Infrared4Arduino — Yet another infrared library for the Arduino

Table of contents

Date	Description
2016-05-04	Initial version.

Table 1: Revision history

1 Introduction

This is yet another infrared library for the Arduino. (Although its name does not start with YA...) It is a major rewrite of Chris Young's IRLib, (GitHub repo), which itself is a major rewrite of a library called *IRremote*, published by Ken Shirriff in his blog, now maintained on GitHub. It uses Michael Dreher's IrWidget (article in German), see also this forum contribution.

The classes `IrWidget` and `IrWidgetAggregating` are based on Michael's code. The classes `IrReceiverSampler` and `IrSenderPwm`, and in particular the file `IRremoteInt.h`, are adapted from Kevin's and Chris' work. The remaining files are almost completely written from scratch, although the influence of Kevin and Chris is gratefully acknowledged.

This work is a low-level library (like IRremote and IRLib) that can be used in other projects, like AGirs, which is an high-level program taking commands interactively from a user or a program through a bi-directional `Stream`. The goal has been to provide a sound, object oriented basis for the fundamental basis, not to provide maximal functionality, the maximal number of protocols supported, or the most complete support of different hardware. A clean design and high readability, without being "too" inefficient, has been the highest priority. Dynamic memory allocation with `new` and `delete` is used extensively. The user who is afraid of this can create his required objects at the start of the run, and keep them. Most classes are immutable. The classes are `const`-correct.

2 API

2.1 Types

There are some project specific data typedefs in `InfraredTypes.h`. For durations in microseconds, the data type `microseconds_t` is to be used. If desired/necessary, this can be either `uint16_t` or `uint32_t`. For durations in milliseconds, use the type `millisecons_t`. Likewise, use `frequency_t` for modulation frequency in Hz (*not* kHz as in the IRremote/IRLib).

For "sizes", `size_t`, the standard C type, is used.

Implementation dependent types like `int` are used if and *only if* it is OK for the compiler to select any implementation allowed by the C++ language.

`unsigned int` is used for quantities that can "impossibly" be larger than 65535.

2.2 IrSequences and IrSignals

An `IrSequence` is a vector of durations, i.e. sequence of interleaving gaps and spaces. It does not contain the modulation frequence. As opposed to IRremote and IRLib, our sequences always start with a space and end with a gap. It is claimed to be a more relevant representation than the one of IRremote and IRLib.

An `IrSignal` consists of a modulation frequency and three `IrSequence`s: intro-, repeat-, and ending sequence. All of these, but not all, can be empty. If repeat is empty, intro has to be non-empty and ending empty. The intro sequence is always sent first, then comes a zero or more repeat sequences, and finally the ending sequence. To send a signal $n > 0$ times shall mean the following: If the intro is non-empty, send intro, n - 1 repeats, and then the ending. If the intro is empty, send n repeats, and then then ending.

2.3 Class construction

For some receiving and transmitting classes, multiple instantiations are not sensible, for other it may be. In this library, the classes that should only be instantiated once are implemented as singleton classes, i.e. with no public constructor, but instead a static "factory" method (`newThing()`) that delivers a pointer to a newly constructed instance of `Thing`, provided that `Thing` has not been instantiated before. The classes, where multiple instances is sensible, come with public constructors. (However, the user still has to take responsibility for avoiding pin- and timer-conflicts.)

2.4 Hardware configuration

For hardware support, the file `IRremoteInt.h` from the IRremote project is used. This means that all hardware that project supports is also supported here (for `IrReceiverSampler` and `IrSenderPwm`). (Actually, a small fix, borrowed from IRLib, was used to support Arduinos with ATMega32U4 (Leonardo, Micro).) However, IrWidgetAggregating is currently supported on the boards Uno/Nano (ATmega328P), Leonardo/Micro (ATmega32U4), and Mega2560 (ATmega2560).

Several of the sending and receiving classes take a GPIO pin as argument to the constructor. However, the sending pin of `IrSenderPwm` and the capture pin of `IrWidgetAggregating` are not configurable, but (due to hardware limitations) have to be taken from the following table:

	Sender Pin	Capture Pin
Uno/Nano (ATmega328P)	3	8
Leonardo/Micro (ATmega32U4)	9	4
Mega2560 (ATmega2560)	9	49

3 Timeouts

All the receiving classes adhere to the following conventions: When initialized, it waits up to the time `beginningTimeout` for the first on-period. If not received within that period, it returns with a timeout. Otherwise, is starts collecting data. It will collect data until one of the following occurs:

- A silence of length `endingTimeout` has been detected. This is the normal ending. The detected last gap is returned with the data.
- The buffer gets full. Reception stops.

4 User parameters

As opposed to other infrared libraries, there are no user changeable parameters as CPP symbols. However, the timer configuration is compiled in, depending on the CPP processors given to the compiler, see the file `IRremoteInt.h`.

5 Files

As opposed to the predecessor projects, this project has a header (`*.h`) file and an implementation file (`*.cpp`, sometimes missing) for each public class.

6 Error handling

Simple answer: there is none. If a function is sent erroneous data, it just silently ignores the request, or does something else instead. This (unfortunately) seems to be the standard procedure in Arduino programming.

I am used to exception based error handling, for some reason this is not used by the Arduino community.

Constructive suggestions are welcome.

7 Protocols

Comparing with the predecessor works, this project may look meager, currently supporting only two protocols (NEC1 and RC5). It is planned to generate the corresponding C++ code automatically from the IRP notation. (For this reason, contributed implementations of more protocols are not solicited.)

8 Sending non-modulated signals.

RF signals (433 MHz and other carrier frequencies) do not use the IR typical modulation. Also there are a few IR protocols (like Revox, Barco, Archer) that do not using modulation. These signals can be sent by the class `IrSenderNonMod`, after connecting suitable hardware capable of sending non-modulated (IR- *or* RF-) signals to the GPIO pin given as argument to the constructor.

9 Dependencies

This library does not depend on any other libraries; only the standard Arduino environment.

10 Questions and answers

10.1 What is the difference between the IrReceiver* and the IrWidget* classes?

They are intended for two different use cases, receiving and capturing. Differently put, "receive" uses a demodulating receiver (TSOPxxx, etc.), "capture" a non-demodulating decoder (TSMPxxx, OPLxxx QSExxx, etc.). Note that this terminology is not universally accepted (yet!).

11 Coding style

My goal is to write excellent code, even though I do not always succeed :-). "Everything as simple as possible, but not simpler." Cleanliness, logical structure, readability and maintainability are the most important requirements. Efficiency (runtime and/or space) is also important, although it normally comes on second place. [The Arduino Style Guide](https://www.arduino.cc/en/Reference/APIStyleGuide) has different goals (essentially optimizing for novice programmers, "Some of these run counter to professional programming practice"). It is therefore not given priority in this project.

12 Documentation

The main documentation for the classes is found in the source files themselves. It can be extracted to a browse-able documentation using the program Doxygen. After installing the program, fire up the program in the source directory. It will generate documentation in a subdirectory `apidoc`. To browse, open `apidoc/index.html` in a browser.

The documentation is written for the *user* of the library, not the developer. For this reason, the file `Arduino.h` has been deliberately excluded from the documentation, to keep it centered on the main issues for the programming on the target system.

13 Multi platform coding

For someone used to, e.g., Netbeans or Eclipse, the Arduino IDE feels "somewhat" primitive and limited. In particular, it does not support debugging. Mainly for this reason, the code in the present library is designed to compile, and at least to some extent, run in a normal C++ environment on the host compiler. For this, some code modifications, in particular, a customized `Arduino.h` was needed. If the preprocessor symbol `ARDUINO` is defined, just includes the standard Arduino `Arduino.h` is included, otherwise (i.e. for compiling for the host), some more-or-less dummy stuff are defined, allowing compiling for, and execution/debugging on the host.

This way, certain types of problems can be solved much faster. The drawback is that the code is "polluted" with ugly `#ifdef ARDUINO` statements, which decreases readability and makes maintenance harder.

The subdirectory `tests` contains test(s) that run on the host. The supplied `Makefile` is intended for compiling for the host as target. It creates a library in the standard sense (`*.a`), and can be used to build and run tests in subdirectory `tests`.

With the provided `Doxyfile`, Doxygen will document only the (strict) Arduino parts, not the "portable C++".

14 License

The entire work is licensed under the GPL3 license. Chris' as well as Ken's code is licensed under the LGPL 2.1-license. Michael's code carries the GPL2-license, although he is willing to agree to "or later versions".

15 Links

The GitHub repository.

API documentation.