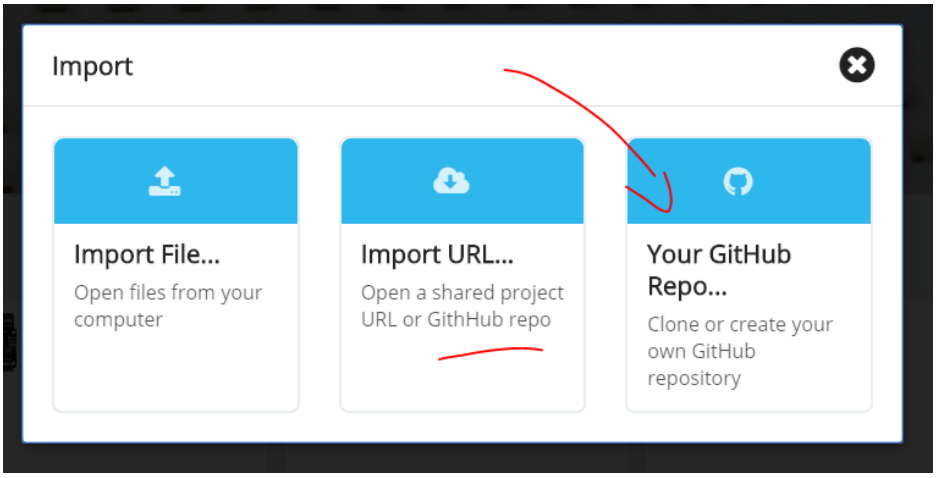
MakeCode allows to package and share code as **Extensions**. **Extensions** are stored as GitHub repositories and can be edited directly from the MakeCode editor.

Account setup

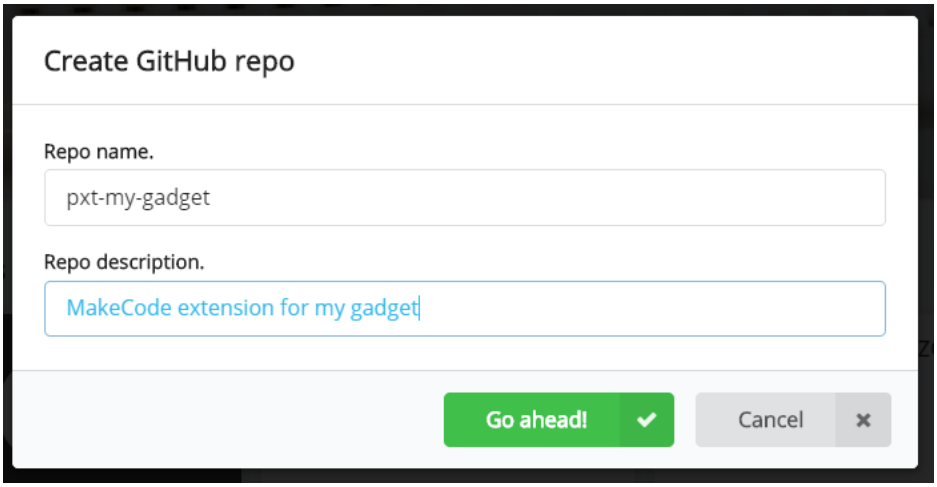
First, you need a GitHub account if you don't have one yet. GitHub is the largest host of source code in the world, with over 30 million users.

Once you have your account, you'll need to tie the MakeCode web app to your account. To do that, open any project in https://maker.makecode.com, go to the **Gear Wheel** menu on top, and select **Extensions**. At the bottom, there should be a link to log in to GitHub. A dialog will appear asking you to generate a GitHub token. Follow the instructions and paste the token into the dialog.

Once you've logged in, go back to the home screen. Now, the dialog that comes up after you press the **Import** button will have an additional option to list your GitHub repositories or create a new one.
Additionally, the **Import URL** option will now support `https://github.com/...` URLs, which is useful if you can't find your repository in the list (especially organizational repos), or as way to search the list faster using a copy/paste of the URL.

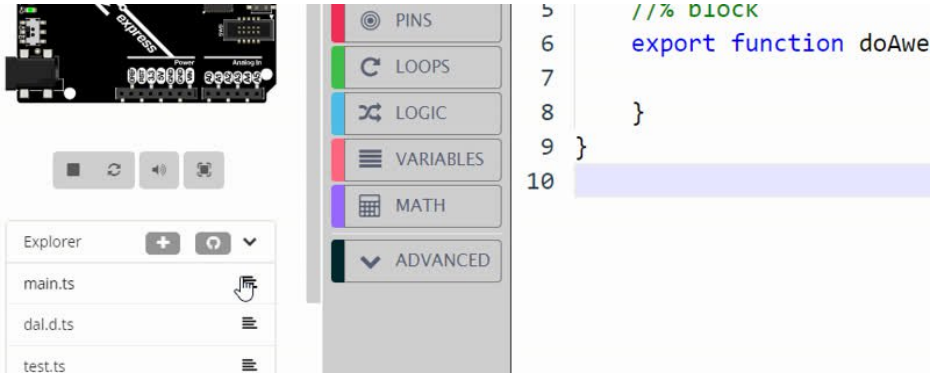


If you import a completely empty repo, or create a fresh one, MakeCode will automatically initialize it with `pxt.json` and other supporting files. If you import a non-empty repo without the `pxt.json` file, you will be asked if you want it initialized. Note that this might overwrite your existing files.



Commit and push

Once you have your repo set up, edit files as usual. Whenever you get to a stable state, or just every now and then to keep history and insure against losing your work, push the changes to GitHub. This is done with a little GitHub sync button on top of the **Explorer**. The button will check if there are any pending changes to check in. If there are, it will create a commit, pull the latest changes from GitHub, merge or fast-forward the commit if needed, and push the results to GitHub.

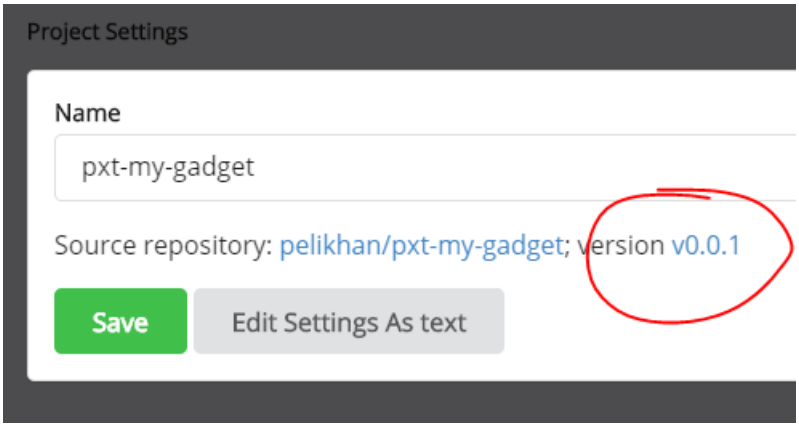


If there are changes, you will be asked for a commit message. Try to write something meaningful, like Fixed temperature reading in sub-freezing conditions or Added mysensor.readTemperature() function.

When describing changes, you are also given an option to **bump** the version number. This is a signal that the version you're pushing is stable and the users should upgrade to it. When your package is first referenced, the latest bumped version is used. Similarly, if there is a newer bumped version, a little upgrade button will appear next to the package. Commits without bump are generally not accessible to most users, so they are mostly for you to keep track of things.

We do not really distinguish between a commit, push, and pull - it all happens at once in the sync operation.

You can view a history of changes by following the version number link on the **Project Settings** page.



There is also another button next to the GitHub sync - you can use it to add new files to the project. This is mostly to help keep the project organized. For our TypeScript compiler it doesn't matter if you use one big file or a bunch of smaller ones.

Conflicts

It's possible that multiple people are editing the same package at the same time causing edit conflicts. This is similar to the situation where the same person edits the package using several computers, browsers, or web sites. In the conflict description below, for simplicity, we'll just concentrate on the case of multiple people working on the same package.

Typically, two people would sync a GitHub package at the same version, and then they both edit it. The first person pushes the changes successfully. When MakeCode tries to push the changes from the second person, it will notice that these are changes against a non-current version. It will create a commit based on the previous version and try to use the standard git merge (run server-side by GitHub). This usually succeeds if the two people edited different files, or at least different parts of the file - you end up with both sets of changes logically combined. There is no user interaction required in that case.

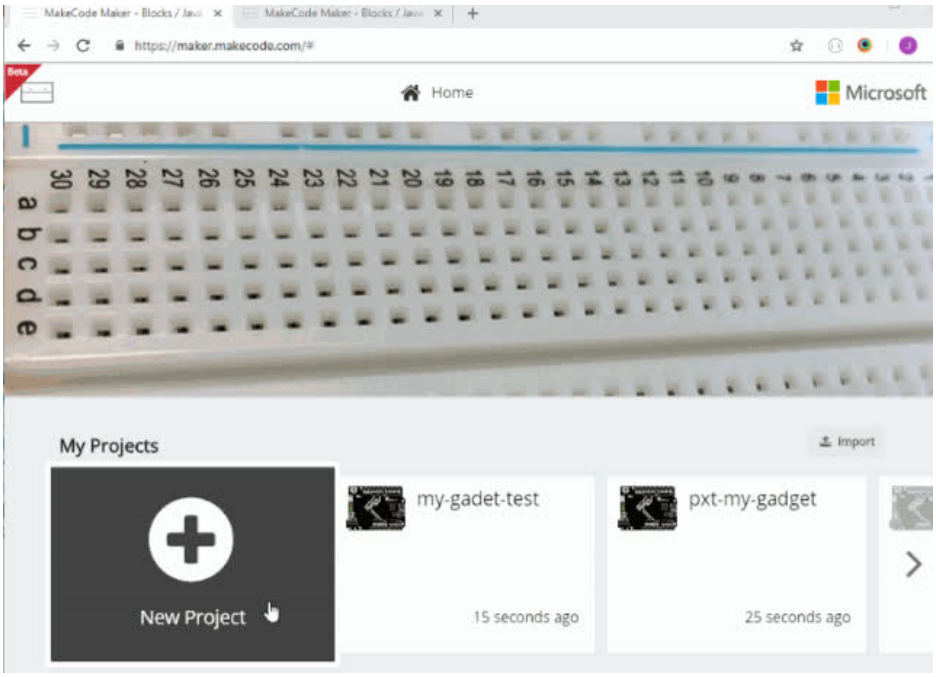
If the automatic merge fails, MakeCode will create a new branch, push the commit there, and then create a pull request (PR) on GitHub. The dialog that appears after this happens will let you go to the GitHub web site and resolve the conflicts. Before you resolve conflicts and merge the PR, the `master` branch will not have your changes (it will have changes from the other person, who managed to commit first). After creating the PR, MakeCode moves your local version to the `master` branch (without your changes), but don't despair they are not lost! Just resolve the conflict in GitHub and sync to get all changes back. MakeCode will also sync automatically when you close the PR dialog (presumably, after you resolved the conflict in another tab).

Testing your package

To test blocks in your package, press the **New Project** button on the home screen and go to the **Extensions** dialog. It will list all your GitHub projects as available for addition. Select your package and see what the blocks look like.

You can have one browser tab open with that test project, and another one with the package. When you switch between them, they reload automatically.

For testing TypeScript APIs you don't need a separate project, and instead can use the `test.ts` file in the package itself. It is only used when you run the package directly, not when you add it to a project. You can put TypeScript test code in there.


https://maker.makecode.com/#
Home
Microsoft
My Projects
Import
New Project
my-gadet-test
15 seconds ago
pxt-my-gadget
25 seconds ago