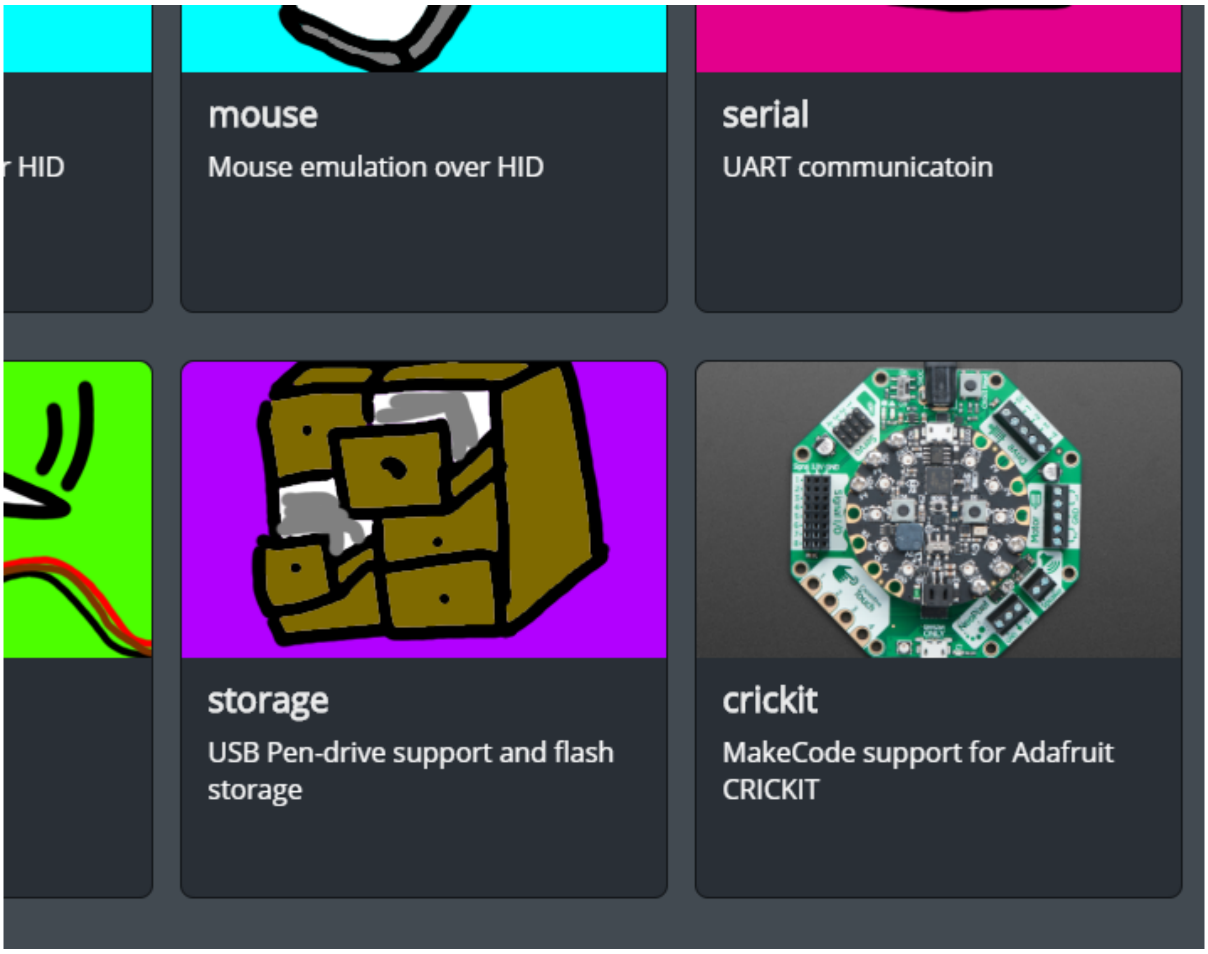
Getting Started with Custom Extensions for MakeCode

Created by Peli de Halleux


HID
mouse
Mouse emulation over HID
serial
UART communicatoin
storage
USB Pen-drive support and flash storage
crickit
MakeCode support for Adafruit CRICKIT

https://learn.adafruit.com/custom-extensions-for-makecode

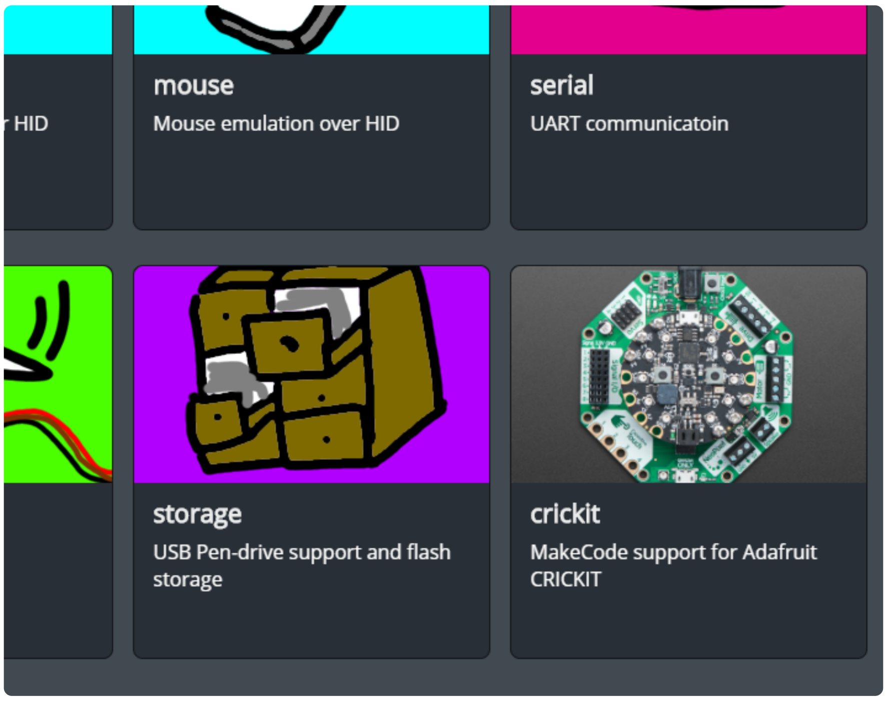
Last updated on 2024-06-03 02:42:52 PM EDT

Table of Contents

Introduction to Extensions

This guide will teach you how to package code and blocks for the MakeCode editors and surface them into the **Extensions** dialog.

Extensions are GitHub repositories that can be imported into MakeCode projects. Extensions can add blocks, JavaScript and even new editor functionalies.


r HID
mouse
Mouse emulation over HID
serial
UART communicatoin
storage
USB Pen-drive support and flash storage
crickit
MakeCode support for Adafruit CRICKIT

Who is this guide for?

You have been using MakeCode and you have some nifty code that you'd like to share across project or with other users.

MakeCode is a web-based code editor where you can program your Circuit Playground Express using drag-and-drop blocks or JavaScript.

- **If you are not familiar with MakeCode**, we recommend trying it out before starting.

While MakeCode supports block programming and JavaScript, extensions are exclusively authored in "JavaScript", which is really Static TypeScript.

- **If you are still learning about JavaScript in MakeCode**, we recommend you write your program in block code, switch to JavaScript, and try learning this language one step at a time :).

Extensions are stored as repositories on GitHub. You do not need to be a Git guru to use this!

- **You will need a GitHub account (it's free)**

Creating your Extension

GitHub Account

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.

GitHub Token

Once you have your account, you'll need to tie the MakeCode web app to your account. To do that,

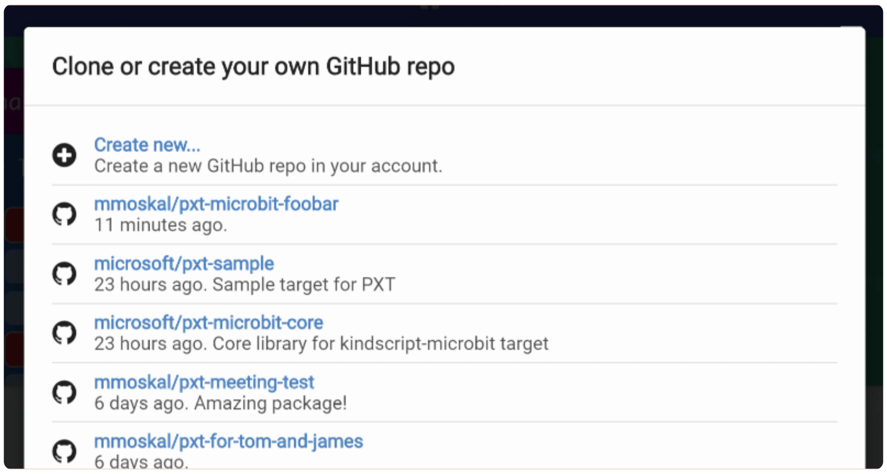
- open any project
- go to the **Gear Wheel** menu on top, and select **Extensions**
- at the bottom, click on the link to log in to GitHub
- A dialog will appear asking you to generate a GitHub token. Follow the instructions and paste the token into the dialog.

Create a new Repository

Once you've stored your token,

- go back to the home screen
- 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.



Clone or create your own GitHub repo

Create new...
Create a new GitHub repo in your account.

mmoskal/pxt-microbit-foobar
11 minutes ago.

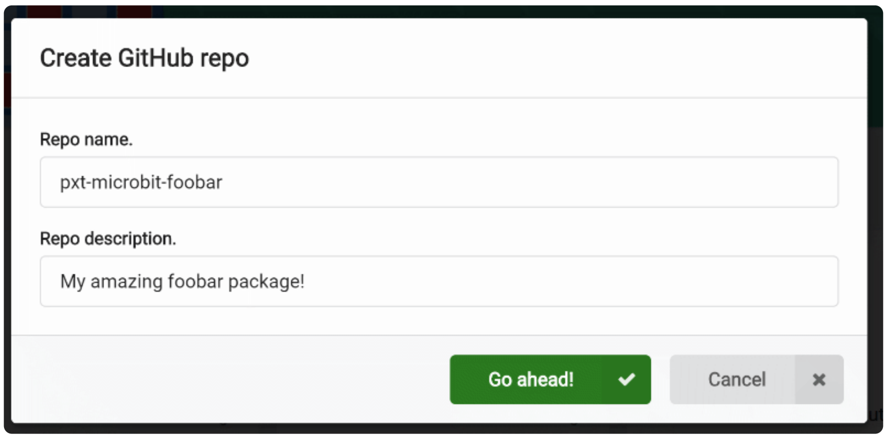
microsoft/pxt-sample
23 hours ago. Sample target for PXT

microsoft/pxt-microbit-core
23 hours ago. Core library for kindscript-microbit target

mmoskal/pxt-meeting-test
6 days ago. Amazing package!

mmoskal/pxt-for-tom-and-james
6 days ago.

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.



Create GitHub repo

Repo name.
pxt-microbit-foobar

Repo description.
My amazing foobar package!

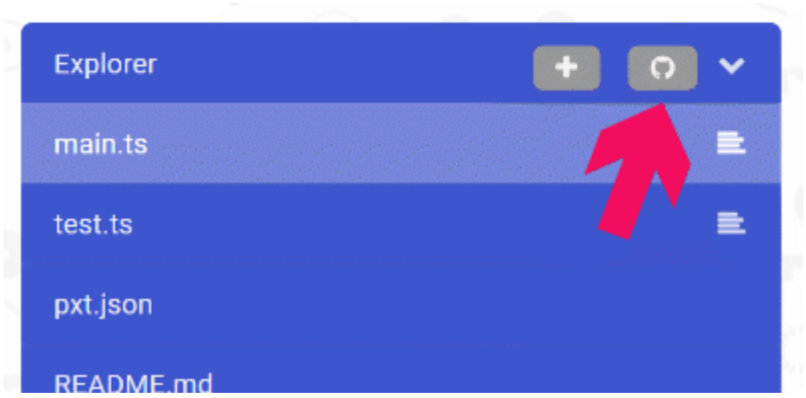
Currently, there is no way to push an existing project into GitHub. As a workaround, create a new project and copy/paste the contents of the `main.ts` file.

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.

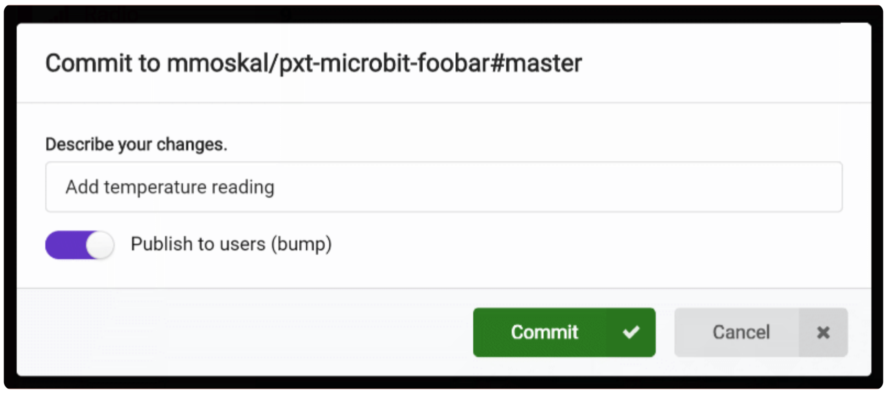
- click on the a little GitHub sync button on top of the **Explorer**.

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 extension 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 extension. Commits without bump are generally not accessible to most users, so they are mostly for you to keep track of things.


Commit to mmoskal/pxt-microbit-foobar#master

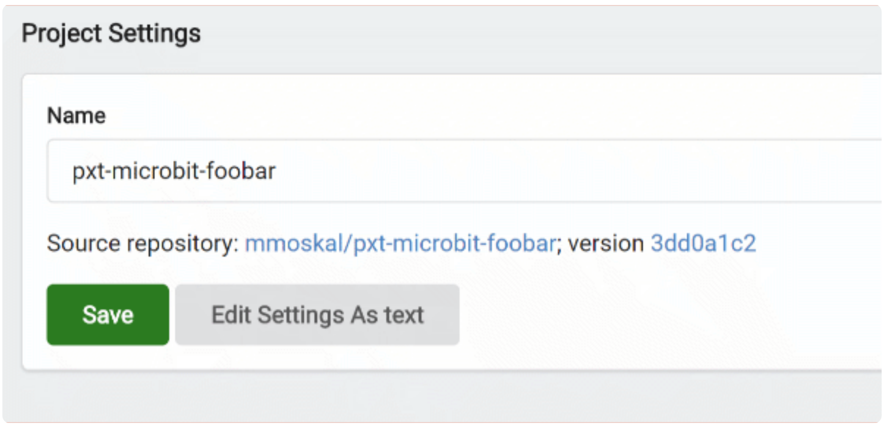
Describe your changes.

Add temperature reading

Publish to users (bump)

There's really no distinguishing 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.


Project Settings

Name

pxt-microbit-foobar

Source repository: mmoskal/pxt-microbit-foobar; version 3dd0a1c2

There's 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 the 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 extension at the same time, causing edit conflicts. This is similar to the situation where the same person edits the extension 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 extension.

Typically, two people would sync a GitHub extension 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.

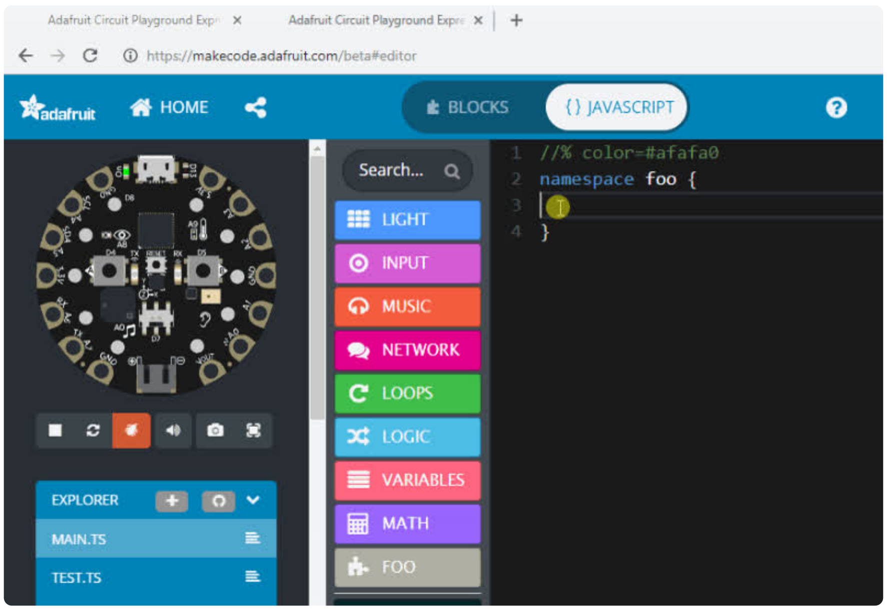
If the automatic merge fails, MakeCode will create a new branch, push the commit there, and then create a pull request (PR) on GitHub. The dialog that appears after this happens will let you go to the GitHub web site and resolve the conflicts. Before you resolve conflicts and merge the PR, the `master` 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).

Test your Extension

To test blocks in your extension, 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 extension and see what the blocks look like.

The Genius Trick

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



```
//% color=#afafa0
namespace foo {

}
```

Tests

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

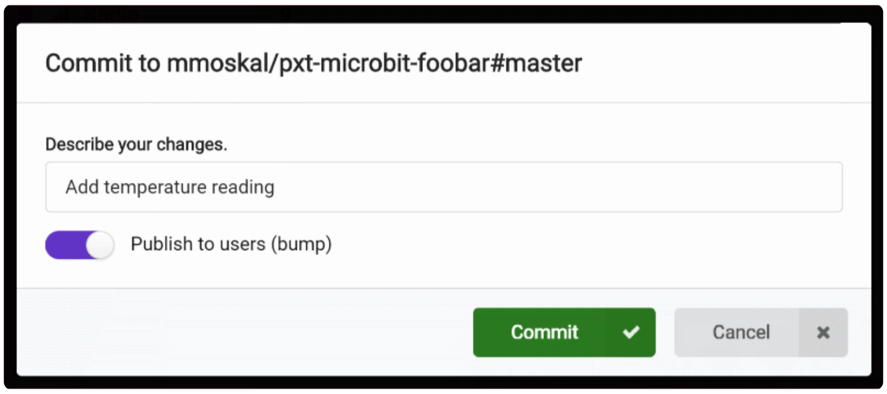
Limitations

The web app will not let you create extensions with C++. This you still need to do from command line, after installing all required compilers or Docker (depending on target). The good news is that very few extensions contain C++ (mostly because TypeScript is easier to write, test, and in most cases sufficient). The main reason seen so far for needing C++ extensions is the lack of floating point support on the micro:bit (this is now fixed with the v1 release).

Publishing Your Extension

Bumping

When clicking on the **GitHub sync** button, you are also given an option to **bump the version number.**


Commit to mmoskal/pxt-microbit-foobar#master

Describe your changes.

Add temperature reading

Publish to users (bump)

Bump is a signal that the version you're pushing is stable and the users should upgrade to it.

When your extension 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 extension.

Commits without bump are generally not accessible to most users, so they are mostly for you to keep track of things.

Approval and Extensions dialog

In order to appear in the Extensions dialog -- or in search result, you need to get your Extension approved. In order to get approved, open an issue on GitHub at https://github.com/Microsoft/pxt-adafruit/issues.

Custom Blocks

In MakeCode, all blocks are defined from a Static TypeScript function or property using some annotations in the comment. MakeCode takes care of handling the conversion to Google Blockly and back.

Your First Block!

Let's start with a simple statement block that blinks an LED. Place this code in **main.ts** of your Extension.

```
/**
* Cool LED functionalities
*/
namespace ledTricks {
    /**
    * Blinks the LED once.
    */
    //% block
    export function blink() {
        pins.LED.digitalWrite(true)
        pause(500)
        pins.LED.digitalWrite(false)
        pause(500)
    }
}
```

If you switch tab to your test project, you will see a `LED TRICKS` category in the toolbox with a single `blink` block. Let's break it down line by line:

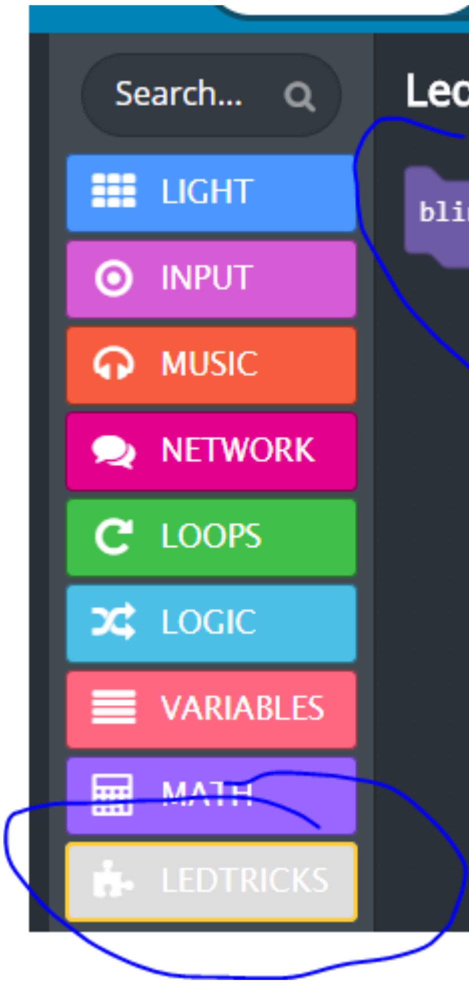
```
/**
* Cool LED functionalities
*/
namespace ledTricks {
```

The name of namespacem, `ledTricks` is used to organize your blocks, just like it organizes your JavaScript functions.

```
    /**
    * Blinks the LED once.
    */
    //% block
    export function blink() {
```

The `//% block` comment tells MakeCode to "mount" a block for this function. MakeCode will infer the shape of the block by looking at the inputs and return value of the function. In this case, this function does not have any input or output.

The descriptive comment is also mined by MakeCode and injected in the tooltip of the block.



That's it! You've just created a new block!!!

More on blocks

There is actually a ton of options and knobs to beautify the blocks shown in MakeCode editors. The best place to learn about them is the MakeCode Playground.

The complete docs on this topic is available at https://makecode.com/defining-blocks .

Additional Notes

- You can use a non-`master` branch by going to **Import URL** and saying something like `https://github.com/jrandomhacker/pxt-mypkg#mybranch`. User note, this hasn't been extensively tested yet.

- MakeCode will generally only download files listed in `pxt.json`. Files in GitHub but not in `pxt.json` will be ignored and left alone.

More Information

- Extensions documentation
- Static TypeScript Reference