Contributing

If you find an error or have a comment, please start a discussion by submitting an issue on our repositories!

- https://github.com/montoyamoraga/tiny-trainable-instruments/issues
- https://github.com/montoyamoraga/TinyTrainable/issues

Tools

clang-format

Tool for automation of formatting to source code. More information here: https://clang.llvm.org/docs/.ClangFormat.html

Doxygen

Tool for generating documentation from the source code. More information: https://www.doxygen.nl/.

GitHub Actions

Every time we push code to the TinyTrainable repositories, a GitHub action creates a virtual machine, and runs a script to generate the Doxygen documentation and push it to the gh-pages branch.

Jupyter

Jupyter is a free, open-source browser application that allows users to easily read and write code in a clean, accessible environment. Code is segmented into cells, which users can execute individually by clicking into and selecting the triangle "play" button at the top. Subsequent code executes based on operations done in previous cells. Basically, Jupyter notebooks allow programmers to create clean, step-by-step interactive walkthroughs through their code. More information: https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/index.html.

Markdown

Markdown is a lightweight markup language with simple, intuitive syntax. Aside from a few key differences, it is largely the same as plaintext. The documentation of this project is written using Markdown, including this document! More info: https://guides.github.com/features/mastering-markdown/.

GitHub instructions

To contribute to this repository:

1. Create a free GitHub account
2. Fork the repository
3. Clone your new repository to your computer

```
git clone https://github.com/your_username/tiny-trainable-instruments.git
```

Optionally, you can also clone the submodules of this repository, with the command

```
git submodule update --init --recursive
```

4. Change directory (cd) into the project folder

```
cd tiny-trainable-instruments
```

5. Make some changes

6. Stage and make a commit to your repository on your computer

```
git add .
```

```
git commit -m "your comment"
```

7. Push your commit to your personal fork on GitHub

```
git push
```

8. Open your repository online

9. Open a pull request and wait for comments or approval

For subsequent pull requests:

If your fork is behind origin:

1. Open a pull request, but reverse the order (merge main into your fork)

2. Approve the merge to update your fork

3. Pull your repository to your computer

```
```

```
git pull
```

4. Continue from step 4 above

Contributing documentation

For more information about how to contribute documentation to an open source artistic project, we recommend looking at the documentation by the p5.js project, available at https://github.com/processing/p5.js/blob/main/contributor_docs/contributing_documentation.md

Adding submodules

If you think there are more repositories we should include as submodules for archival purposes, use the following command, replacing GITPATH with the location of the repository you want to include, and FOLDERPATH with the destination.

```
git submodule add GITPATH FOLDERPATH
```

Helper scripts

The helper scripts are located on the assets/ folder. To run them, cd to assets/ and then use the following commands

Compiling code

This script uses arduino-cli for checking the compilation of all examples of TinyTrainable.

```
sh compile-code.sh
```

Delete metadata

This script uses exiftool to delete the metadata of all pictures in docs/

```
sh delete-metadata.sh
```

Format code

This script uses clang-format to organize all the code in TinyTrainable

```
```

```
sh format-code.sh
```

Markdown to PDF

This script uses pandoc to convert the documentation from Markdown to PDF format.

```
sh markdown-to-pdf.sh
```

Files

Files with the extension .h are source code written in C++ and they include the definition and declaration of the classes, methods, and variables.

Files with the extension .cp are source code written in C++ that implement the corresponding .h file.

Arduino sketches have the extension .ino.

Including libraries

There are 2 ways of including libraries, with <> or "". If you use "" the file is searched on the directory, and if you use <> you are asking Arduino to also check its libraries folder.

```
#include <Arduino.h>
#include "Arduino.h"
```

Data structures

```
// using a template datatype allows debugPrint to take in any datatype, like
// Serial.println(). it needs to be defined here in the header file so it
// compiles before anything else, since it's called in the Inst0.cpp file
template <typename T> void debugPrint(T message) {
      if (_serialDebugging) {
        Serial.println(message);
      }
    };
```

C++

We have a class called TinyTrainable, and subclasses for each of the other instruments.

We declare certain methods as protected instead of private, because if they were private, all the instrument classes that inherit TinyTrainable wouldn't have access to these anymore. If we want to make them private we'd have to make public getter/setter methods.

static void vs void

Peter's explanation: i used "static void" because you only need to run it once per arduino even if you have multiple instances of the instrument classes (it doesn't need to be an instance method).