Arduboy2 Library

Generated by Doxygen 1.8.11

Contents

Chapter 1

Arduboy2 Library

The Arduboy2 library is maintained in a git repository hosted on GitHub at:

https://github.com/MLXXXp/Arduboy2

The **Arduboy2** library is a fork of the Arduboy library, which provides a standard *application programming interface* (API) to the display, buttons and other hardware of the Arduino based Arduboy miniature game system.

The name *Arduboy2* doesn't indicate that it's for a new "next generation" of the Arduboy hardware. The name was changed so it can coexist in the Arduino IDE with the current *Arduboy* library, without conflict. This way, existing sketches can continue to use the *Arduboy* library and class, without changes, while new sketches can be written (or old ones modified) to use and take advantage of the capabilities of the *Arduboy2* class and library.

For notes on the differences between the *Arduboy2* library and the original *Arduboy* library, and for information on migrating a sketch currently using the *Arduboy* library, see the sections at the end of this document.

Library documentation

Comments in the library header files are formatted for the Doxygen document generation system. The HTML files generated using the configuration file *extras/Doxyfile* can be found at:

https://MLXXXp.github.io/documents/Arduino/libraries/Arduboy2/Doxygen/html/index.html

A generated PDF file can be found at:

https://MLXXXp.github.io/documents/Arduino/libraries/Arduboy2/Doxygen/pdf/Arduboy2.pdf

Installation

The Arduboy2 library can be installed using the Arduino IDE Library Manager:

- In the Arduino IDE select from the menus: `Sketch > Include Library > Manage Libraries...`
- In the Library Manager *Filter your search...* field enter *arduboy2*.
- Click somewhere within the Arduboy2 entry.
- Click on the *Install* button.

For more library installation information see

Installing Additional Arduino Libraries - Using the Library Manager

Start up features

The *begin()* function, used to initialize the library, includes features that are intended to be available to all sketches using the library (unless the sketch developer has chosen to disable one or more of them to free up some code space):

The boot logo

At the start of the sketch, the **ARDUBOY** logo scrolls down from the top of the screen to the center.

The RGB LED lights red then green then blue while the logo is scrolling. (If your Arduboy is one of those that has the RGB LED installed incorrectly, then it will light blue then off then red). For users who do not wish to have the RGB LED flash during the boot logo sequence, a flag can be set in system EEPROM to have it remain off. The included *SetSystemEEPROM* example sketch can be used to set this flag.

A user settable *unit name* of up to 6 characters can be saved in system EEPROM memory. If set, this name will be briefly displayed at the bottom of the boot logo screen, after the logo stops scrolling down. This feature is only available if the *Arduboy2* class is used, not the *Arduboy2Base* class. This is because it requires the text display functions, which are only available in the *Arduboy2* class. A flag in system EEPROM controls whether or not the *unit name* is displayed on the boot logo screen, regardless of whether the *unit name* itself has been set. The included *SetSystemEEPROM* example sketch can be used to set both the *unit name* and this flag.

Once the logo display sequence completes, the sketch continues.

For developers who wish to quickly begin testing, or impatient users who want to go strait to playing their game, the boot logo sequence can be bypassed by holding the *RIGHT* button while powering up, and then releasing it. Alternatively, the *RIGHT* button can be pressed while the logo is scrolling down.

For users who wish to always disable the displaying of the boot logo sequence on boot up, a flag in system EEPROM is available for this. The included *SetSystemEEPROM* example sketch can be used to set this flag.

"Flashlight" mode

If the *UP* button is pressed and held when the Arduboy is powered on, it enters *flashlight* mode. This turns the RGB LED fully on, and all the pixels of the screen are lit, resulting in a bright white light suitable as a small flashlight. (For an incorrect RGB LED, only the screen will light). To exit *flashlight* mode the Arduboy must be restarted.

Flashlight mode is also sometimes useful to allow uploading of new sketches, in case the sketch currently loaded uses a large amount of RAM which creates a bootloader problem.

Audio mute control

Pressing and holding the *B* button when powering on will enter *System Control* mode. The RGB LED will light blue (red for an incorrect LED) to indicate that you are in *system control* mode. You must continue to hold the *B* button to remain in this mode. The only *system control* function currently implemented is *audio mute control*.

Pressing the *UP* button (while still holding *B*) will set a flag in system EEPROM indicating *audio enabled*. The RGB LED will flash green once (off for an incorrect LED) to indicate this action.

Pressing the *DOWN* button (while still holding *B*) will set the flag to *audio disabled* (muted). The RGB LED will flash red once (blue for an incorrect LED) to indicate this action.

Releasing the *B* button will exit *system control* mode and the sketch will continue.

Note that the audio control feature only sets a flag in EEPROM. Whatever code actually produces the sound must use the *audio.enabled()* function to check and honor the mute state. Audio libraries written with the Arduboy system in mind, such as the available *ArduboyPlaytune* and *ArduboyTones*, should do this. However, be aware that for some sketches, which don't use the Arduboy2 or other compliant library and generate sounds in their own way, this method of muting sound may not work.

Using the library in a sketch

As with most libraries, to use Arduboy2 in your sketch you must include its header file at the start:

```
#include <Arduboy2.h>
```

You must then create an Arduboy2 class object:

```
Arduboy2 arduboy;
```

Naming the object *arduboy* has become somewhat of a standard, but you can use a different name if you wish.

To initialize the library, you must call its *begin()* function. This is usually done at the start of the sketch's *setup()* function:

```
void setup()
{
  arduboy.begin();
  // more setup code follows, if required
}
```

The rest of the Arduboy2 functions will now be available for use.

If you wish to use the Sprites class functions you must create a Sprites object:

```
Sprites sprites;
```

Sample sketches have been included with the library as examples of how to use it. To load an example, for examination and uploading to the Arduboy, using the Arduino IDE menus select:

`File > Examples > Arduboy2`

More information on writing sketches for the Arduboy can be found in the `Arduboy Community Forum`.

Using EEPROM in a sketch

The Arduboy2 library reserves an area at the start of EEPROM for storing system information, such as the current audio mute state and the Unit Name and Unit ID. A sketch **must not** use this reserved area for its own purposes. A sketch may use any EEPROM past this reserved area. The first EEPROM address available for sketch use is given as the defined value *EEPROM_STORAGE_SPACE_START*

Audio control functions

The library includes an Arduboy2Audio class. This class provides functions to enable and disable (mute) sound and also save the current mute state so that it remains in effect over power cycles and after loading a different sketch. It doesn't contain anything to actually produce sound.

The Arduboy2Base class, and thus the Arduboy2 class, creates an Arduboy2Audio class object named *audio*, so a sketch doesn't need to create its own Arduboy2Audio object.

Example:

```
#include <Arduboy2.h>

Arduboy2 arduboy;

// Arduboy2Audio functions can be called as follows:
  arduboy.audio.on();
  arduboy.audio.off();
```

Simple tone generation

The *BeepPin1* and *BeepPin2* classes are available to generate simple square wave tones using speaker pin 1 and speaker pin 2 respectively. These classes are documented in file *Arduboy2Beep.h*. Also, *BeepDemo* is included as one of the example sketches, which demonstrates basic use.

NOTE: These functions will not work with a DevKit Arduboy because the speaker pins used cannot be directly controlled by a timer/counter. "Dummy" functions are provided so a sketch will compile and work properly but no sound will be produced.

Ways to make more code space available to sketches

Sound effects and music

If all you want is to play single tones, using the built in *BeepPin1* or *BeepPin2* classes will be very efficient.

If you want to be able to play sequences of tones or background music, using the `ArduboyTones` library will be more code efficient than using `ArduboyPlaytune` or most other sound libraries compatible with the Arduboy. *ArduboyTones* even produces less code than the `Arduino built in tone() function`. You'll have to decide on the appropriate library or functions you use to generate sound, based on the features required and how much memory you want it to use.

Remove the text functions

If your sketch doesn't use any of the functions for displaying text, such as *setCursor()* and *print()*, you can remove them. You could do this if your sketch generates whatever text it requires by some other means. Removing the text functions frees up code by not including the font table and some code that is always pulled in by inheriting the `Arduino Print class`.

To eliminate text capability in your sketch, when creating the library object simply use the *Arduboy2Base* class instead of *Arduboy2*:

For example, if the object will be named *arduboy*:

Replace

```
Arduboy2 arduboy;
```

with

```
Arduboy2Base arduboy;
```

Remove boot up features

As previously described, the *begin()* function includes features that are intended to be available to all sketches during boot up. However, if you're looking to gain some code space, you can call *boot()* instead of *begin()*. This will initialize the system but not include any of the extra boot up features. If desired, you can then add back in any of these features by calling the functions that perform them. You will have to trade off between the desirability of having a feature and how much memory you can recover by not including it.

A good way to use *boot()* instead of *begin()* is to copy the code from the body of the *begin()* function, in file *Arduboy2.cpp*, into your sketch and then edit it to retain the *boot()* call and any feature calls desired.

As of this writing, the begin function is:

```
void Arduboy2Base::begin()
{
  boot(); // raw hardware

  display(); // blank the display (sBuffer is global, so cleared automatically)

  flashlight(); // light the RGB LED and screen if UP button is being held.

  // check for and handle buttons held during start up for system control
  systemButtons();

  audio.begin();

  bootLogo();

  waitNoButtons(); // wait for all buttons to be released
}
```

To incorporate it into your sketch just keep *boot()* and whatever feature calls are desired, if any. Comment out or delete the rest. Remember to add the class object name in front of each function call, since they're now being called from outside the class itself. If your sketch uses sound, it's a good idea to keep the call to *audio.begin()*.

For example: Let's say a sketch has its own code to enable, disable and save the *audio on/off* setting, and wants to keep the *flashlight* function. In *setup()* it could replace *begin()* with:

```
  arduboy.boot(); // raw hardware

// *** This particular sketch clears the display soon, so it doesn't need this:
//  display(); // blank the display (sBuffer is global, so cleared automatically)

  arduboy.flashlight(); // light the RGB LED and screen if UP button is being held.

  // check for and handle buttons held during start up for system control
//  systemButtons();

  arduboy.audio.begin();

//  bootLogo();

//  waitNoButtons(); // wait for all buttons to be released
```

This saves whatever code *display()*, *systemButtons()*, *bootLogo()* and *waitNoButtons()* would use.

There are a few functions provided that are roughly equivalent to the standard functions used by *begin()* but which use less code space.

- *bootLogoCompressed()*, *bootLogoSpritesSelfMasked()*, *bootLogoSpritesOverwrite()*, *bootLogoSpritesBSelfMasked()* and *bootLogoSpritesBOverwrite()* will do the same as *bootLogo()* but will use *drawCompressed()*, or *Sprites* / *SpritesB* class *drawSelfMasked()* or *drawOverwrite()* functions respectively, instead of *drawBitmask()*, to render the logo. If the sketch uses one of these functions, then using the boot logo function that also uses it may reduce code size. It's best to try each of them to see which one produces the smallest size.
- *bootLogoText()* can be used in place *bootLogo()* in the case where the sketch uses text functions. It renders the logo as text instead of as a bitmap (so doesn't look as good).
- *safeMode()* can be used in place of *flashlight()* for cases where it's needed to allow uploading a new sketch when the bootloader "magic key" problem is an issue. It only lights the red RGB LED, so you don't get the bright light that is the primary purpose of *flashlight()*.

Use the SpritesB class instead of Sprites

The *SpritesB* class has functions identical to the *Sprites* class. The difference is that *SpritesB* is optimized for small code size rather than execution speed. If you want to use the sprites functions, and the slower speed of *SpritesB* doesn't affect your sketch, you may be able to use it to gain some code space.

Even if the speed is acceptable when using *SpritesB*, you should still try using *Sprites*. In some cases *Sprites* will produce less code than *SpritesB*, notably when only one of the functions is used.

You can easily switch between using *Sprites* or *SpritesB* by using one or the other to create an object instance:

```
Sprites sprites;  // Use this to optimize for execution speed
SpritesB sprites; // Use this to (likely) optimize for code size
```

Eliminate the USB stack code

Warning: Although this will free up a fair amount of code and some RAM space, without an active USB interface uploader programs will be unable to automatically force a reset to invoke the bootloader. This means the user will have to manually initiate a reset in order to upload a new sketch. This can be an inconvenience or even frustrating for a user, due to the fact that timing the sequence can sometimes be tricky. Therefore, using this technique should be considered as a last resort. If it is used, the sketch documentation should state clearly what will be involved to upload a new sketch.

The *ARDUBOY_NO_USB* macro is used to eliminate the USB code. The *exitToBootloader()* function is available to make it easier for a user to invoke the bootloader. For more details, see the documentation provided for these.

What's different from Arduboy library V1.1

A main goal of Arduboy2 is to provide ways in which more code space can be freed for use by large sketches. Another goal is to allow methods other than the *tunes* functions to be used to produce sounds. Arduboy2 remains substantially compatible with `Arduboy library V1.1`, which was the latest stable release at the time of the fork. Arduboy2 is based on the code targeted for Arduboy library V1.2, which was still in development and unreleased at the time it was forked.

Main differences between Arduboy2 and Arduboy V1.1 are:

- The *ArduboyTunes* subclass, which provided the *tunes.xxx()* functions, has been removed. It's functionality is available in a separate `ArduboyPlaytune library`. By removing these functions, more code space may become available because interrupt routines and other support code was being compiled in even if a sketch didn't make use them. Another benefit is that without the automatic installation of timer interrupt service routines, other audio generating functions and libraries, that need access to the same interrupts, can now be used. Removal of the *tunes* functions is the main API incompatibility with Arduboy V1.1. Sketches written to use *tunes* functions will need some minor modifications in order to make them work with Arduboy2 plus ArduboyPlaytune, `ArduboyTones`, or some other audio library.
- Arduboy library V1.1 uses timer 1 for the *tunes* functions. This causes problems when attempting to control the Arduboy's RGB LED using PWM, such as with *setRGBled()*, because it also requires timer 1. Since the *tunes* functionality has been removed from Arduboy2, there are no problems with using the RGB LED (except those caused by the RGB LED being incorrectly installed). Of course, using an external library that uses timer 1, such as *ArduboyPlaytune*, may reintroduce the problems. However, using a library that doesn't use timer 1, such as *ArduboyTones*, is now an option.

- The code to generate text output, using *setCursor()*, *print()*, etc., can be removed to free up code space, if a sketch doesn't use any text functions. The *Arduboy2* class includes the text functions but using the *Arduboy2Base* class instead will eliminate them. With text functions included, the font table and some support functions are always compiled in even if not used. The API for using text functions is the same as Arduboy V1.1 with some additional functions added:
  - *setTextColor()* and *setTextBackground()* allow for printing black text on a white background.
  - *getCursorX()* and *getCursorY()* allow for determining the current text cursor position.
  - The *clear()* function will now reset the text cursor to home position 0, 0.
- A new feature has been added which allows the *audio on/off* flag in system EEPROM to be configured by the user when the sketch starts. The flag is used by the Arduboy and Arduboy2 *audio* subclass, along with external sound functions and libraries, to provide a standardized sound mute capability. See the information above, under the heading *Audio mute control*, for more details.
- The *color* parameter, which is the last parameter for most of the drawing functions, has been made optional and will default to WHITE if not included in the call. This doesn't save any code but has been added as a convenience, since most drawing functions are called with WHITE specified.
- A new function *digitalWriteRGB()* has been added to control the RGB LED digitally instead of using PWM. This uses less code if just turning the RGB LEDs fully on or off is all that's required.
- The *beginNoLogo()* function is not included. This function could be used in Arduboy V1.1 in place of *begin()* to suppress the displaying of the ARDUBOY logo and thus free up the code that it required. Instead, Arduboy2 allows a sketch to call *boot()* and then add in any extra features that *begin()* provides by calling their functions directly after *boot()*, if desired.
- The *ArduboyCore* and *ArduboyAudio* base classes, previously only available to, and used to derive, the *Arduboy* class, have been made publicly available for the benefit of developers who may wish to use them as the base of an entirely new library. This change doesn't affect the existing API.

As of version 2.1.0 functionality from the `Team A.R.G.` *Arglib* library has been added:

- The sprite drawing functions, collision detection functions, and button handling functions that Team A.R.G. incorporated from the `ArduboyExtra` project. The *poll()* function was renamed *pollButtons()* for clarity. The *Sprites* class doesn't require a parameter for the constructor, whereas in *Arglib* a pointer to an Arduboy class object is required.
- The *drawCompressed()* function, which allows compressed bitmaps to be drawn. Saving bitmaps in compressed form may reduce overall sketch size.

Team A.R.G. has now migrated all of their games and demos to use the Arduboy2 library.

Migrating a sketch from Arduboy library V1.1 to Arduboy2

Since the Arduboy2 library can coexist in the Arduino IDE alongside the Arduboy library V1.1, a currently working sketch that uses Arduboy V1.1 doesn't have to be migrated to Arduboy2. However, if you want to switch a sketch to Arduboy2 for further development, in order to take advantage of any of the changes and enhancements, it's generally relatively easy.

The Arduboy2 library, for the most part, is compatible with Arduboy library V1.1 but migrating a sketch to Arduboy2 will require some small changes, and more so if it uses the *tunes* functions, such as *tunes.tone()* or *tunes.playScore()*.

Required changes

The first thing to do is change the `include` for the library header file:

```
#include <Arduboy.h>
```

becomes

```
#include <Arduboy2.h>
```

If it was "Arduboy.h" (in quotes), it's still better to change it to <Arduboy2.h> (in angle brackets).

The same thing has to be done with creating the library object. (If the object name isn't *arduboy*, keep whatever name is used.):

```
Arduboy arduboy;
```

becomes

```
Arduboy2 arduboy;
```

If the sketch doesn't use any *tunes* functions, there's a good chance this is all that has to be done to make it compile.

Sketch uses only *tunes.tone()* for sound

If the sketch has sound but only uses *tunes.tone()*, solutions are:

Solution 1: Switch to using Arduino *tone()*

An easy change is to use the Arduino built in *tone()* function. You can add a function to the sketch that wraps *tone()* so that it works like *tunes.tone()*, like so:

```
// Wrap the Arduino tone() function so that the pin doesn't have to be
// specified each time. Also, don't play if audio is set to off.
void playTone(unsigned int frequency, unsigned long duration)
{
  if (arduboy.audio.enabled() == true)
  {
    tone(PIN_SPEAKER_1, frequency, duration);
  }
}
```

You then change all *tunes.tone()* calls to *playTone()* calls using the same parameter values. For example:

```
arduboy.tunes.tone(1000, 250);
```

becomes

```
playTone(1000, 250);
```

Solution 2: Switch to using the ArduboyTones library

Changing to the *ArduboyTones* library is slightly more complicated. The advantage is that it will generate less code than using *tone()* and will also allow you to easily enhance the sketch to play tone sequences instead of just single tones. ArduboyTones can also play each tone at either normal or a higher volume.

You have to add an include for the ArduboyTones header file:

```
#include <ArduboyTones.h>
```

You then have to create an object for the *ArduboyTones* class and pass it a pointer to the Arduboy2 *audio.enabled()* function. This must go after the creation of the Arduboy2 object, like so:

```
Arduboy2 arduboy;
ArduboyTones sound(arduboy.audio.enabled);
```

You then change all Arduboy *tunes.tone()* calls to ArduboyTones *tone()* calls using the same parameter values. For example:

```
arduboy.tunes.tone(1000, 250);
```

becomes

```
sound.tone(1000, 250);
```

See the `ArduboyTones` README file for more information on installing and using it.

Solution 3: Switch to using the ArduboyPlaytune library.

See the following for how to do this:

Sketch uses *tunes.playScore()*

If the sketch uses *tunes.playScore()*, probably the easiest solution is to use the *ArduboyPlaytune* library. *ArduboyPlaytune* is essentially the code that was in the Arduboy V1.1 *tunes* subclass, which has been removed from Arduboy2. It's been cleaned up and a few enhancements have been added, but all the Arduboy V1.1 *tunes* functions are available.

You have to add an include for the ArduboyPlaytune header file:

```
#include <ArduboyPlaytune.h>
```

You then have to create an object for the *ArduboyPlaytune* class and pass it a pointer to the Arduboy2 *audio.enabled()* function. This must go after the creation of the Arduboy2 object, like so:

```
Arduboy2 arduboy;
ArduboyPlaytune tunes(arduboy.audio.enabled);
```

The sound channels must then be initialzed and assigned to the speaker pins. This code would go in the *setup()* function:

```
  // audio setup
  tunes.initChannel(PIN_SPEAKER_1);
#ifndef AB_DEVKIT
  // if not a DevKit
  tunes.initChannel(PIN_SPEAKER_2);
#else
  // if it's a DevKit
  tunes.initChannel(PIN_SPEAKER_1); // use the same pin for both channels
  tunes.toneMutesScore(true);       // mute the score when a tone is sounding
#endif
```

If you name the ArduboyPlaytune object *tunes* as shown above, then you just have to remove the Arduboy object name from any *tunes* calls. For example:

```
arduboy.tunes.playScore(mySong);
```

becomes

```
tunes.playScore(mySong);
```

See the *ArduboyPlaytune library* documentation for more information.

If you don't need to play scores containing two parts, and don't require tones to be played in parallel with a score that's playing, then as an alternative to using *ArduboyPlaytune* you may wish to consider switching to *ArduboyTones*. This may require a bit of work because any *ArduboyPlaytune* scores would have to be converted to *ArduboyTones* format. It would involve changing note numbers to frequencies. This could be simplified by using the provided *NOTE_* defines. Also, durations would have to be converted, including adding silent "rest" tones as necessary.

The benefit of using *ArduboyTones* would be reduced code size and possibly easier addition of new sequences without the need of a MIDI to Playtune format converter.

Sketch uses the *beginNoLogo()* function instead of *begin()*

The *beginNoLogo()* function has been removed. Instead, *boot()* can be used with additional functions following it to add back in desired boot functionality. See the information above, under the heading *Remove boot up features*, for more details. Assuming the object is named *arduboy*, a direct replacement for *beginNoLogo()* would be:

```
arduboy.boot();
arduboy.display();
arduboy.flashlight();
arduboy.audio.begin();
```

Chapter 2

Software License Agreements

Software License Agreements

```
based on the Adafruit-GFX library:
https://github.com/adafruit/Adafruit-GFX-Library
Copyright (c) 2012 Adafruit Industries
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice,
  this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice,
  this list of conditions and the following disclaimer in the documentation
  and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE
LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
POSSIBILITY OF SUCH DAMAGE.

------------------------------------------------------------------------------
Licensed under the MIT license:

Code from ArduboyExtra:
https://github.com/yyyc514/ArduboyExtra
Copyright (c) 2015 Josh Goebel

Code for drawing compressed bitmaps:
https://github.com/TEAMarg/drawCompressed
Copyright (c) 2016 TEAM a.r.g.

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.

------------------------------------------------------------------------------
Licensed under the GNU LGPL license:
https://www.gnu.org/licenses/old-licenses/lgpl-2.1.en.html

ArduBreakout example sketch:
Original work:
Copyright (c) 2011 Sebastian Goscik
All rights reserved.
Modified work:
Copyright (c) 2016 Scott Allen
All rights reserved.

Buttons and HelloWorld example sketches:
Copyright (c) 2015 David Martinez
All rights reserved.

This work is free software; you can redistribute it and/or
modify it under the terms of the GNU Lesser General Public
```

```
License as published by the Free Software Foundation; either
version 2.1 of the License, or (at your option) any later version.

This library is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
Lesser General Public License for more details.

--------------------------------------------------------------------------------
Placed in the public domain:

BeepDemo example sketch:
By Scott Allen

RGBled example sketch:
By Scott Allen

================================================================================
```

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 4

Class Index

4.1 Class List

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

Chapter 5

File Index

5.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 6

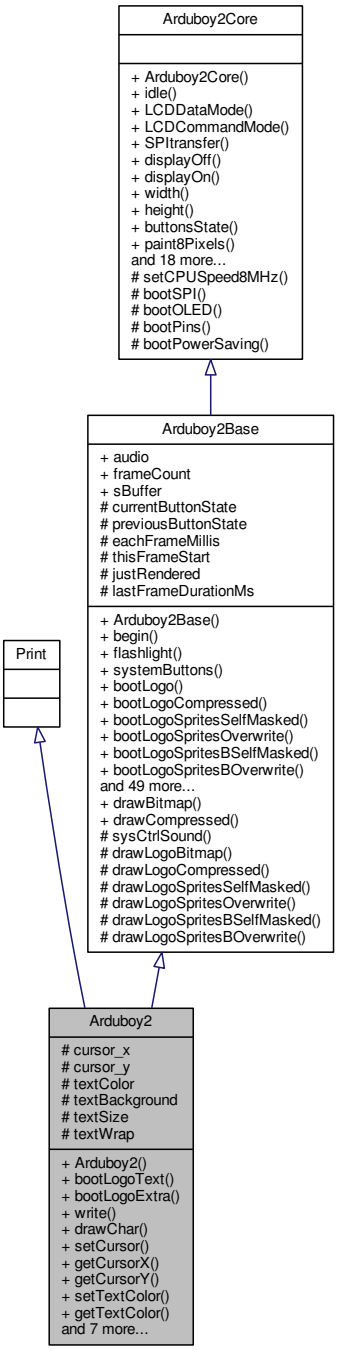
Class Documentation

6.1 Arduboy2 Class Reference

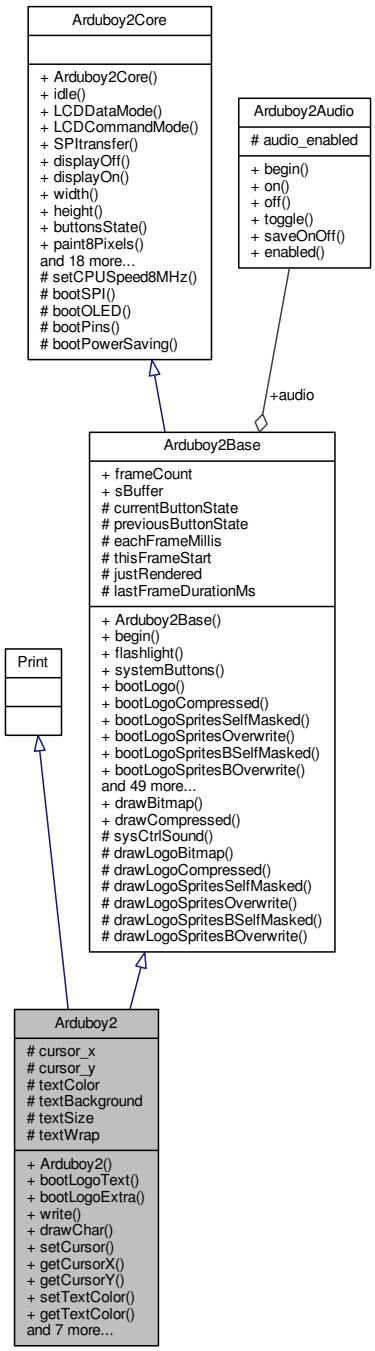
The main functions provided for writing sketches for the Arduboy, *including* text output.

```
#include <Arduboy2.h>
```

Inheritance diagram for Arduboy2:



Collaboration diagram for Arduboy2:



Public Member Functions

- void bootLogoText ()

  Display the boot logo sequence using printed text instead of a bitmap.
- virtual void bootLogoExtra ()

  Show the unit name at the bottom of the boot logo screen.
- virtual size_t write (uint8_t)

Write a single ASCII character at the current text cursor location.

- void drawChar (int16_t x, int16_t y, unsigned char c, uint8_t color, uint8_t bg, uint8_t size)

  Draw a single ASCII character at the specified location in the screen buffer.
- void setCursor (int16_t x, int16_t y)

  Set the location of the text cursor.
- int16_t getCursorX ()

  Get the X coordinate of the current text cursor position.
- int16_t getCursorY ()

  Get the Y coordinate of the current text cursor position.
- void setTextColor (uint8_t color)

  Set the text foreground color.
- uint8_t getTextColor ()

  Get the currently set text foreground color.
- void setTextBackground (uint8_t bg)

  Set the text background color.
- uint8_t getTextBackground ()

  Get the currently set text background color.
- void setTextSize (uint8_t s)

  Set the text character size.
- uint8_t getTextSize ()

  Get the currently set text size.
- void setTextWrap (bool w)

  Set or disable text wrap mode.
- bool getTextWrap ()

  Get the currently set text wrap mode.
- void clear ()

  Clear the display buffer and set the text cursor to location 0, 0.
- void begin ()

  Initialize the hardware, display the boot logo, provide boot utilities, etc.
- void flashlight ()

  Turn the RGB LED and display fully on to act as a small flashlight/torch.
- void systemButtons ()

  Handle buttons held on startup for system control.
- void bootLogo ()

  Display the boot logo sequence using `drawBitmap()`.
- void bootLogoCompressed ()

  Display the boot logo sequence using `drawCompressed()`.
- void bootLogoSpritesSelfMasked ()

  Display the boot logo sequence using `Sprites::drawSelfMasked()`.
- void bootLogoSpritesOverwrite ()

  Display the boot logo sequence using `Sprites::drawOverwrite()`.
- void bootLogoSpritesBSelfMasked ()

  Display the boot logo sequence using `SpritesB::drawSelfMasked()`.
- void bootLogoSpritesBOverwrite ()

  Display the boot logo sequence using `SpritesB::drawOverwrite()`.
- void bootLogoShell (void(*drawLogo)(int16_t))

  Display the boot logo sequence using the provided function.
- void waitNoButtons ()

  Wait until all buttons have been released.
- void display ()

  Copy the contents of the display buffer to the display.

- void display (bool clear)

  Copy the contents of the display buffer to the display. The display buffer can optionally be cleared.
- void drawPixel (int16_t x, int16_t y, uint8_t color=WHITE)

  Set a single pixel in the display buffer to the specified color.
- uint8_t getPixel (uint8_t x, uint8_t y)

  Returns the state of the given pixel in the screen buffer.
- void drawCircle (int16_t x0, int16_t y0, uint8_t r, uint8_t color=WHITE)

  Draw a circle of a given radius.
- void fillCircle (int16_t x0, int16_t y0, uint8_t r, uint8_t color=WHITE)

  Draw a filled-in circle of a given radius.
- void drawLine (int16_t x0, int16_t y0, int16_t x1, int16_t y1, uint8_t color=WHITE)

  Draw a line between two specified points.
- void drawRect (int16_t x, int16_t y, uint8_t w, uint8_t h, uint8_t color=WHITE)

  Draw a rectangle of a specified width and height.
- void drawFastVLine (int16_t x, int16_t y, uint8_t h, uint8_t color=WHITE)

  Draw a vertical line.
- void drawFastHLine (int16_t x, int16_t y, uint8_t w, uint8_t color=WHITE)

  Draw a horizontal line.
- void fillRect (int16_t x, int16_t y, uint8_t w, uint8_t h, uint8_t color=WHITE)

  Draw a filled-in rectangle of a specified width and height.
- void fillScreen (uint8_t color=WHITE)

  Fill the screen buffer with the specified color.
- void drawRoundRect (int16_t x, int16_t y, uint8_t w, uint8_t h, uint8_t r, uint8_t color=WHITE)

  Draw a rectangle with rounded corners.
- void fillRoundRect (int16_t x, int16_t y, uint8_t w, uint8_t h, uint8_t r, uint8_t color=WHITE)

  Draw a filled-in rectangle with rounded corners.
- void drawTriangle (int16_t x0, int16_t y0, int16_t x1, int16_t y1, int16_t x2, int16_t y2, uint8_t color=WHITE)

  Draw a triangle given the coordinates of each corner.
- void fillTriangle (int16_t x0, int16_t y0, int16_t x1, int16_t y1, int16_t x2, int16_t y2, uint8_t color=WHITE)

  Draw a filled-in triangle given the coordinates of each corner.
- void drawSlowXYBitmap (int16_t x, int16_t y, const uint8_t *bitmap, uint8_t w, uint8_t h, uint8_t color=WHITE)

  Draw a bitmap from a horizontally oriented array in program memory.
- uint8_t * getBuffer ()

  Get a pointer to the display buffer in RAM.
- void initRandomSeed ()

  Seed the random number generator with a random value.
- void setFrameRate (uint8_t rate)

  Set the frame rate used by the frame control functions.
- void setFrameDuration (uint8_t duration)

  Set the frame rate, used by the frame control functions, by giving the duration of each frame.
- bool nextFrame ()

  Indicate that it's time to render the next frame.
- bool nextFrameDEV ()

  Indicate that it's time to render the next frame, and visually indicate if the code is running slower than the desired frame rate. ***FOR USE DURING DEVELOPMENT***
- bool everyXFrames (uint8_t frames)

  Indicate if the specified number of frames has elapsed.
- int cpuLoad ()

  Return the load on the CPU as a percentage.
- bool pressed (uint8_t buttons)

  Test if the specified buttons are pressed.

- bool notPressed (uint8_t buttons)

  Test if the specified buttons are not pressed.
- void pollButtons ()

  Poll the buttons and track their state over time.
- bool justPressed (uint8_t button)

  Check if a button has just been pressed.
- bool justReleased (uint8_t button)

  Check if a button has just been released.
- bool collide (Point point, Rect rect)

  Test if a point falls within a rectangle.
- bool collide (Rect rect1, Rect rect2)

  Test if a rectangle is intersecting with another rectangle.
- uint16_t readUnitID ()

  Read the unit ID from system EEPROM.
- void writeUnitID (uint16_t id)

  Write a unit ID to system EEPROM.
- uint8_t readUnitName (char *name)

  Read the unit name from system EEPROM.
- void writeUnitName (char *name)

  Write a unit name to system EEPROM.
- bool readShowBootLogoFlag ()

  Read the "Show Boot Logo" flag in system EEPROM.
- void writeShowBootLogoFlag (bool val)

  Write the "Show Boot Logo" flag in system EEPROM.
- bool readShowUnitNameFlag ()

  Read the "Show Unit Name" flag in system EEPROM.
- void writeShowUnitNameFlag (bool val)

  Write the "Show Unit Name" flag in system EEPROM.
- bool readShowBootLogoLEDsFlag ()

  Read the "Show LEDs with boot logo" flag in system EEPROM.
- void writeShowBootLogoLEDsFlag (bool val)

  Write the "Show LEDs with boot logo" flag in system EEPROM.

Static Public Member Functions

- static void drawBitmap (int16_t x, int16_t y, const uint8_t *bitmap, uint8_t w, uint8_t h, uint8_t color=WHITE)

  Draw a bitmap from an array in program memory.
- static void drawCompressed (int16_t sx, int16_t sy, const uint8_t *bitmap, uint8_t color=WHITE)

  Draw a bitmap from an array of compressed data.
- static void idle ()

  Idle the CPU to save power.
- static void LCDDataMode ()

  Put the display into data mode.
- static void LCDCommandMode ()

  Put the display into command mode.
- static void SPItransfer (uint8_t data)

  Transfer a byte to the display.
- static void displayOff ()

  Turn the display off.
- static void displayOn ()

Turn the display on.

- static uint8_t width ()

  Get the width of the display in pixels.
- static uint8_t height ()

  Get the height of the display in pixels.
- static uint8_t buttonsState ()

  Get the current state of all buttons as a bitmask.
- static void paint8Pixels (uint8_t pixels)

  Paint 8 pixels vertically to the display.
- static void paintScreen (const uint8_t *image)

  Paints an entire image directly to the display from program memory.
- static void paintScreen (uint8_t image[ ], bool clear=false)

  Paints an entire image directly to the display from an array in RAM.
- static void blank ()

  Blank the display screen by setting all pixels off.
- static void invert (bool inverse)

  Invert the entire display or set it back to normal.
- static void allPixelsOn (bool on)

  Turn all display pixels on or display the buffer contents.
- static void flipVertical (bool flipped)

  Flip the display vertically or set it back to normal.
- static void flipHorizontal (bool flipped)

  Flip the display horizontally or set it back to normal.
- static void sendLCDCommand (uint8_t command)

  Send a single command byte to the display.
- static void setRGBled (uint8_t red, uint8_t green, uint8_t blue)

  Set the light output of the RGB LED.
- static void setRGBled (uint8_t color, uint8_t val)

  Set the brightness of one of the RGB LEDs without affecting the others.
- static void freeRGBled ()

  Relinquish analog control of the RGB LED.
- static void digitalWriteRGB (uint8_t red, uint8_t green, uint8_t blue)

  Set the RGB LEDs digitally, to either fully on or fully off.
- static void digitalWriteRGB (uint8_t color, uint8_t val)

  Set one of the RGB LEDs digitally, to either fully on or fully off.
- static void boot ()

  Initialize the Arduboy's hardware.
- static void safeMode ()

  Allow upload when the bootloader "magic number" could be corrupted.
- static void delayShort (uint16_t ms) __attribute__((noinline))

  Delay for the number of milliseconds, specified as a 16 bit value.
- static void exitToBootloader ()

  Exit the sketch and start the bootloader.

Public Attributes

- Arduboy2Audio audio

  An object created to provide audio control functions within this class.
- uint16_t frameCount

  A counter which is incremented once per frame.

Static Public Attributes

- static uint8_t sBuffer [(HEIGHT ∗WIDTH)/8]

  The display buffer array in RAM.

6.1.1 Detailed Description

The main functions provided for writing sketches for the Arduboy, *including* text output.

This class is derived from Arduboy2Base. It provides text output functions in addition to all the functions inherited from Arduboy2Base.

Note

A friend class named *Arduboy2Ex* is declared by this class. The intention is to allow a sketch to create an *Arduboy2Ex* class which would have access to the private and protected members of the Arduboy2 class. It is hoped that this may eliminate the need to create an entire local copy of the library, in order to extend the functionality, in most circumstances.

See also

Arduboy2Base

Definition at line 1301 of file Arduboy2.h.

6.1.2 Member Function Documentation

6.1.2.1 void Arduboy2Core::allPixelsOn (bool *on*) [static], [inherited]

Turn all display pixels on or display the buffer contents.

Parameters

on	`true` turns all pixels on. `false` displays the contents of the hardware display buffer.

Calling this function with a value of `true` will override the contents of the hardware display buffer and turn all pixels on. The contents of the hardware buffer will remain unchanged.

Calling this function with a value of `false` will set the normal state of displaying the contents of the hardware display buffer.

Note

All pixels will be lit even if the display is in inverted mode.

See also

invert()

Definition at line 414 of file Arduboy2Core.cpp.

6.1.2.2 void Arduboy2Base::begin ( ) [inherited]

Initialize the hardware, display the boot logo, provide boot utilities, etc.

This function should be called once near the start of the sketch, usually in `setup()`, before using any other functions in this class. It initializes the display, displays the boot logo, provides "flashlight" and system control features and initializes audio control.

Note

To free up some code space for use by the sketch, `boot()` can be used instead of `begin()` to allow the elimination of some of the things that aren't really required, such as displaying the boot logo.

See also

boot()

Definition at line 30 of file Arduboy2.cpp.

6.1.2.3 void Arduboy2Core::blank ( ) [static], [inherited]

Blank the display screen by setting all pixels off.

All pixels on the screen will be written with a value of 0 to turn them off.

Definition at line 392 of file Arduboy2Core.cpp.

6.1.2.4 void Arduboy2Core::boot ( ) [static], [inherited]

Initialize the Arduboy's hardware.

This function initializes the display, buttons, etc.

This function is called by begin() so isn't normally called within a sketch. However, in order to free up some code space, by eliminating some of the start up features, it can be called in place of begin(). The functions that begin() would call after boot() can then be called to add back in some of the start up features, if desired. See the README file or documentation on the main page for more details.

See also

Arduboy2Base::begin()

Definition at line 76 of file Arduboy2Core.cpp.

6.1.2.5 void Arduboy2Base::bootLogo ( ) [inherited]

Display the boot logo sequence using `drawBitmap()`.

This function is called by `begin()` and can be called by a sketch after `boot()`.

The Arduboy logo scrolls down from the top of the screen to the center while the RGB LEDs light in sequence.

The `bootLogoShell()` helper function is used to perform the actual sequence. The documentation for `bootLogoShell()` provides details on how it operates.

See also

begin() boot() bootLogoShell() Arduboy2::bootLogoText()

Definition at line 102 of file Arduboy2.cpp.

6.1.2.6 void Arduboy2Base::bootLogoCompressed ( ) [inherited]

Display the boot logo sequence using `drawCompressed()`.

This function can be called by a sketch after `boot()` as an alternative to `bootLogo()`. This may reduce code size if the sketch itself uses `drawCompressed()`.

See also

bootLogo() begin() boot()

Definition at line 112 of file Arduboy2.cpp.

6.1.2.7 void Arduboy2::bootLogoExtra ( ) [virtual]

Show the unit name at the bottom of the boot logo screen.

This function is called by `bootLogoShell()` and `bootLogoText()`.

If a unit name has been saved in system EEPROM, it will be displayed at the bottom of the screen. This function pauses for a short time to allow the name to be seen.

If the SYS_FLAG_UNAME flag in system EEPROM is cleared, this function will return without showing the unit name or pausing.

Note

This function would not normally be called directly from within a sketch itself.

See also

readUnitName() writeUnitName() bootLogo() bootLogoShell() bootLogoText() writeShowUnitNameFlag() begin()

Reimplemented from Arduboy2Base.

Definition at line 1212 of file Arduboy2.cpp.

6.1.2.8 void Arduboy2Base::bootLogoShell (void(*)(int16_t) *drawLogo*) [inherited]

Display the boot logo sequence using the provided function.

Parameters

drawLogo	A reference to a function which will draw the boot logo at the given Y position.

This common function executes the sequence to display the boot logo. It is called by `bootLogo()` and other similar functions which provide it with a reference to a function which will do the actual drawing of the logo.

This function calls `bootLogoExtra()` after the logo stops scrolling down, which derived classes can implement to add additional information to the logo screen. The `Arduboy2` class uses this to display the unit name.

If the RIGHT button is pressed while the logo is scrolling down, the boot logo sequence will be aborted. This can be useful for developers who wish to quickly start testing, or anyone else who is impatient and wants to go straight to the actual sketch.

If the SYS_FLAG_SHOW_LOGO_LEDS flag in system EEPROM is cleared, the RGB LEDs will not be flashed during the logo display sequence.

If the SYS_FLAG_SHOW_LOGO flag in system EEPROM is cleared, this function will return without executing the logo display sequence.

The prototype for the function provided to draw the logo is:

```
void drawLogo(int16_t y);
```

The y parameter is the Y offset for the top of the logo. It is expected that the logo will be 16 pixels high and centered horizontally. This will result in the logo stopping in the middle of the screen at the end of the sequence. If the logo height is not 16 pixels, the Y value can be adjusted to compensate.

See also

bootLogo() boot() Arduboy2::bootLogoExtra()

Definition at line 164 of file Arduboy2.cpp.

6.1.2.9 void Arduboy2Base::bootLogoSpritesBOverwrite ( ) [inherited]

Display the boot logo sequence using `SpritesB::drawOverwrite()`.

This function can be called by a sketch after `boot()` as an alternative to `bootLogo()`. This may reduce code size if the sketch itself uses `SpritesB` class functions.

See also

bootLogo() begin() boot() SpritesB

Definition at line 152 of file Arduboy2.cpp.

6.1.2.10 void Arduboy2Base::bootLogoSpritesBSelfMasked ( ) [inherited]

Display the boot logo sequence using `SpritesB::drawSelfMasked()`.

This function can be called by a sketch after `boot()` as an alternative to `bootLogo()`. This may reduce code size if the sketch itself uses `SpritesB` class functions.

See also

bootLogo() begin() boot() SpritesB

Definition at line 142 of file Arduboy2.cpp.

6.1.2.11 void Arduboy2Base::bootLogoSpritesOverwrite ( ) [inherited]

Display the boot logo sequence using `Sprites::drawOverwrite()`.

This function can be called by a sketch after `boot()` as an alternative to `bootLogo()`. This may reduce code size if the sketch itself uses `Sprites` class functions.

See also

bootLogo() begin() boot() Sprites

Definition at line 132 of file Arduboy2.cpp.

6.1.2.12 void Arduboy2Base::bootLogoSpritesSelfMasked ( ) [inherited]

Display the boot logo sequence using `Sprites::drawSelfMasked()`.

This function can be called by a sketch after `boot()` as an alternative to `bootLogo()`. This may reduce code size if the sketch itself uses `Sprites` class functions.

See also

bootLogo() begin() boot() Sprites

Definition at line 122 of file Arduboy2.cpp.

6.1.2.13 void Arduboy2::bootLogoText ( )

Display the boot logo sequence using printed text instead of a bitmap.

This function can be called by a sketch after `boot()` as an alternative to `bootLogo()`.

The Arduboy logo scrolls down from the top of the screen to the center while the RGB LEDs light in sequence.

This function is the same as `bootLogo()` except the logo is printed as text instead of being rendered as a bitmap. It can be used to save some code space in a case where the sketch is using the Print class functions to display text. However, the logo will not look as good when printed as text as it does with the bitmap used by `bootLogo()`.

If the RIGHT button is pressed while the logo is scrolling down, the boot logo sequence will be aborted. This can be useful for developers who wish to quickly start testing, or anyone else who is impatient and wants to go straight to the actual sketch.

If the SYS_FLAG_SHOW_LOGO_LEDS flag in system EEPROM is cleared, the RGB LEDs will not be flashed during the logo display sequence.

If the SYS_FLAG_SHOW_LOGO flag in system EEPROM is cleared, this function will return without executing the logo display sequence.

See also

bootLogo() boot() Arduboy2::bootLogoExtra()

Definition at line 1167 of file Arduboy2.cpp.

6.1.2.14 uint8_t Arduboy2Core::buttonsState ( ) [static], [inherited]

Get the current state of all buttons as a bitmask.

Returns

A bitmask of the state of all the buttons.

The returned mask contains a bit for each button. For any pressed button, its bit will be 1. For released buttons their associated bits will be 0.

The following defined mask values should be used for the buttons:

LEFT_BUTTON, RIGHT_BUTTON, UP_BUTTON, DOWN_BUTTON, A_BUTTON, B_BUTTON

Definition at line 528 of file Arduboy2Core.cpp.

6.1.2.15 bool Arduboy2Base::collide (Point *point*, Rect *rect*) [inherited]

Test if a point falls within a rectangle.

Parameters

point	A structure describing the location of the point.
rect	A structure describing the location and size of the rectangle.

Returns

`true` if the specified point is within the specified rectangle.

This function is intended to detemine if an object, whose boundaries are are defined by the given rectangle, is in contact with the given point.

See also

Point Rect

Definition at line 1042 of file Arduboy2.cpp.

6.1.2.16 bool Arduboy2Base::collide (Rect *rect1*, Rect *rect2*) [inherited]

Test if a rectangle is intersecting with another rectangle.

Parameters

rect1,rect2	Structures describing the size and locations of the rectangles.

Returns

`true` if the first rectangle is intersecting the second.

This function is intended to detemine if an object, whose boundaries are are defined by the given rectangle, is in contact with another rectangular object.

See also

Rect

Definition at line 1048 of file Arduboy2.cpp.

6.1.2.17 int Arduboy2Base::cpuLoad ( ) [inherited]

Return the load on the CPU as a percentage.

Returns

The load on the CPU as a percentage of the total frame time.

The returned value gives the time spent processing a frame as a percentage the total time allotted for a frame, as determined by the frame rate.

This function normally wouldn't be used in the final program. It is intended for use during program development as an aid in helping with frame timing.

Note

The percentage returned can be higher than 100 if more time is spent processing a frame than the time allotted per frame. This would indicate that the frame rate should be made slower or the frame processing code should be optimized to run faster.

See also

setFrameRate() nextFrame()

Definition at line 273 of file Arduboy2.cpp.

6.1.2.18 void Arduboy2Core::delayShort (uint16_t *ms*) [static], [inherited]

Delay for the number of milliseconds, specified as a 16 bit value.

Parameters

ms	The delay in milliseconds.

This function works the same as the Arduino `delay()` function except the provided value is 16 bits long, so the maximum delay allowed is 65535 milliseconds (about 65.5 seconds). Using this function instead of Arduino

`delay()` will save a few bytes of code.

Definition at line 557 of file Arduboy2Core.cpp.

6.1.2.19 void Arduboy2Core::digitalWriteRGB (uint8_t *red,* uint8_t *green,* uint8_t *blue*) [static],[inherited]

Set the RGB LEDs digitally, to either fully on or fully off.

Parameters

red,green,blue	Use value RGB_ON or RGB_OFF to set each LED.

The RGB LED is actually individual red, green and blue LEDs placed very close together in a single package. This 3 parameter version of the function will set each LED either on or off, to set the RGB LED to 7 different colors at their highest brightness or turn it off.

The colors are as follows:

```
RED_LED   GREEN_LED   BLUE_LED   COLOR
-------   ---------   --------   -----
RGB_OFF    RGB_OFF    RGB_OFF    OFF
RGB_OFF    RGB_OFF    RGB_ON     Blue
RGB_OFF    RGB_ON     RGB_OFF    Green
RGB_OFF    RGB_ON     RGB_ON     Cyan
RGB_ON     RGB_OFF    RGB_OFF    Red
RGB_ON     RGB_OFF    RGB_ON     Magenta
RGB_ON     RGB_ON     RGB_OFF    Yellow
RGB_ON     RGB_ON     RGB_ON     White
```

Note

Using the RGB LED in analog mode will prevent digital control of the LED. To restore the ability to control the LED digitally, use the `freeRGBled()` function.

Note

Many of the Kickstarter Arduboys were accidentally shipped with the RGB LED installed incorrectly. For these units, the green LED cannot be lit. As long as the green led is set to off, turning on the red LED will actually light the blue LED and turning on the blue LED will actually light the red LED. If the green LED is turned on, none of the LEDs will light.

See also

digitalWriteRGB(uint8_t, uint8_t) setRGBled() freeRGBled()

Definition at line 488 of file Arduboy2Core.cpp.

6.1.2.20 void Arduboy2Core::digitalWriteRGB (uint8_t *color,* uint8_t *val*) [static],[inherited]

Set one of the RGB LEDs digitally, to either fully on or fully off.

Parameters

color	The name of the LED to set. The value given should be one of RED_LED, GREEN_LED or BLUE_LED.
val	Indicates whether to turn the specified LED on or off. The value given should be RGB_ON or RGB_OFF.

This 2 parameter version of the function will set a single LED within the RGB LED either fully on or fully off. See the description of the 3 parameter version of this function for more details on the RGB LED.

See also

digitalWriteRGB(uint8_t, uint8_t, uint8_t) setRGBled() freeRGBled()

Definition at line 502 of file Arduboy2Core.cpp.

6.1.2.21 void Arduboy2Base::display ( ) [inherited]

Copy the contents of the display buffer to the display.

The contents of the display buffer in RAM are copied to the display and will appear on the screen.

See also

display(bool)

Definition at line 1001 of file Arduboy2.cpp.

6.1.2.22 void Arduboy2Base::display (bool *clear*) [inherited]

Copy the contents of the display buffer to the display. The display buffer can optionally be cleared.

Parameters

clear	If `true` the display buffer will be cleared to zero. The defined value `CLEAR_BUFFER` should be used instead of `true` to make it more meaningful.

Operation is the same as calling `display()` without parameters except additionally the display buffer will be cleared if the parameter evaluates to `true`. (The defined value `CLEAR_BUFFER` can be used for this)

Using `display(CLEAR_BUFFER)` is faster and produces less code than calling `display()` followed by `clear()`.

See also

display() clear()

Definition at line 1006 of file Arduboy2.cpp.

6.1.2.23 void Arduboy2Core::displayOff ( ) [static], [inherited]

Turn the display off.

The display will clear and be put into a low power mode. This can be used to extend battery life when a game is paused or when a sketch doesn't require anything to be displayed for a relatively long period of time.

See also

displayOn()

Definition at line 283 of file Arduboy2Core.cpp.

6.1.2.24 void Arduboy2Core::displayOn ( ) [static], [inherited]

Turn the display on.

Used to power up and reinitialize the display after calling `displayOff()`.

Note

The previous call to `displayOff()` will have caused the display's buffer contents to be lost. The display will have to be re-painted, which is usually done by calling `display()`.

See also

displayOff()

Definition at line 294 of file Arduboy2Core.cpp.

6.1.2.25 void Arduboy2Base::drawBitmap (int16_t *x*, int16_t *y*, const uint8_t * *bitmap*, uint8_t *w*, uint8_t *h*, uint8_t *color* = WHITE) [static], [inherited]

Draw a bitmap from an array in program memory.

Parameters

x	The X coordinate of the top left pixel affected by the bitmap.
y	The Y coordinate of the top left pixel affected by the bitmap.
bitmap	A pointer to the bitmap array in program memory.
w	The width of the bitmap in pixels.
h	The height of the bitmap in pixels.
color	The color of pixels for bits set to 1 in the bitmap. If the value is INVERT, bits set to 1 will invert the corresponding pixel. (optional; defaults to WHITE).

Bits set to 1 in the provided bitmap array will have their corresponding pixel set to the specified color. For bits set to 0 in the array, the corresponding pixel will be left unchanged.

Each byte in the array specifies a vertical column of 8 pixels, with the least significant bit at the top.

The array must be located in program memory by using the PROGMEM modifier.

Definition at line 809 of file Arduboy2.cpp.

6.1.2.26 void Arduboy2::drawChar (int16_t *x*, int16_t *y*, unsigned char *c*, uint8_t *color*, uint8_t *bg*, uint8_t *size*)

Draw a single ASCII character at the specified location in the screen buffer.

Parameters

x	The X coordinate, in pixels, for where to draw the character.
y	The Y coordinate, in pixels, for where to draw the character.
c	The ASCII value of the character to be drawn.
color	the forground color of the character.
bg	the background color of the character.
size	The size of the character to draw.

The specified ASCII character is drawn starting at the provided coordinate. The point specified by the X and Y coordinates will be the top left corner of the character.

Note

This is a low level function used by the `write()` function to draw a character. Although it's available as a public function, it wouldn't normally be used. In most cases the Arduino Print class should be used for writing text.

See also

Print write() setTextColor() setTextBackground() setTextSize()

Definition at line 1267 of file Arduboy2.cpp.

6.1.2.27 void Arduboy2Base::drawCircle (int16_t *x0*, int16_t *y0*, uint8_t *r*, uint8_t *color* = WHITE) [inherited]

Draw a circle of a given radius.

Parameters

x0	The X coordinate of the circle's center.
y0	The Y coordinate of the circle's center.
r	The radius of the circle in pixels.
color	The circle's color (optional; defaults to WHITE).

Definition at line 367 of file Arduboy2.cpp.

6.1.2.28 void Arduboy2Base::drawCompressed (int16_t *sx*, int16_t *sy*, const uint8_t * *bitmap*, uint8_t *color* = WHITE) [static], [inherited]

Draw a bitmap from an array of compressed data.

Parameters

sx	The X coordinate of the top left pixel affected by the bitmap.
sy	The Y coordinate of the top left pixel affected by the bitmap.
bitmap	A pointer to the compressed bitmap array in program memory.
color	The color of pixels for bits set to 1 in the bitmap. (optional; defaults to WHITE).

Draw a bitmap starting at the given coordinates from an array that has been compressed using an algorthm implemented by Team A.R.G. For more information see: https://github.com/TEAMarg/drawCompressed https://github.com/TEAMarg/Cabi

Bits set to 1 in the provided bitmap array will have their corresponding pixel set to the specified color. For bits set to 0 in the array, the corresponding pixel will be left unchanged.

The array must be located in program memory by using the PROGMEM modifier.

Definition at line 906 of file Arduboy2.cpp.

6.1.2.29 void Arduboy2Base::drawFastHLine (int16_t *x*, int16_t *y*, uint8_t *w*, uint8_t *color* = WHITE) [inherited]

Draw a horizontal line.

Parameters

x	The X coordinate of the left start point.
y	The Y coordinate of the left start point.
w	The width of the line.
color	The color of the line (optional; defaults to WHITE).

Definition at line 563 of file Arduboy2.cpp.

6.1.2.30 void Arduboy2Base::drawFastVLine (int16_t *x*, int16_t *y*, uint8_t *h*, uint8_t *color* = WHITE) [inherited]

Draw a vertical line.

Parameters

x	The X coordinate of the upper start point.
y	The Y coordinate of the upper start point.
h	The height of the line.
color	The color of the line (optional; defaults to WHITE).

Definition at line 553 of file Arduboy2.cpp.

6.1.2.31 void Arduboy2Base::drawLine (int16_t *x0*, int16_t *y0*, int16_t *x1*, int16_t *y1*, uint8_t *color* = WHITE) [inherited]

Draw a line between two specified points.

Parameters

x0,x1	The X coordinates of the line ends.
y0,y1	The Y coordinates of the line ends.
color	The line's color (optional; defaults to WHITE).

Draw a line from the start point to the end point using Bresenham's algorithm. The start and end points can be at any location with respect to the other.

Definition at line 493 of file Arduboy2.cpp.

6.1.2.32 void Arduboy2Base::drawPixel (int16_t *x*, int16_t *y*, uint8_t *color* = WHITE) [inherited]

Set a single pixel in the display buffer to the specified color.

Parameters

x	The X coordinate of the pixel.
y	The Y coordinate of the pixel.
color	The color of the pixel (optional; defaults to WHITE).

The single pixel specified location in the display buffer is set to the specified color. The values WHITE or BLACK can be used for the color. If the `color` parameter isn't included, the pixel will be set to WHITE.

Definition at line 306 of file Arduboy2.cpp.

6.1.2.33 void Arduboy2Base::drawRect (int16_t *x*, int16_t *y*, uint8_t *w*, uint8_t *h*, uint8_t *color* = WHITE) [inherited]

Draw a rectangle of a specified width and height.

Parameters

x	The X coordinate of the upper left corner.
y	The Y coordinate of the upper left corner.
w	The width of the rectangle.
h	The height of the rectangle.
color	The color of the pixel (optional; defaults to WHITE).

Definition at line 544 of file Arduboy2.cpp.

6.1.2.34 void Arduboy2Base::drawRoundRect (int16_t *x*, int16_t *y*, uint8_t *w*, uint8_t *h*, uint8_t *r*, uint8_t *color* = WHITE) [inherited]

Draw a rectangle with rounded corners.

Parameters

x	The X coordinate of the left edge.
y	The Y coordinate of the top edge.
w	The width of the rectangle.
h	The height of the rectangle.
r	The radius of the semicircles forming the corners.
color	The color of the rectangle (optional; defaults to WHITE).

Definition at line 672 of file Arduboy2.cpp.

6.1.2.35 void Arduboy2Base::drawSlowXYBitmap (int16_t *x*, int16_t *y*, const uint8_t * *bitmap*, uint8_t *w*, uint8_t *h*, uint8_t *color* = WHITE) [inherited]

Draw a bitmap from a horizontally oriented array in program memory.

Parameters

x	The X coordinate of the top left pixel affected by the bitmap.
y	The Y coordinate of the top left pixel affected by the bitmap.
bitmap	A pointer to the bitmap array in program memory.
w	The width of the bitmap in pixels.
h	The height of the bitmap in pixels.
color	The color of pixels for bits set to 1 in the bitmap. (optional; defaults to WHITE).

Bits set to 1 in the provided bitmap array will have their corresponding pixel set to the specified color. For bits set to 0 in the array, the corresponding pixel will be left unchanged.

Each byte in the array specifies a horizontal row of 8 pixels, with the most significant bit at the left end of the row.

The array must be located in program memory by using the PROGMEM modifier.

Note

This function requires a lot of additional CPU power and will draw images slower than `drawBitmap()`, which uses bitmaps that are stored in a format that allows them to be directly written to the screen. It is recommended you use `drawBitmap()` when possible.

Definition at line 855 of file Arduboy2.cpp.

6.1.2.36 void Arduboy2Base::drawTriangle (int16_t *x0*, int16_t *y0*, int16_t *x1*, int16_t *y1*, int16_t *x2*, int16_t *y2*, uint8_t *color* = WHITE) [inherited]

Draw a triangle given the coordinates of each corner.

Parameters

x0,x1,x2	The X coordinates of the corners.
y0,y1,y2	The Y coordinates of the corners.
color	The triangle's color (optional; defaults to WHITE).

A triangle is drawn by specifying each of the three corner locations. The corners can be at any position with respect to the others.

Definition at line 698 of file Arduboy2.cpp.

6.1.2.37 bool Arduboy2Base::everyXFrames (uint8_t *frames*) [inherited]

Indicate if the specified number of frames has elapsed.

Parameters

frames	The desired number of elapsed frames.

Returns

`true` if the specified number of frames has elapsed.

This function should be called with the same value each time for a given event. It will return `true` if the given number of frames has elapsed since the previous frame in which it returned `true`.

For example, if you wanted to fire a shot every 5 frames while the A button is being held down:

```
if (arduboy.everyXFrames(5)) {
  if arduboy.pressed(A_BUTTON) {
    fireShot();
  }
}
```

See also

setFrameRate() nextFrame()

Definition at line 227 of file Arduboy2.cpp.

6.1.2.38 void Arduboy2Core::exitToBootloader ( ) [static], [inherited]

Exit the sketch and start the bootloader.

The sketch will exit and the bootloader will be started in command mode. The effect will be similar to pressing the reset button.

This function is intended to be used to allow uploading a new sketch, when the USB code has been removed to gain more code space. Ideally, the sketch would present a "New Sketch Upload" menu or prompt telling the user to "Press and hold the DOWN button when the procedure to upload a new sketch has been initiated". The sketch would then wait for the DOWN button to be pressed and then call this function.

See also

ARDUBOY_NO_USB

Definition at line 562 of file Arduboy2Core.cpp.

6.1.2.39 void Arduboy2Base::fillCircle (int16_t *x0*, int16_t *y0*, uint8_t *r*, uint8_t *color* = **WHITE**) [inherited]

Draw a filled-in circle of a given radius.

Parameters

x0	The X coordinate of the circle's center.
y0	The Y coordinate of the circle's center.
r	The radius of the circle in pixels.
color	The circle's color (optional; defaults to WHITE).

Definition at line 449 of file Arduboy2.cpp.

6.1.2.40 void Arduboy2Base::fillRect (int16_t *x*, int16_t *y*, uint8_t *w*, uint8_t *h*, uint8_t *color* = **WHITE**) [inherited]

Draw a filled-in rectangle of a specified width and height.

Parameters

x	The X coordinate of the upper left corner.
y	The Y coordinate of the upper left corner.
w	The width of the rectangle.
h	The height of the rectangle.
color	The color of the pixel (optional; defaults to WHITE).

Definition at line 614 of file Arduboy2.cpp.

6.1.2.41 void Arduboy2Base::fillRoundRect (int16_t *x*, int16_t *y*, uint8_t *w*, uint8_t *h*, uint8_t *r*, uint8_t *color* = **WHITE**) [inherited]

Draw a filled-in rectangle with rounded corners.

Parameters

x	The X coordinate of the left edge.
y	The Y coordinate of the top edge.
w	The width of the rectangle.
h	The height of the rectangle.
r	The radius of the semicircles forming the corners.
color	The color of the rectangle (optional; defaults to WHITE).

Definition at line 687 of file Arduboy2.cpp.

6.1.2.42 void Arduboy2Base::fillScreen (uint8_t *color* = **WHITE**) [inherited]

Fill the screen buffer with the specified color.

Parameters

color	The fill color (optional; defaults to WHITE).

Definition at line 623 of file Arduboy2.cpp.

6.1.2.43 void Arduboy2Base::fillTriangle (int16_t *x0*, int16_t *y0*, int16_t *x1*, int16_t *y1*, int16_t *x2*, int16_t *y2*, uint8_t *color* = WHITE) [inherited]

Draw a filled-in triangle given the coordinates of each corner.

Parameters

x0,x1,x2	The X coordinates of the corners.
y0,y1,y2	The Y coordinates of the corners.
color	The triangle's color (optional; defaults to WHITE).

A triangle is drawn by specifying each of the three corner locations. The corners can be at any position with respect to the others.

Definition at line 706 of file Arduboy2.cpp.

6.1.2.44 void Arduboy2Base::flashlight ( ) [inherited]

Turn the RGB LED and display fully on to act as a small flashlight/torch.

Checks if the UP button is pressed and if so turns the RGB LED and all display pixels fully on. If the UP button is detected, this function does not exit. The Arduboy must be restarted after flashlight mode is used.

This function is called by `begin()` and can be called by a sketch after `boot()`.

Note

This function also contains code to address a problem with uploading a new sketch, for sketches that interfere with the bootloader "magic number". This problem occurs with certain sketches that use large amounts of RAM. Being in flashlight mode when uploading a new sketch can fix this problem.
Therefore, for sketches that potentially could cause this problem, and use `boot()` instead of `begin()`, it is recommended that a call to `flashlight()` be included after calling `boot()`. If program space is limited, `safeMode()` can be used instead of `flashlight()`.

See also

begin() boot() safeMode()

Definition at line 55 of file Arduboy2.cpp.

6.1.2.45 void Arduboy2Core::flipHorizontal (bool *flipped*) [static], [inherited]

Flip the display horizontally or set it back to normal.

Parameters

flipped	`true` will set horizontal flip mode. `false` will set normal horizontal orientation.

Calling this function with a value of `true` will cause the X coordinate to start at the left edge of the display instead of the right, effectively flipping the display horizontally.

Once in horizontal flip mode, it will remain this way until normal horizontal mode is set by calling this function with a value of `false`.

See also

flipVertical()

Definition at line 426 of file Arduboy2Core.cpp.

6.1.2.46 void Arduboy2Core::flipVertical (bool *flipped*) [static], [inherited]

Flip the display vertically or set it back to normal.

Parameters

flipped	`true` will set vertical flip mode. `false` will set normal vertical orientation.

Calling this function with a value of `true` will cause the Y coordinate to start at the bottom edge of the display instead of the top, effectively flipping the display vertically.

Once in vertical flip mode, it will remain this way until normal vertical mode is set by calling this function with a value of `false`.

See also

flipHorizontal()

Definition at line 420 of file Arduboy2Core.cpp.

6.1.2.47 void Arduboy2Core::freeRGBled ( ) [static], [inherited]

Relinquish analog control of the RGB LED.

Using the RGB LED in analog mode prevents further use of the LED in digital mode. This function will restore the pins used for the LED, so it can be used in digital mode.

See also

digitalWriteRGB() setRGBled()

Definition at line 479 of file Arduboy2Core.cpp.

6.1.2.48 uint8_t * Arduboy2Base::getBuffer ( ) [inherited]

Get a pointer to the display buffer in RAM.

Returns

A pointer to the display buffer array in RAM.

The location of the display buffer in RAM, which is displayed using `display()`, can be gotten using this function. The buffer can then be read and directly manipulated.

Note

The display buffer array, `sBuffer`, is public. A sketch can access it directly. Doing so may be more efficient than accessing it via the pointer returned by `getBuffer()`.

See also

sBuffer

Definition at line 1011 of file Arduboy2.cpp.

6.1.2.49 int16_t Arduboy2::getCursorX ( )

Get the X coordinate of the current text cursor position.

Returns

The X coordinate of the current text cursor position.

The X coordinate returned is a pixel location with 0 indicating the leftmost column.

See also

getCursorY() setCursor()

Definition at line 1311 of file Arduboy2.cpp.

6.1.2.50 int16_t Arduboy2::getCursorY ( )

Get the Y coordinate of the current text cursor position.

Returns

The Y coordinate of the current text cursor position.

The Y coordinate returned is a pixel location with 0 indicating the topmost row.

See also

getCursorX() setCursor()

Definition at line 1316 of file Arduboy2.cpp.

6.1.2.51 uint8_t Arduboy2Base::getPixel (uint8_t *x*, uint8_t *y*) [inherited]

Returns the state of the given pixel in the screen buffer.

Parameters

x	The X coordinate of the pixel.
y	The Y coordinate of the pixel.

Returns

WHITE if the pixel is on or BLACK if the pixel is off.

Definition at line 360 of file Arduboy2.cpp.

6.1.2.52 uint8_t Arduboy2::getTextBackground ( )

Get the currently set text background color.

Returns

The background color that will be used to display any following text.

See also

setTextBackground()

Definition at line 1336 of file Arduboy2.cpp.

6.1.2.53 uint8_t Arduboy2::getTextColor ( )

Get the currently set text foreground color.

Returns

The color that will be used to display any following text.

See also

setTextColor()

Definition at line 1326 of file Arduboy2.cpp.

6.1.2.54 uint8_t Arduboy2::getTextSize ( )

Get the currently set text size.

Returns

The size that will be used for any following text.

See also

setTextSize()

Definition at line 1347 of file Arduboy2.cpp.

6.1.2.55 bool Arduboy2::getTextWrap ( )

Get the currently set text wrap mode.

Returns

`true` if text wrapping is on, `false` if wrapping is off.

See also

setTextWrap()

Definition at line 1357 of file Arduboy2.cpp.

6.1.2.56 uint8_t Arduboy2Core::height ( ) [static], [inherited]

Get the height of the display in pixels.

Returns

The height of the display in pixels.

Note

In most cases, the defined value `HEIGHT` would be better to use instead of this function.

Definition at line 301 of file Arduboy2Core.cpp.

6.1.2.57 void Arduboy2Core::idle ( ) [static], [inherited]

Idle the CPU to save power.

This puts the CPU in *idle* sleep mode. You should call this as often as you can for the best power savings. The timer 0 overflow interrupt will wake up the chip every 1ms, so even at 60 FPS a well written app should be able to sleep maybe half the time in between rendering it's own frames.

Definition at line 266 of file Arduboy2Core.cpp.

6.1.2.58 void Arduboy2Base::initRandomSeed ( ) [inherited]

Seed the random number generator with a random value.

The Arduino random number generator is seeded with a random value derived from entropy from an ADC reading of a floating pin combined with the microseconds since boot.

This method is still most effective when called after a semi-random time, such as after a user hits a button to start a game or other semi-random event.

Definition at line 278 of file Arduboy2.cpp.

6.1.2.59 void Arduboy2Core::invert (bool *inverse*) [static], [inherited]

Invert the entire display or set it back to normal.

Parameters

inverse	`true` will invert the display. `false` will set the display to no-inverted.

Calling this function with a value of `true` will set the display to inverted mode. A pixel with a value of 0 will be on and a pixel set to 1 will be off.

Once in inverted mode, the display will remain this way until it is set back to non-inverted mode by calling this function with `false`.

Definition at line 407 of file Arduboy2Core.cpp.

6.1.2.60 bool Arduboy2Base::justPressed (uint8_t *button*) [inherited]

Check if a button has just been pressed.

Parameters

button	The button to test for. Only one button should be specified.

Returns

`true` if the specified button has just been pressed.

Return `true` if the given button was pressed between the latest call to `pollButtons()` and previous call to `pollButtons()`. If the button has been held down over multiple polls, this function will return `false`.

There is no need to check for the release of the button since it must have been released for this function to return `true` when pressed again.

This function should only be used to test a single button.

See also

pollButtons() justReleased()

Definition at line 1032 of file Arduboy2.cpp.

6.1.2.61 bool Arduboy2Base::justReleased (uint8_t *button*) [inherited]

Check if a button has just been released.

Parameters

button	The button to test for. Only one button should be specified.

Returns

`true` if the specified button has just been released.

Return `true` if the given button, having previously been pressed, was released between the latest call to `pollButtons()` and previous call to `pollButtons()`. If the button has remained released over multiple polls, this function will return `false`.

There is no need to check for the button having been pressed since it must have been previously pressed for this function to return `true` upon release.

This function should only be used to test a single button.

Note

There aren't many cases where this function would be needed. Wanting to know if a button has been released, without knowing when it was pressed, is uncommon.

See also

pollButtons() justPressed()

Definition at line 1037 of file Arduboy2.cpp.

6.1.2.62 void Arduboy2Core::LCDCommandMode ( ) [static], [inherited]

Put the display into command mode.

When placed in command mode, data that is sent to the display will be treated as commands.

See the SSD1306 controller and OLED display documents for available commands and command sequences.

Links:

- https://www.adafruit.com/datasheets/SSD1306.pdf
- http://www.buydisplay.com/download/manual/ER-OLED013-1_Series_Datasheet.pdf

Note

This is a low level function that is not intended for general use in a sketch. It has been made public and documented for use by derived classes.

See also

LCDDataMode() sendLCDCommand() SPItransfer()

Definition at line 220 of file Arduboy2Core.cpp.

6.1.2.63 void Arduboy2Core::LCDDataMode ( ) [static], [inherited]

Put the display into data mode.

When placed in data mode, data that is sent to the display will be considered as data to be displayed.

Note

This is a low level function that is not intended for general use in a sketch. It has been made public and documented for use by derived classes.

See also

LCDCommandMode() SPItransfer()

Definition at line 215 of file Arduboy2Core.cpp.

6.1.2.64 bool Arduboy2Base::nextFrame ( ) [inherited]

Indicate that it's time to render the next frame.

Returns

`true` if it's time for the next frame.

When this function returns `true`, the amount of time has elapsed to display the next frame, as specified by `setFrameRate()`.

This function will normally be called at the start of the rendering loop which would wait for `true` to be returned before rendering and displaying the next frame.

example:

```
void loop() {
  if (!arduboy.nextFrame()) {
    return; // go back to the start of the loop
  }
  // render and display the next frame
}
```

See also

setFrameRate() setFrameDuration() nextFrameDEV()

Definition at line 232 of file Arduboy2.cpp.

6.1.2.65 bool Arduboy2Base::nextFrameDEV ( ) [inherited]

Indicate that it's time to render the next frame, and visually indicate if the code is running slower than the desired frame rate. **FOR USE DURING DEVELOPMENT**

Returns

`true` if it's time for the next frame.

This function is intended to be used in place of `nextFrame()` during the development of a sketch. It does the same thing as `nextFrame()` but additionally will light the yellow TX LED (at the bottom, to the left of the USB connector) whenever a frame takes longer to generate than the time allotted per frame, as determined by the `setFrameRate()` function.

Therefore, whenever the TX LED comes on (while not communicating over USB), it indicates that the sketch is running slower than the desired rate set by `setFrameRate()`. In this case the developer may wish to set a slower frame rate, or reduce or optimize the code for such frames.

Note

Once a sketch is ready for release, it would be expected that `nextFrameDEV()` calls be restored to `nextFrame()`.

See also

nextFrame() cpuLoad() setFrameRate()

Definition at line 260 of file Arduboy2.cpp.

6.1.2.66 bool Arduboy2Base::notPressed (uint8_t *buttons*) [inherited]

Test if the specified buttons are not pressed.

Parameters

buttons	A bit mask indicating which buttons to test. (Can be a single button)

Returns

`true` if *all* buttons in the provided mask are currently released.

Read the state of the buttons and return `true` if all the buttons in the specified mask are currently released.

Example: `if (notPressed(UP_BUTTON))`

Note

This function does not perform any button debouncing.

Definition at line 1021 of file Arduboy2.cpp.

6.1.2.67 void Arduboy2Core::paint8Pixels (uint8_t *pixels*) [static], [inherited]

Paint 8 pixels vertically to the display.

Parameters

pixels	A byte whose bits specify a vertical column of 8 pixels.

A byte representing a vertical column of 8 pixels is written to the display at the current page and column address. The address is then incremented. The page/column address will wrap to the start of the display (the top left) when it increments past the end (lower right).

The least significant bit represents the top pixel in the column. A bit set to 1 is lit, 0 is unlit.

Example:

```
X = lit pixels, . = unlit pixels

blank()                          paint8Pixels() 0xFF, 0, 0xF0, 0, 0x0F
v TOP LEFT corner (8x9)          v TOP LEFT corner
. . . . . . . . (page 1)         X . . . X . . . (page 1)
. . . . . . . .                  X . . . X . . .
. . . . . . . .                  X . . . X . . .
. . . . . . . .                  X . . . X . . .
. . . . . . . .                  X . X . . . . .
. . . . . . . .                  X . X . . . . .
. . . . . . . .                  X . X . . . . .
. . . . . . . . (end of page 1)  X . X . . . . . (end of page 1)
. . . . . . . . (page 2)         . . . . . . . . (page 2)
```

Definition at line 306 of file Arduboy2Core.cpp.

6.1.2.68 void Arduboy2Core::paintScreen (const uint8_t * *image*) [static],[inherited]

Paints an entire image directly to the display from program memory.

Parameters

image	A byte array in program memory representing the entire contents of the display.

The contents of the specified array in program memory is written to the display. Each byte in the array represents a vertical column of 8 pixels with the least significant bit at the top. The bytes are written starting at the top left, progressing horizontally and wrapping at the end of each row, to the bottom right. The size of the array must exactly match the number of pixels in the entire display.

See also

paint8Pixels()

Definition at line 311 of file Arduboy2Core.cpp.

6.1.2.69 void Arduboy2Core::paintScreen (uint8_t *image[ ],* bool *clear* = false) [static],[inherited]

Paints an entire image directly to the display from an array in RAM.

Parameters

image	A byte array in RAM representing the entire contents of the display.
clear	If `true` the array in RAM will be cleared to zeros upon return from this function. If `false` the RAM buffer will remain unchanged. (optional; defaults to `false`)

The contents of the specified array in RAM is written to the display. Each byte in the array represents a vertical column of 8 pixels with the least significant bit at the top. The bytes are written starting at the top left, progressing horizontally and wrapping at the end of each row, to the bottom right. The size of the array must exactly match the number of pixels in the entire display.

If parameter `clear` is set to `true` the RAM array will be cleared to zeros after its contents are written to the display.

See also

paint8Pixels()

Definition at line 325 of file Arduboy2Core.cpp.

6.1.2.70 void Arduboy2Base::pollButtons ( ) [inherited]

Poll the buttons and track their state over time.

Read and save the current state of the buttons and also keep track of the button state when this function was previouly called. These states are used by the `justPressed()` and `justReleased()` functions to determine if a button has changed state between now and the previous call to `pollButtons()`.

This function should be called once at the start of each new frame.

The `justPressed()` and `justReleased()` functions rely on this function.

example:

```
void loop() {
  if (!arduboy.nextFrame()) {
    return;
  }
  arduboy.pollButtons();

  // use justPressed() as necessary to determine if a button was just pressed
```

Note

As long as the elapsed time between calls to this function is long enough, buttons will be naturally debounced. Calling it once per frame at a frame rate of 60 or lower (or possibly somewhat higher), should be sufficient.

See also

justPressed() justReleased()

Definition at line 1026 of file Arduboy2.cpp.

6.1.2.71 bool Arduboy2Base::pressed (uint8_t *buttons*) [inherited]

Test if the specified buttons are pressed.

Parameters

buttons	A bit mask indicating which buttons to test. (Can be a single button)

Returns

`true` if *all* buttons in the provided mask are currently pressed.

Read the state of the buttons and return `true` if all the buttons in the specified mask are being pressed.

Example: `if (pressed(LEFT_BUTTON + A_BUTTON))`

Note

This function does not perform any button debouncing.

Definition at line 1016 of file Arduboy2.cpp.

6.1.2.72 bool Arduboy2Base::readShowBootLogoFlag ( ) [inherited]

Read the "Show Boot Logo" flag in system EEPROM.

Returns

`true` if the flag is set to indicate that the boot logo sequence should be displayed. `false` if the flag is set to not display the boot logo sequence.

The "Show Boot Logo" flag is used to determine whether the system boot logo sequence is to be displayed when the system boots up. This function returns the value of this flag.

See also

writeShowBootLogoFlag() bootLogo()

Definition at line 1104 of file Arduboy2.cpp.

6.1.2.73 bool Arduboy2Base::readShowBootLogoLEDsFlag ( ) [inherited]

Read the "Show LEDs with boot logo" flag in system EEPROM.

Returns

`true` if the flag is set to indicate that the RGB LEDs should be flashed. `false` if the flag is set to leave the LEDs off.

The "Show LEDs with boot logo" flag is used to determine whether the RGB LEDs should be flashed in sequence while the boot logo is being displayed. This function returns the value of this flag.

See also

writeShowBootLogoLEDsFlag()

Definition at line 1130 of file Arduboy2.cpp.

6.1.2.74 bool Arduboy2Base::readShowUnitNameFlag ( ) [inherited]

Read the "Show Unit Name" flag in system EEPROM.

Returns

`true` if the flag is set to indicate that the unit name should be displayed. `false` if the flag is set to not display the unit name.

The "Show Unit Name" flag is used to determine whether the system unit name is to be displayed at the end of the boot logo sequence. This function returns the value of this flag.

See also

writeShowUnitNameFlag() writeUnitName() readUnitName() Arduboy2::bootLogoExtra()

Definition at line 1117 of file Arduboy2.cpp.

6.1.2.75 uint16_t Arduboy2Base::readUnitID ( ) [inherited]

Read the unit ID from system EEPROM.

Returns

The value of the unit ID stored in system EEPROM.

This function reads the unit ID that has been set in system EEPROM. The ID can be any value. It is intended to allow different units to be uniquely identified.

See also

writeUnitID() readUnitName()

Definition at line 1056 of file Arduboy2.cpp.

6.1.2.76 uint8_t Arduboy2Base::readUnitName (char * *name*) [inherited]

Read the unit name from system EEPROM.

Parameters

name	A pointer to a string array variable where the unit name will be placed. The string will be up to 6 characters and terminated with a null (0x00) character, so the provided array must be at least 7 bytes long.

Returns

The length of the string (0-6).

This function reads the unit name that has been set in system EEPROM. The name is in ASCII and can contain any values except 0xFF and the null (0x00) terminator value.

The name can be used for any purpose. It could identify the owner or give the unit itself a nickname. A sketch could use it to automatically fill in a name or initials in a high score table, or display it as the "player" when the opponent is the computer.

Note

Sketches can use the defined value `ARDUBOY_UNIT_NAME_LEN` instead of hard coding a 6 when working with the unit name. For example, to allocate a buffer and read the unit name into it:

```
// Buffer for maximum name length plus the terminator
char unitName[ARDUBOY_UNIT_NAME_LEN + 1];

// The actual name length
byte unitNameLength;

unitNameLength = arduboy.readUnitName(unitName);
```

See also

writeUnitName() readUnitID() Arduboy2::bootLogoExtra()

Definition at line 1068 of file Arduboy2.cpp.

6.1.2.77 void Arduboy2Core::safeMode ( ) [static], [inherited]

Allow upload when the bootloader "magic number" could be corrupted.

If the UP button is held when this function is entered, the RGB LED will be lit and timer 0 will be disabled, then the sketch will remain in a tight loop. This is to address a problem with uploading a new sketch, for sketches that interfere with the bootloader "magic number". The problem occurs with certain sketches that use large amounts of RAM.

This function should be called after `boot()` in sketches that potentially could cause the problem.

It is intended to replace the `flashlight()` function when more program space is required. If possible, it is more desirable to use `flashlight()`, so that the actual flashlight feature isn't lost.

See also

Arduboy2Base::flashlight() boot()

Definition at line 247 of file Arduboy2Core.cpp.

6.1.2.78 void Arduboy2Core::sendLCDCommand (uint8_t *command*) [static], [inherited]

Send a single command byte to the display.

Parameters

command	The command byte to send to the display.

The display will be set to command mode then the specified command byte will be sent. The display will then be set to data mode. Multi-byte commands can be sent by calling this function multiple times.

Note

Sending improper commands to the display can place it into invalid or unexpected states, possibly even causing physical damage.

Definition at line 398 of file Arduboy2Core.cpp.

6.1.2.79 void Arduboy2::setCursor (int16_t *x*, int16_t *y*)

Set the location of the text cursor.

Parameters

x	The X coordinate, in pixels, for the new location of the text cursor.
y	The Y coordinate, in pixels, for the new location of the text cursor.

The location of the text cursor is set the the specified coordinates. The coordinates are in pixels. Since the coordinates can specify any pixel location, the text does not have to be placed on specific rows. As with all drawing functions, location 0, 0 is the top left corner of the display. The cursor location will be the top left corner of the next character written.

See also

getCursorX() getCursorY()

Definition at line 1305 of file Arduboy2.cpp.

6.1.2.80 void Arduboy2Base::setFrameDuration (uint8_t *duration*) [inherited]

Set the frame rate, used by the frame control functions, by giving the duration of each frame.

Parameters

duration	The desired duration of each frame in milliseconds.

Set the frame rate by specifying the duration of each frame in milliseconds. This is used by `nextFrame()` to update frames at a given rate. If this function or `setFrameRate()` isn't used, the default will be 16ms per frame.

Normally, the frame rate would be set to the desired value once, at the start of the game, but it can be changed at any time to alter the frame update rate.

See also

nextFrame() setFrameRate()

Definition at line 222 of file Arduboy2.cpp.

6.1.2.81 void Arduboy2Base::setFrameRate (uint8_t *rate*) [inherited]

Set the frame rate used by the frame control functions.

Parameters

rate	The desired frame rate in frames per second.

Set the frame rate, in frames per second, used by `nextFrame()` to update frames at a given rate. If this function or `setFrameDuration()` isn't used, the default rate will be 60 (actually 62.5, see note below).

Normally, the frame rate would be set to the desired value once, at the start of the game, but it can be changed at any time to alter the frame update rate.

Note

The given rate is internally converted to a frame duration in milliseconds, rounded down to the nearest integer. Therefore, the actual rate will be equal to or higher than the rate given.
For example, 60 FPS would be 16.67ms per frame. This will be rounded down to 16ms, giving an actual frame rate of 62.5 FPS.

See also

nextFrame() setFrameDuration()

Definition at line 217 of file Arduboy2.cpp.

6.1.2.82 void Arduboy2Core::setRGBled (uint8_t *red,* uint8_t *green,* uint8_t *blue*) [static], [inherited]

Set the light output of the RGB LED.

Parameters

red,green,blue	The brightness value for each LED.

The RGB LED is actually individual red, green and blue LEDs placed very close together in a single package. By setting the brightness of each LED, the RGB LED can show various colors and intensities. The brightness of each LED can be set to a value from 0 (fully off) to 255 (fully on).

Note

Certain libraries that take control of the hardware timers may interfere with the ability of this function to properly control the RGB LED. *ArduboyPlaytune* is one such library known to do this. The digitalWriteRGB() function will still work properly in this case.

Note

Many of the Kickstarter Arduboys were accidentally shipped with the RGB LED installed incorrectly. For these units, the green LED cannot be lit. As long as the green led is set to off, setting the red LED will actually control the blue LED and setting the blue LED will actually control the red LED. If the green LED is turned fully on, none of the LEDs will light.

See also

setRGBled(uint8_t, uint8_t) digitalWriteRGB() freeRGBled()

Definition at line 433 of file Arduboy2Core.cpp.

6.1.2.83 void Arduboy2Core::setRGBled (uint8_t *color,* uint8_t *val*) [static],[inherited]

Set the brightness of one of the RGB LEDs without affecting the others.

Parameters

color	The name of the LED to set. The value given should be one of RED_LED, GREEN_LED or BLUE_LED.
val	The brightness value for the LED, from 0 to 255.

Note

In order to use this function, the 3 parameter version must first be called at least once, in order to initialize the hardware.

This 2 parameter version of the function will set the brightness of a single LED within the RGB LED without affecting the current brightness of the other two. See the description of the 3 parameter version of this function for more details on the RGB LED.

See also

setRGBled(uint8_t, uint8_t, uint8_t) digitalWriteRGB() freeRGBled()

Definition at line 455 of file Arduboy2Core.cpp.

6.1.2.84 void Arduboy2::setTextBackground (uint8_t *bg*)

Set the text background color.

Parameters

bg	The background color to be used for following text.

See also

setTextColor() getTextBackground()

Definition at line 1331 of file Arduboy2.cpp.

6.1.2.85 void Arduboy2::setTextColor (uint8_t *color*)

Set the text foreground color.

Parameters

color	The color to be used for following text.

See also

setTextBackground() getTextColor()

Definition at line 1321 of file Arduboy2.cpp.

6.1.2.86 void Arduboy2::setTextSize (uint8_t *s*)

Set the text character size.

Parameters

s	The text size multiplier. Must be 1 or higher.

Setting a text size of 1 will result in standard size characters which occupy 6x8 pixels (the result of 5x7 characters with spacing on the right and bottom edges).

The value specified is a multiplier. A value of 2 will double the size so they will occupy 12x16 pixels. A value of 3 will result in 18x24, etc.

See also

getTextSize()

Definition at line 1341 of file Arduboy2.cpp.

6.1.2.87 void Arduboy2::setTextWrap (bool *w*)

Set or disable text wrap mode.

Parameters

w	`true` enables text wrap mode. `false` disables it.

Text wrap mode is enabled by specifying `true`. In wrap mode, the text cursor will be moved to the start of the next line (based on the current text size) if the following character wouldn't fit entirely at the end of the current line.

If wrap mode is disabled, characters will continue to be written to the same line. A character at the right edge of the screen may only be partially displayed and additional characters will be off screen.

See also

getTextWrap()

Definition at line 1352 of file Arduboy2.cpp.

6.1.2.88 void Arduboy2Core::SPItransfer (uint8_t *data*) [static], [inherited]

Transfer a byte to the display.

Parameters

data	The byte to be sent to the display.

Transfer one byte to the display over the SPI port and wait for the transfer to complete. The byte will either be interpreted as a command or as data to be placed on the screen, depending on the command/data mode.

See also

LCDDataMode() LCDCommandMode() sendLCDCommand()

Definition at line 234 of file Arduboy2Core.cpp.

6.1.2.89 void Arduboy2Base::systemButtons ( ) [inherited]

Handle buttons held on startup for system control.

This function is called by `begin()` and can be called by a sketch after `boot()`.

Hold the B button when booting to enter system control mode. The B button must be held continuously to remain in this mode. Then, pressing other buttons will perform system control functions:

- UP: Set "sound enabled" in EEPROM
- DOWN: Set "sound disabled" (mute) in EEPROM

See also

begin() boot()

Definition at line 76 of file Arduboy2.cpp.

6.1.2.90 void Arduboy2Base::waitNoButtons ( ) [inherited]

Wait until all buttons have been released.

This function is called by `begin()` and can be called by a sketch after `boot()`.

It won't return unless no buttons are being pressed. A short delay is performed each time before testing the state of the buttons to do a simple button debounce.

This function is called at the end of `begin()` to make sure no buttons used to perform system start up actions are still being pressed, to prevent them from erroneously being detected by the sketch code itself.

See also

begin() boot()

Definition at line 209 of file Arduboy2.cpp.

6.1.2.91 uint8_t Arduboy2Core::width ( ) [static], [inherited]

Get the width of the display in pixels.

Returns

The width of the display in pixels.

Note

In most cases, the defined value `WIDTH` would be better to use instead of this function.

Definition at line 299 of file Arduboy2Core.cpp.

6.1.2.92 size_t Arduboy2::write (uint8_t *c*) [virtual]

Write a single ASCII character at the current text cursor location.

Parameters

c	The ASCII value of the character to be written.

Returns

The number of characters written (will always be 1).

This is the Arduboy implemetation of the Arduino virtual `write()` function. The single ASCII character specified is written to the the screen buffer at the current text cursor. The text cursor is then moved to the next character position in the screen buffer. This new cursor position will depend on the current text size and possibly the current wrap mode.

Two special characters are handled:

- The newline character `\n`. This will move the text cursor to the start of the next line based on the current text size.
- The carriage return character `\r`. This character will be ignored.

Note

This function is rather low level and, although it's available as a public function, it wouldn't normally be used. In most cases the Arduino Print class should be used for writing text.

See also

Print setTextSize() setTextWrap()

Definition at line 1241 of file Arduboy2.cpp.

6.1.2.93 void Arduboy2Base::writeShowBootLogoFlag (bool *val*) [inherited]

Write the "Show Boot Logo" flag in system EEPROM.

Parameters

val	If `true` the flag is set to indicate that the boot logo sequence should be displayed. If `false` the flag is set to not display the boot logo sequence.

The "Show Boot Logo" flag is used to determine whether the system boot logo sequence is to be displayed when the system boots up. This function allows the flag to be saved with the desired value.

See also

readShowBootLogoFlag() bootLogo()

Definition at line 1109 of file Arduboy2.cpp.

6.1.2.94 void Arduboy2Base::writeShowBootLogoLEDsFlag (bool *val*) [inherited]

Write the "Show LEDs with boot logo" flag in system EEPROM.

Parameters

val	If `true` the flag is set to indicate that the RGB LEDs should be flashed. If `false` the flag is set to leave the LEDs off.

The "Show LEDs with boot logo" flag is used to determine whether the RGB LEDs should be flashed in sequence while the boot logo is being displayed. This function allows the flag to be saved with the desired value.

See also

readShowBootLogoLEDsFlag()

Definition at line 1135 of file Arduboy2.cpp.

6.1.2.95 void Arduboy2Base::writeShowUnitNameFlag (bool *val*) [inherited]

Write the "Show Unit Name" flag in system EEPROM.

Parameters

val	If `true` the flag is set to indicate that the unit name should be displayed. If `false` the flag is set to not display the unit name.

The "Show Unit Name" flag is used to determine whether the system unit name is to be displayed at the end of the boot logo sequence. This function allows the flag to be saved with the desired value.

See also

readShowUnitNameFlag() writeUnitName() readUnitName() Arduboy2::bootLogoExtra()

Definition at line 1122 of file Arduboy2.cpp.

6.1.2.96 void Arduboy2Base::writeUnitID (uint16_t *id*) [inherited]

Write a unit ID to system EEPROM.

Parameters

id	The value of the unit ID to be stored in system EEPROM.

This function writes a unit ID to a reserved location in system EEPROM. The ID can be any value. It is intended to allow different units to be uniquely identified.

See also

readUnitID() writeUnitName()

Definition at line 1062 of file Arduboy2.cpp.

6.1.2.97 void Arduboy2Base::writeUnitName (char * *name*) [inherited]

Write a unit name to system EEPROM.

Parameters

name	A pointer to a string array variable containing the unit name to be saved. The string can be up to 6 characters and must be terminated with a null (0x00) character. It can contain any values except 0xFF.

This function writes a unit name to a reserved area in system EEPROM. The name is in ASCII and can contain any values except 0xFF and the null (0x00) terminator value. The newline character (LF, \n, 0x0A) and carriage return character (CR, \r, 0x0D) should also be avoided.

The name can be used for any purpose. It could identify the owner or give the unit itself a nickname. A sketch could use it to automatically fill in a name or initials in a high score table, or display it as the "player" when the opponent is the computer.

Note

Sketches can use the defined value `ARDUBOY_UNIT_NAME_LEN` instead of hard coding a 6 when working with the unit name.

See also

readUnitName() writeUnitID() Arduboy2::bootLogoExtra()

Definition at line 1088 of file Arduboy2.cpp.

6.1.3 Member Data Documentation

6.1.3.1 Arduboy2Audio Arduboy2Base::audio [inherited]

An object created to provide audio control functions within this class.

This object is created to eliminate the need for a sketch to create an Arduboy2Audio class object itself.

See also

Arduboy2Audio

Definition at line 182 of file Arduboy2.h.

6.1.3.2 uint16_t Arduboy2Base::frameCount [inherited]

A counter which is incremented once per frame.

This counter is incremented once per frame when using the `nextFrame()` function. It will wrap to zero when it reaches its maximum value.

It could be used to have an event occur for a given number of frames, or a given number of frames later, in a way that wouldn't be quantized the way that using `everyXFrames()` might.

example:

```
// move for 10 frames when right button is pressed, if not already moving
if (!moving) {
  if (arduboy.justPressed(RIGHT_BUTTON)) {
    endMoving = arduboy.frameCount + 10;
    moving = true;
  }
} else {
  movePlayer();
  if (arduboy.frameCount == endMoving) {
    moving = false;
  }
}
```

This counter could also be used to determine the number of frames that have elapsed between events but the possibility of the counter wrapping would have to be accounted for.

See also

nextFrame() everyXFrames()

Definition at line 1240 of file Arduboy2.h.

6.1.3.3 uint8_t Arduboy2Base::sBuffer [static],[inherited]

The display buffer array in RAM.

The display buffer (also known as the screen buffer) contains an image bitmap of the desired contents of the display, which is written to the display using the `display()` function. The drawing functions of this library manipulate the contents of the display buffer. A sketch can also access the display buffer directly.

See also

getBuffer()

Definition at line 1254 of file Arduboy2.h.

The documentation for this class was generated from the following files:

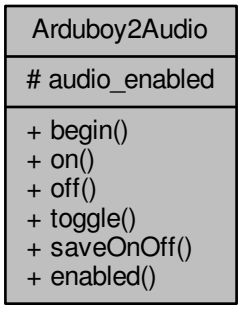
- src/Arduboy2.h
- src/Arduboy2.cpp

6.2 Arduboy2Audio Class Reference

Provide speaker and sound control.

`#include <Arduboy2Audio.h>`

Collaboration diagram for Arduboy2Audio:



Static Public Member Functions

- static void begin ()
  Initialize the speaker based on the current mute setting.
- static void on ()
  Turn sound on.
- static void off ()
  Turn sound off (mute).
- static void toggle ()
  Toggle the sound on/off state.
- static void saveOnOff ()
  Save the current sound state in EEPROM.
- static bool enabled ()
  Get the current sound state.

6.2.1 Detailed Description

Provide speaker and sound control.

This class provides functions to initialize the speaker and control the enabling and disabling (muting) of sound. It doesn't provide any functions to actually produce sound.

The state of sound muting is stored in system EEPROM and so is retained over power cycles.

An Arduboy2Audio class object named `audio` will be created by the Arduboy2Base class, so there is no need for a sketch itself to create an Arduboy2Audio object. Arduboy2Audio functions can be called using the Arduboy2 or Arduboy2Base `audio` object.

Example:

```
#include <Arduboy2.h>

Arduboy2 arduboy;

// Arduboy2Audio functions can be called as follows:
  arduboy.audio.on();
  arduboy.audio.off();
```

Note

In order for this class to be fully functional, the external library or functions used by a sketch to actually to produce sounds should be compliant with this class. This means they should only produce sound if it is enabled, or mute the sound if it's disabled. The `enabled()` function can be used to determine if sound is enabled or muted. Generally a compliant library would accept the `enabled()` function as an initialization parameter and then call it as necessary to determine the current state.
For example, the ArduboyTones and ArduboyPlaytune libraries require an `enabled()` type function to be passed as a parameter in the constructor, like so:

```
#include <Arduboy2.h>
#include <ArduboyTones.h>

Arduboy2 arduboy;
ArduboyTones sound(arduboy.audio.enabled);
```

Note

A friend class named *Arduboy2Ex* is declared by this class. The intention is to allow a sketch to create an *Arduboy2Ex* class which would have access to the private and protected members of the Arduboy2Audio class. It is hoped that this may eliminate the need to create an entire local copy of the library, in order to extend the functionality, in most circumstances.

Definition at line 73 of file Arduboy2Audio.h.

6.2.2 Member Function Documentation

6.2.2.1 void Arduboy2Audio::begin ( ) [static]

Initialize the speaker based on the current mute setting.

The speaker is initialized based on the current mute setting saved in system EEPROM. This function is called by `Arduboy2Base::begin()` so it isn't normally required to call it within a sketch. However, if `Arduboy2Core::boot()` is used instead of `Arduboy2Base::begin()` and the sketch includes sound, then this function should be called after `boot()`.

Definition at line 49 of file Arduboy2Audio.cpp.

6.2.2.2 bool Arduboy2Audio::enabled ( ) [static]

Get the current sound state.

Returns

`true` if sound is currently enabled (not muted).

This function should be used by code that actually generates sound. If `true` is returned, sound can be produced. If `false` is returned, sound should be muted.

See also

on() off() toggle()

Definition at line 57 of file Arduboy2Audio.cpp.

6.2.2.3 void Arduboy2Audio::off ( ) [static]

Turn sound off (mute).

The system is configured to not produce sound (mute). This function sets the sound mode only until the unit is powered off. To save the current mode use `saveOnOff()`.

See also

on() toggle() saveOnOff()

Definition at line 24 of file Arduboy2Audio.cpp.

6.2.2.4 void Arduboy2Audio::on ( ) [static]

Turn sound on.

The system is configured to generate sound. This function sets the sound mode only until the unit is powered off. To save the current mode use `saveOnOff()`.

See also

off() toggle() saveOnOff()

Definition at line 12 of file Arduboy2Audio.cpp.

6.2.2.5 void Arduboy2Audio::saveOnOff ( ) [static]

Save the current sound state in EEPROM.

The current sound state, set by `on()` or `off()`, is saved to the reserved system area in EEPROM. This allows the state to carry over between power cycles and after uploading a different sketch.

Note

EEPROM is limited in the number of times it can be written to. Sketches should not continuously change and then save the state rapidly.

See also

on() off() toggle()

Definition at line 44 of file Arduboy2Audio.cpp.

6.2.2.6 void Arduboy2Audio::toggle ( ) [static]

Toggle the sound on/off state.

If the system is configured for sound on, it will be changed to sound off (mute). If sound is off, it will be changed to on. This function sets the sound mode only until the unit is powered off. To save the current mode use `saveOnOff()`.

See also

on() off() saveOnOff()

Definition at line 36 of file Arduboy2Audio.cpp.

The documentation for this class was generated from the following files:

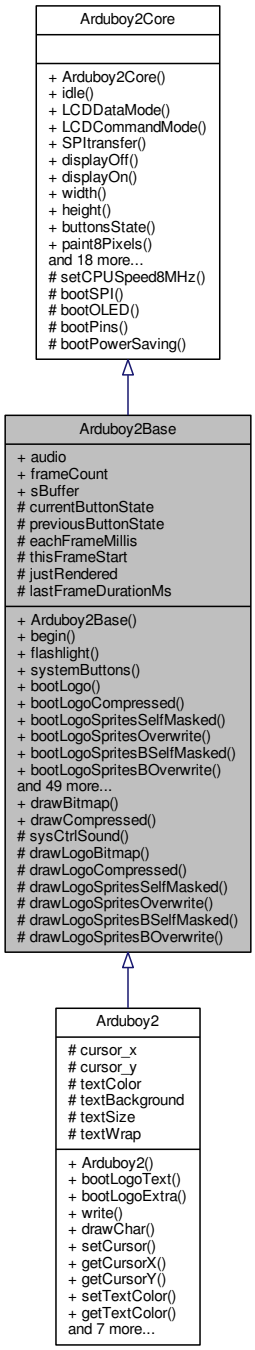
- src/Arduboy2Audio.h
- src/Arduboy2Audio.cpp

6.3 Arduboy2Base Class Reference

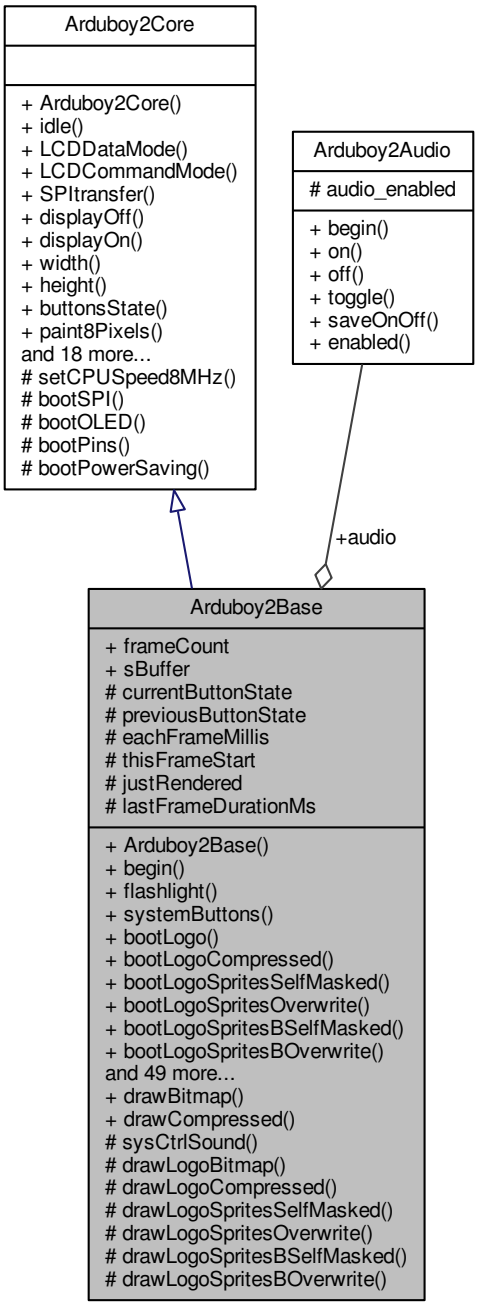
The main functions provided for writing sketches for the Arduboy, *minus* text output.

```
#include <Arduboy2.h>
```

Inheritance diagram for Arduboy2Base:



Collaboration diagram for Arduboy2Base:



Public Member Functions

- void begin ()

  Initialize the hardware, display the boot logo, provide boot utilities, etc.
- void flashlight ()

  Turn the RGB LED and display fully on to act as a small flashlight/torch.
- void systemButtons ()

Handle buttons held on startup for system control.

- void bootLogo ()

  Display the boot logo sequence using `drawBitmap()`.
- void bootLogoCompressed ()

  Display the boot logo sequence using `drawCompressed()`.
- void bootLogoSpritesSelfMasked ()

  Display the boot logo sequence using `Sprites::drawSelfMasked()`.
- void bootLogoSpritesOverwrite ()

  Display the boot logo sequence using `Sprites::drawOverwrite()`.
- void bootLogoSpritesBSelfMasked ()

  Display the boot logo sequence using `SpritesB::drawSelfMasked()`.
- void bootLogoSpritesBOverwrite ()

  Display the boot logo sequence using `SpritesB::drawOverwrite()`.
- void bootLogoShell (void(*drawLogo)(int16_t))

  Display the boot logo sequence using the provided function.
- void waitNoButtons ()

  Wait until all buttons have been released.
- void clear ()

  Clear the display buffer.
- void display ()

  Copy the contents of the display buffer to the display.
- void display (bool clear)

  Copy the contents of the display buffer to the display. The display buffer can optionally be cleared.
- void drawPixel (int16_t x, int16_t y, uint8_t color=WHITE)

  Set a single pixel in the display buffer to the specified color.
- uint8_t getPixel (uint8_t x, uint8_t y)

  Returns the state of the given pixel in the screen buffer.
- void drawCircle (int16_t x0, int16_t y0, uint8_t r, uint8_t color=WHITE)

  Draw a circle of a given radius.
- void fillCircle (int16_t x0, int16_t y0, uint8_t r, uint8_t color=WHITE)

  Draw a filled-in circle of a given radius.
- void drawLine (int16_t x0, int16_t y0, int16_t x1, int16_t y1, uint8_t color=WHITE)

  Draw a line between two specified points.
- void drawRect (int16_t x, int16_t y, uint8_t w, uint8_t h, uint8_t color=WHITE)

  Draw a rectangle of a specified width and height.
- void drawFastVLine (int16_t x, int16_t y, uint8_t h, uint8_t color=WHITE)

  Draw a vertical line.
- void drawFastHLine (int16_t x, int16_t y, uint8_t w, uint8_t color=WHITE)

  Draw a horizontal line.
- void fillRect (int16_t x, int16_t y, uint8_t w, uint8_t h, uint8_t color=WHITE)

  Draw a filled-in rectangle of a specified width and height.
- void fillScreen (uint8_t color=WHITE)

  Fill the screen buffer with the specified color.
- void drawRoundRect (int16_t x, int16_t y, uint8_t w, uint8_t h, uint8_t r, uint8_t color=WHITE)

  Draw a rectangle with rounded corners.
- void fillRoundRect (int16_t x, int16_t y, uint8_t w, uint8_t h, uint8_t r, uint8_t color=WHITE)

  Draw a filled-in rectangle with rounded corners.
- void drawTriangle (int16_t x0, int16_t y0, int16_t x1, int16_t y1, int16_t x2, int16_t y2, uint8_t color=WHITE)

  Draw a triangle given the coordinates of each corner.
- void fillTriangle (int16_t x0, int16_t y0, int16_t x1, int16_t y1, int16_t x2, int16_t y2, uint8_t color=WHITE)

  Draw a filled-in triangle given the coordinates of each corner.

- void drawSlowXYBitmap (int16_t x, int16_t y, const uint8_t *bitmap, uint8_t w, uint8_t h, uint8_t color=WHITE)

  Draw a bitmap from a horizontally oriented array in program memory.
- uint8_t * getBuffer ()

  Get a pointer to the display buffer in RAM.
- void initRandomSeed ()

  Seed the random number generator with a random value.
- void setFrameRate (uint8_t rate)

  Set the frame rate used by the frame control functions.
- void setFrameDuration (uint8_t duration)

  Set the frame rate, used by the frame control functions, by giving the duration of each frame.
- bool nextFrame ()

  Indicate that it's time to render the next frame.
- bool nextFrameDEV ()

  Indicate that it's time to render the next frame, and visually indicate if the code is running slower than the desired frame rate. ***FOR USE DURING DEVELOPMENT***
- bool everyXFrames (uint8_t frames)

  Indicate if the specified number of frames has elapsed.
- int cpuLoad ()

  Return the load on the CPU as a percentage.
- bool pressed (uint8_t buttons)

  Test if the specified buttons are pressed.
- bool notPressed (uint8_t buttons)

  Test if the specified buttons are not pressed.
- void pollButtons ()

  Poll the buttons and track their state over time.
- bool justPressed (uint8_t button)

  Check if a button has just been pressed.
- bool justReleased (uint8_t button)

  Check if a button has just been released.
- bool collide (Point point, Rect rect)

  Test if a point falls within a rectangle.
- bool collide (Rect rect1, Rect rect2)

  Test if a rectangle is intersecting with another rectangle.
- uint16_t readUnitID ()

  Read the unit ID from system EEPROM.
- void writeUnitID (uint16_t id)

  Write a unit ID to system EEPROM.
- uint8_t readUnitName (char *name)

  Read the unit name from system EEPROM.
- void writeUnitName (char *name)

  Write a unit name to system EEPROM.
- bool readShowBootLogoFlag ()

  Read the "Show Boot Logo" flag in system EEPROM.
- void writeShowBootLogoFlag (bool val)

  Write the "Show Boot Logo" flag in system EEPROM.
- bool readShowUnitNameFlag ()

  Read the "Show Unit Name" flag in system EEPROM.
- void writeShowUnitNameFlag (bool val)

  Write the "Show Unit Name" flag in system EEPROM.
- bool readShowBootLogoLEDsFlag ()

  Read the "Show LEDs with boot logo" flag in system EEPROM.
- void writeShowBootLogoLEDsFlag (bool val)

  Write the "Show LEDs with boot logo" flag in system EEPROM.

Static Public Member Functions

- static void drawBitmap (int16_t x, int16_t y, const uint8_t *bitmap, uint8_t w, uint8_t h, uint8_t color=WHITE)

  Draw a bitmap from an array in program memory.
- static void drawCompressed (int16_t sx, int16_t sy, const uint8_t *bitmap, uint8_t color=WHITE)

  Draw a bitmap from an array of compressed data.
- static void idle ()

  Idle the CPU to save power.
- static void LCDDataMode ()

  Put the display into data mode.
- static void LCDCommandMode ()

  Put the display into command mode.
- static void SPItransfer (uint8_t data)

  Transfer a byte to the display.
- static void displayOff ()

  Turn the display off.
- static void displayOn ()

  Turn the display on.
- static uint8_t width ()

  Get the width of the display in pixels.
- static uint8_t height ()

  Get the height of the display in pixels.
- static uint8_t buttonsState ()

  Get the current state of all buttons as a bitmask.
- static void paint8Pixels (uint8_t pixels)

  Paint 8 pixels vertically to the display.
- static void paintScreen (const uint8_t *image)

  Paints an entire image directly to the display from program memory.
- static void paintScreen (uint8_t image[ ], bool clear=false)

  Paints an entire image directly to the display from an array in RAM.
- static void blank ()

  Blank the display screen by setting all pixels off.
- static void invert (bool inverse)

  Invert the entire display or set it back to normal.
- static void allPixelsOn (bool on)

  Turn all display pixels on or display the buffer contents.
- static void flipVertical (bool flipped)

  Flip the display vertically or set it back to normal.
- static void flipHorizontal (bool flipped)

  Flip the display horizontally or set it back to normal.
- static void sendLCDCommand (uint8_t command)

  Send a single command byte to the display.
- static void setRGBled (uint8_t red, uint8_t green, uint8_t blue)

  Set the light output of the RGB LED.
- static void setRGBled (uint8_t color, uint8_t val)

  Set the brightness of one of the RGB LEDs without affecting the others.
- static void freeRGBled ()

  Relinquish analog control of the RGB LED.
- static void digitalWriteRGB (uint8_t red, uint8_t green, uint8_t blue)

  Set the RGB LEDs digitally, to either fully on or fully off.
- static void digitalWriteRGB (uint8_t color, uint8_t val)

Set one of the RGB LEDs digitally, to either fully on or fully off.

- static void boot ()

  Initialize the Arduboy's hardware.
- static void safeMode ()

  Allow upload when the bootloader "magic number" could be corrupted.
- static void delayShort (uint16_t ms) __attribute__((noinline))

  Delay for the number of milliseconds, specified as a 16 bit value.
- static void exitToBootloader ()

  Exit the sketch and start the bootloader.

Public Attributes

- Arduboy2Audio audio

  An object created to provide audio control functions within this class.
- uint16_t frameCount

  A counter which is incremented once per frame.

Static Public Attributes

- static uint8_t sBuffer [(HEIGHT ∗WIDTH)/8]

  The display buffer array in RAM.

6.3.1 Detailed Description

The main functions provided for writing sketches for the Arduboy, *minus* text output.

This class in inherited by Arduboy2, so if text output functions are required Arduboy2 should be used instead.

Note

An Arduboy2Audio class object named `audio` will be created by the Arduboy2Base class, so there is no need for a sketch itself to create an Arduboy2Audio object. Arduboy2Audio functions can be called using the Arduboy2 or Arduboy2Base `audio` object.
Example:

```
#include <Arduboy2.h>

Arduboy2 arduboy;

// Arduboy2Audio functions can be called as follows:
  arduboy.audio.on();
  arduboy.audio.off();
```

Note

A friend class named *Arduboy2Ex* is declared by this class. The intention is to allow a sketch to create an *Arduboy2Ex* class which would have access to the private and protected members of the Arduboy2Base class. It is hoped that this may eliminate the need to create an entire local copy of the library, in order to extend the functionality, in most circumstances.

See also

Arduboy2

Definition at line 166 of file Arduboy2.h.

6.3.2 Member Function Documentation

6.3.2.1 void Arduboy2Core::allPixelsOn (bool *on*) [static], [inherited]

Turn all display pixels on or display the buffer contents.

Parameters

on	`true` turns all pixels on. `false` displays the contents of the hardware display buffer.

Calling this function with a value of `true` will override the contents of the hardware display buffer and turn all pixels on. The contents of the hardware buffer will remain unchanged.

Calling this function with a value of `false` will set the normal state of displaying the contents of the hardware display buffer.

Note

All pixels will be lit even if the display is in inverted mode.

See also

invert()

Definition at line 414 of file Arduboy2Core.cpp.

6.3.2.2 void Arduboy2Base::begin ( )

Initialize the hardware, display the boot logo, provide boot utilities, etc.

This function should be called once near the start of the sketch, usually in `setup()`, before using any other functions in this class. It initializes the display, displays the boot logo, provides "flashlight" and system control features and initializes audio control.

Note

To free up some code space for use by the sketch, `boot()` can be used instead of `begin()` to allow the elimination of some of the things that aren't really required, such as displaying the boot logo.

See also

boot()

Definition at line 30 of file Arduboy2.cpp.

6.3.2.3 void Arduboy2Core::blank ( ) [static], [inherited]

Blank the display screen by setting all pixels off.

All pixels on the screen will be written with a value of 0 to turn them off.

Definition at line 392 of file Arduboy2Core.cpp.

6.3.2.4 void Arduboy2Core::boot ( ) [static], [inherited]

Initialize the Arduboy's hardware.

This function initializes the display, buttons, etc.

This function is called by begin() so isn't normally called within a sketch. However, in order to free up some code space, by eliminating some of the start up features, it can be called in place of begin(). The functions that begin() would call after boot() can then be called to add back in some of the start up features, if desired. See the README file or documentation on the main page for more details.

See also

Arduboy2Base::begin()

Definition at line 76 of file Arduboy2Core.cpp.

6.3.2.5 void Arduboy2Base::bootLogo ( )

Display the boot logo sequence using `drawBitmap()`.

This function is called by `begin()` and can be called by a sketch after `boot()`.

The Arduboy logo scrolls down from the top of the screen to the center while the RGB LEDs light in sequence.

The `bootLogoShell()` helper function is used to perform the actual sequence. The documentation for `bootLogoShell()` provides details on how it operates.

See also

begin() boot() bootLogoShell() Arduboy2::bootLogoText()

Definition at line 102 of file Arduboy2.cpp.

6.3.2.6 void Arduboy2Base::bootLogoCompressed ( )

Display the boot logo sequence using `drawCompressed()`.

This function can be called by a sketch after `boot()` as an alternative to `bootLogo()`. This may reduce code size if the sketch itself uses `drawCompressed()`.

See also

bootLogo() begin() boot()

Definition at line 112 of file Arduboy2.cpp.

6.3.2.7 void Arduboy2Base::bootLogoShell (void(*)(int16_t) *drawLogo*)

Display the boot logo sequence using the provided function.

Parameters

drawLogo	A reference to a function which will draw the boot logo at the given Y position.

This common function executes the sequence to display the boot logo. It is called by `bootLogo()` and other similar functions which provide it with a reference to a function which will do the actual drawing of the logo.

This function calls `bootLogoExtra()` after the logo stops scrolling down, which derived classes can implement to add additional information to the logo screen. The `Arduboy2` class uses this to display the unit name.

If the RIGHT button is pressed while the logo is scrolling down, the boot logo sequence will be aborted. This can be useful for developers who wish to quickly start testing, or anyone else who is impatient and wants to go straight to the actual sketch.

If the SYS_FLAG_SHOW_LOGO_LEDS flag in system EEPROM is cleared, the RGB LEDs will not be flashed during the logo display sequence.

If the SYS_FLAG_SHOW_LOGO flag in system EEPROM is cleared, this function will return without executing the logo display sequence.

The prototype for the function provided to draw the logo is:

```
void drawLogo(int16_t y);
```

The y parameter is the Y offset for the top of the logo. It is expected that the logo will be 16 pixels high and centered horizontally. This will result in the logo stopping in the middle of the screen at the end of the sequence. If the logo height is not 16 pixels, the Y value can be adjusted to compensate.

See also

bootLogo() boot() Arduboy2::bootLogoExtra()

Definition at line 164 of file Arduboy2.cpp.

6.3.2.8 void Arduboy2Base::bootLogoSpritesBOverwrite ( )

Display the boot logo sequence using `SpritesB::drawOverwrite()`.

This function can be called by a sketch after `boot()` as an alternative to `bootLogo()`. This may reduce code size if the sketch itself uses `SpritesB` class functions.

See also

bootLogo() begin() boot() SpritesB

Definition at line 152 of file Arduboy2.cpp.

6.3.2.9 void Arduboy2Base::bootLogoSpritesBSelfMasked ( )

Display the boot logo sequence using `SpritesB::drawSelfMasked()`.

This function can be called by a sketch after `boot()` as an alternative to `bootLogo()`. This may reduce code size if the sketch itself uses `SpritesB` class functions.

See also

bootLogo() begin() boot() SpritesB

Definition at line 142 of file Arduboy2.cpp.

6.3.2.10 void Arduboy2Base::bootLogoSpritesOverwrite ( )

Display the boot logo sequence using `Sprites::drawOverwrite()`.

This function can be called by a sketch after `boot()` as an alternative to `bootLogo()`. This may reduce code size if the sketch itself uses `Sprites` class functions.

See also

bootLogo() begin() boot() Sprites

Definition at line 132 of file Arduboy2.cpp.

6.3.2.11 void Arduboy2Base::bootLogoSpritesSelfMasked ( )

Display the boot logo sequence using `Sprites::drawSelfMasked()`.

This function can be called by a sketch after `boot()` as an alternative to `bootLogo()`. This may reduce code size if the sketch itself uses `Sprites` class functions.

See also

bootLogo() begin() boot() Sprites

Definition at line 122 of file Arduboy2.cpp.

6.3.2.12 uint8_t Arduboy2Core::buttonsState ( ) [static], [inherited]

Get the current state of all buttons as a bitmask.

Returns

A bitmask of the state of all the buttons.

The returned mask contains a bit for each button. For any pressed button, its bit will be 1. For released buttons their associated bits will be 0.

The following defined mask values should be used for the buttons:

LEFT_BUTTON, RIGHT_BUTTON, UP_BUTTON, DOWN_BUTTON, A_BUTTON, B_BUTTON

Definition at line 528 of file Arduboy2Core.cpp.

6.3.2.13 void Arduboy2Base::clear ( )

Clear the display buffer.

The entire contents of the screen buffer are cleared to BLACK.

See also

display(bool)

Definition at line 293 of file Arduboy2.cpp.

6.3.2.14 bool Arduboy2Base::collide (Point *point*, Rect *rect*)

Test if a point falls within a rectangle.

Parameters

point	A structure describing the location of the point.
rect	A structure describing the location and size of the rectangle.

Returns

`true` if the specified point is within the specified rectangle.

This function is intended to detemine if an object, whose boundaries are are defined by the given rectangle, is in contact with the given point.

See also

Point Rect

Definition at line 1042 of file Arduboy2.cpp.

6.3.2.15 bool Arduboy2Base::collide (Rect *rect1*, Rect *rect2*)

Test if a rectangle is intersecting with another rectangle.

Parameters

rect1,rect2	Structures describing the size and locations of the rectangles.

Returns

`true` if the first rectangle is intersecting the second.

This function is intended to detemine if an object, whose boundaries are are defined by the given rectangle, is in contact with another rectangular object.

See also

Rect

Definition at line 1048 of file Arduboy2.cpp.

6.3.2.16 int Arduboy2Base::cpuLoad ( )

Return the load on the CPU as a percentage.

Returns

The load on the CPU as a percentage of the total frame time.

The returned value gives the time spent processing a frame as a percentage the total time allotted for a frame, as determined by the frame rate.

This function normally wouldn't be used in the final program. It is intended for use during program development as an aid in helping with frame timing.

Note

The percentage returned can be higher than 100 if more time is spent processing a frame than the time allotted per frame. This would indicate that the frame rate should be made slower or the frame processing code should be optimized to run faster.

See also

setFrameRate() nextFrame()

Definition at line 273 of file Arduboy2.cpp.

6.3.2.17 void Arduboy2Core::delayShort (uint16_t *ms*) [static], [inherited]

Delay for the number of milliseconds, specified as a 16 bit value.

Parameters

ms	The delay in milliseconds.

This function works the same as the Arduino `delay()` function except the provided value is 16 bits long, so the maximum delay allowed is 65535 milliseconds (about 65.5 seconds). Using this function instead of Arduino `delay()` will save a few bytes of code.

Definition at line 557 of file Arduboy2Core.cpp.

6.3.2.18 void Arduboy2Core::digitalWriteRGB (uint8_t *red,* uint8_t *green,* uint8_t *blue*) [static], [inherited]

Set the RGB LEDs digitally, to either fully on or fully off.

Parameters

red,green,blue	Use value RGB_ON or RGB_OFF to set each LED.

The RGB LED is actually individual red, green and blue LEDs placed very close together in a single package. This 3 parameter version of the function will set each LED either on or off, to set the RGB LED to 7 different colors at their highest brightness or turn it off.

The colors are as follows:

```
RED_LED   GREEN_LED   BLUE_LED   COLOR
-------   ---------   --------   -----
RGB_OFF    RGB_OFF    RGB_OFF    OFF
RGB_OFF    RGB_OFF    RGB_ON     Blue
RGB_OFF    RGB_ON     RGB_OFF    Green
RGB_OFF    RGB_ON     RGB_ON     Cyan
RGB_ON     RGB_OFF    RGB_OFF    Red
RGB_ON     RGB_OFF    RGB_ON     Magenta
RGB_ON     RGB_ON     RGB_OFF    Yellow
RGB_ON     RGB_ON     RGB_ON     White
```

Note

Using the RGB LED in analog mode will prevent digital control of the LED. To restore the ability to control the LED digitally, use the `freeRGBled()` function.

Note

Many of the Kickstarter Arduboys were accidentally shipped with the RGB LED installed incorrectly. For these units, the green LED cannot be lit. As long as the green led is set to off, turning on the red LED will actually light the blue LED and turning on the blue LED will actually light the red LED. If the green LED is turned on, none of the LEDs will light.

See also

digitalWriteRGB(uint8_t, uint8_t) setRGBled() freeRGBled()

Definition at line 488 of file Arduboy2Core.cpp.

6.3.2.19 void Arduboy2Core::digitalWriteRGB (uint8_t *color,* uint8_t *val*) [static],[inherited]

Set one of the RGB LEDs digitally, to either fully on or fully off.

Parameters

color	The name of the LED to set. The value given should be one of RED_LED, GREEN_LED or BLUE_LED.
val	Indicates whether to turn the specified LED on or off. The value given should be RGB_ON or RGB_OFF.

This 2 parameter version of the function will set a single LED within the RGB LED either fully on or fully off. See the description of the 3 parameter version of this function for more details on the RGB LED.

See also

digitalWriteRGB(uint8_t, uint8_t, uint8_t) setRGBled() freeRGBled()

Definition at line 502 of file Arduboy2Core.cpp.

6.3.2.20 void Arduboy2Base::display ( )

Copy the contents of the display buffer to the display.

The contents of the display buffer in RAM are copied to the display and will appear on the screen.

See also

display(bool)

Definition at line 1001 of file Arduboy2.cpp.

6.3.2.21 void Arduboy2Base::display (bool *clear*)

Copy the contents of the display buffer to the display. The display buffer can optionally be cleared.

Parameters

clear	If `true` the display buffer will be cleared to zero. The defined value `CLEAR_BUFFER` should be used instead of `true` to make it more meaningful.

Operation is the same as calling `display()` without parameters except additionally the display buffer will be cleared if the parameter evaluates to `true`. (The defined value `CLEAR_BUFFER` can be used for this)

Using `display(CLEAR_BUFFER)` is faster and produces less code than calling `display()` followed by `clear()`.

See also

display() clear()

Definition at line 1006 of file Arduboy2.cpp.

6.3.2.22 void Arduboy2Core::displayOff ( ) [static], [inherited]

Turn the display off.

The display will clear and be put into a low power mode. This can be used to extend battery life when a game is paused or when a sketch doesn't require anything to be displayed for a relatively long period of time.

See also

displayOn()

Definition at line 283 of file Arduboy2Core.cpp.

6.3.2.23 void Arduboy2Core::displayOn ( ) [static],[inherited]

Turn the display on.

Used to power up and reinitialize the display after calling `displayOff()`.

Note

The previous call to `displayOff()` will have caused the display's buffer contents to be lost. The display will have to be re-painted, which is usually done by calling `display()`.

See also

displayOff()

Definition at line 294 of file Arduboy2Core.cpp.

6.3.2.24 void Arduboy2Base::drawBitmap (int16_t *x*, int16_t *y*, const uint8_t * *bitmap*, uint8_t *w*, uint8_t *h*, uint8_t *color* = WHITE) [static]

Draw a bitmap from an array in program memory.

Parameters

x	The X coordinate of the top left pixel affected by the bitmap.
y	The Y coordinate of the top left pixel affected by the bitmap.
bitmap	A pointer to the bitmap array in program memory.
w	The width of the bitmap in pixels.
h	The height of the bitmap in pixels.
color	The color of pixels for bits set to 1 in the bitmap. If the value is INVERT, bits set to 1 will invert the corresponding pixel. (optional; defaults to WHITE).

Bits set to 1 in the provided bitmap array will have their corresponding pixel set to the specified color. For bits set to 0 in the array, the corresponding pixel will be left unchanged.

Each byte in the array specifies a vertical column of 8 pixels, with the least significant bit at the top.

The array must be located in program memory by using the PROGMEM modifier.

Definition at line 809 of file Arduboy2.cpp.

6.3.2.25 void Arduboy2Base::drawCircle (int16_t *x0*, int16_t *y0*, uint8_t *r*, uint8_t *color* = WHITE)

Draw a circle of a given radius.

Parameters

x0	The X coordinate of the circle's center.
y0	The Y coordinate of the circle's center.
r	The radius of the circle in pixels.
color	The circle's color (optional; defaults to WHITE).

Definition at line 367 of file Arduboy2.cpp.

6.3.2.26 void Arduboy2Base::drawCompressed (int16_t *sx*, int16_t *sy*, const uint8_t * *bitmap*, uint8_t *color* = WHITE) [static]

Draw a bitmap from an array of compressed data.

Parameters

sx	The X coordinate of the top left pixel affected by the bitmap.
sy	The Y coordinate of the top left pixel affected by the bitmap.
bitmap	A pointer to the compressed bitmap array in program memory.
color	The color of pixels for bits set to 1 in the bitmap. (optional; defaults to WHITE).

Draw a bitmap starting at the given coordinates from an array that has been compressed using an algorthm implemented by Team A.R.G. For more information see: https://github.com/TEAMarg/drawCompressed https://github.com/TEAMarg/Cabi

Bits set to 1 in the provided bitmap array will have their corresponding pixel set to the specified color. For bits set to 0 in the array, the corresponding pixel will be left unchanged.

The array must be located in program memory by using the PROGMEM modifier.

Definition at line 906 of file Arduboy2.cpp.

6.3.2.27 void Arduboy2Base::drawFastHLine (int16_t *x*, int16_t *y*, uint8_t *w*, uint8_t *color* = WHITE)

Draw a horizontal line.

Parameters

x	The X coordinate of the left start point.
y	The Y coordinate of the left start point.
w	The width of the line.
color	The color of the line (optional; defaults to WHITE).

Definition at line 563 of file Arduboy2.cpp.

6.3.2.28 void Arduboy2Base::drawFastVLine (int16_t *x*, int16_t *y*, uint8_t *h*, uint8_t *color* = WHITE)

Draw a vertical line.

Parameters

x	The X coordinate of the upper start point.
y	The Y coordinate of the upper start point.
h	The height of the line.
color	The color of the line (optional; defaults to WHITE).

Definition at line 553 of file Arduboy2.cpp.

6.3.2.29 void Arduboy2Base::drawLine (int16_t *x0,* int16_t *y0,* int16_t *x1,* int16_t *y1,* uint8_t *color* = **WHITE**)

Draw a line between two specified points.

Parameters

x0,x1	The X coordinates of the line ends.
y0,y1	The Y coordinates of the line ends.
color	The line's color (optional; defaults to WHITE).

Draw a line from the start point to the end point using Bresenham's algorithm. The start and end points can be at any location with respect to the other.

Definition at line 493 of file Arduboy2.cpp.

6.3.2.30 void Arduboy2Base::drawPixel (int16_t *x,* int16_t *y,* uint8_t *color* = **WHITE**)

Set a single pixel in the display buffer to the specified color.

Parameters

x	The X coordinate of the pixel.
y	The Y coordinate of the pixel.
color	The color of the pixel (optional; defaults to WHITE).

The single pixel specified location in the display buffer is set to the specified color. The values WHITE or BLACK can be used for the color. If the `color` parameter isn't included, the pixel will be set to WHITE.

Definition at line 306 of file Arduboy2.cpp.

6.3.2.31 void Arduboy2Base::drawRect (int16_t *x,* int16_t *y,* uint8_t *w,* uint8_t *h,* uint8_t *color* = **WHITE**)

Draw a rectangle of a specified width and height.

Parameters

x	The X coordinate of the upper left corner.
y	The Y coordinate of the upper left corner.
w	The width of the rectangle.
h	The height of the rectangle.
color	The color of the pixel (optional; defaults to WHITE).

Definition at line 544 of file Arduboy2.cpp.

6.3.2.32 void Arduboy2Base::drawRoundRect (int16_t *x*, int16_t *y*, uint8_t *w*, uint8_t *h*, uint8_t *r*, uint8_t *color* = WHITE)

Draw a rectangle with rounded corners.

Parameters

x	The X coordinate of the left edge.
y	The Y coordinate of the top edge.
w	The width of the rectangle.
h	The height of the rectangle.
r	The radius of the semicircles forming the corners.
color	The color of the rectangle (optional; defaults to WHITE).

Definition at line 672 of file Arduboy2.cpp.

6.3.2.33 void Arduboy2Base::drawSlowXYBitmap (int16_t *x*, int16_t *y*, const uint8_t * *bitmap*, uint8_t *w*, uint8_t *h*, uint8_t *color* = WHITE)

Draw a bitmap from a horizontally oriented array in program memory.

Parameters

x	The X coordinate of the top left pixel affected by the bitmap.
y	The Y coordinate of the top left pixel affected by the bitmap.
bitmap	A pointer to the bitmap array in program memory.
w	The width of the bitmap in pixels.
h	The height of the bitmap in pixels.
color	The color of pixels for bits set to 1 in the bitmap. (optional; defaults to WHITE).

Bits set to 1 in the provided bitmap array will have their corresponding pixel set to the specified color. For bits set to 0 in the array, the corresponding pixel will be left unchanged.

Each byte in the array specifies a horizontal row of 8 pixels, with the most significant bit at the left end of the row.

The array must be located in program memory by using the PROGMEM modifier.

Note

This function requires a lot of additional CPU power and will draw images slower than `drawBitmap()`, which uses bitmaps that are stored in a format that allows them to be directly written to the screen. It is recommended you use `drawBitmap()` when possible.

Definition at line 855 of file Arduboy2.cpp.

6.3.2.34 void Arduboy2Base::drawTriangle (int16_t *x0*, int16_t *y0*, int16_t *x1*, int16_t *y1*, int16_t *x2*, int16_t *y2*, uint8_t *color* = WHITE)

Draw a triangle given the coordinates of each corner.

Parameters

x0,x1,x2	The X coordinates of the corners.
y0,y1,y2	The Y coordinates of the corners.
color	The triangle's color (optional; defaults to WHITE).

A triangle is drawn by specifying each of the three corner locations. The corners can be at any position with respect to the others.

Definition at line 698 of file Arduboy2.cpp.

6.3.2.35 bool Arduboy2Base::everyXFrames (uint8_t *frames*)

Indicate if the specified number of frames has elapsed.

Parameters

frames	The desired number of elapsed frames.

Returns

`true` if the specified number of frames has elapsed.

This function should be called with the same value each time for a given event. It will return `true` if the given number of frames has elapsed since the previous frame in which it returned `true`.

For example, if you wanted to fire a shot every 5 frames while the A button is being held down:

```
if (arduboy.everyXFrames(5)) {
  if arduboy.pressed(A_BUTTON) {
    fireShot();
  }
}
```

See also

setFrameRate() nextFrame()

Definition at line 227 of file Arduboy2.cpp.

6.3.2.36 void Arduboy2Core::exitToBootloader ( ) [static],[inherited]

Exit the sketch and start the bootloader.

The sketch will exit and the bootloader will be started in command mode. The effect will be similar to pressing the reset button.

This function is intended to be used to allow uploading a new sketch, when the USB code has been removed to gain more code space. Ideally, the sketch would present a "New Sketch Upload" menu or prompt telling the user to "Press and hold the DOWN button when the procedure to upload a new sketch has been initiated". The sketch would then wait for the DOWN button to be pressed and then call this function.

See also

ARDUBOY_NO_USB

Definition at line 562 of file Arduboy2Core.cpp.

6.3.2.37 void Arduboy2Base::fillCircle (int16_t *x0*, int16_t *y0*, uint8_t *r*, uint8_t *color* = WHITE)

Draw a filled-in circle of a given radius.

Parameters

x0	The X coordinate of the circle's center.
y0	The Y coordinate of the circle's center.
r	The radius of the circle in pixels.
color	The circle's color (optional; defaults to WHITE).

Definition at line 449 of file Arduboy2.cpp.

6.3.2.38 void Arduboy2Base::fillRect (int16_t *x*, int16_t *y*, uint8_t *w*, uint8_t *h*, uint8_t *color* = WHITE)

Draw a filled-in rectangle of a specified width and height.

Parameters

x	The X coordinate of the upper left corner.
y	The Y coordinate of the upper left corner.
w	The width of the rectangle.
h	The height of the rectangle.
color	The color of the pixel (optional; defaults to WHITE).

Definition at line 614 of file Arduboy2.cpp.

6.3.2.39 void Arduboy2Base::fillRoundRect (int16_t *x*, int16_t *y*, uint8_t *w*, uint8_t *h*, uint8_t *r*, uint8_t *color* = WHITE)

Draw a filled-in rectangle with rounded corners.

Parameters

x	The X coordinate of the left edge.
y	The Y coordinate of the top edge.
w	The width of the rectangle.
h	The height of the rectangle.
r	The radius of the semicircles forming the corners.
color	The color of the rectangle (optional; defaults to WHITE).

Definition at line 687 of file Arduboy2.cpp.

6.3.2.40 void Arduboy2Base::fillScreen (uint8_t *color* = WHITE)

Fill the screen buffer with the specified color.

Parameters

color	The fill color (optional; defaults to WHITE).

Definition at line 623 of file Arduboy2.cpp.

6.3.2.41 void Arduboy2Base::fillTriangle (int16_t *x0*, int16_t *y0*, int16_t *x1*, int16_t *y1*, int16_t *x2*, int16_t *y2*, uint8_t *color* = WHITE)

Draw a filled-in triangle given the coordinates of each corner.

Parameters

x0,x1,x2	The X coordinates of the corners.
y0,y1,y2	The Y coordinates of the corners.
color	The triangle's color (optional; defaults to WHITE).

A triangle is drawn by specifying each of the three corner locations. The corners can be at any position with respect to the others.

Definition at line 706 of file Arduboy2.cpp.

6.3.2.42 void Arduboy2Base::flashlight ( )

Turn the RGB LED and display fully on to act as a small flashlight/torch.

Checks if the UP button is pressed and if so turns the RGB LED and all display pixels fully on. If the UP button is detected, this function does not exit. The Arduboy must be restarted after flashlight mode is used.

This function is called by `begin()` and can be called by a sketch after `boot()`.

Note

This function also contains code to address a problem with uploading a new sketch, for sketches that interfere with the bootloader "magic number". This problem occurs with certain sketches that use large amounts of RAM. Being in flashlight mode when uploading a new sketch can fix this problem.
Therefore, for sketches that potentially could cause this problem, and use `boot()` instead of `begin()`, it is recommended that a call to `flashlight()` be included after calling `boot()`. If program space is limited, `safeMode()` can be used instead of `flashlight()`.

See also

begin() boot() safeMode()

Definition at line 55 of file Arduboy2.cpp.

6.3.2.43 void Arduboy2Core::flipHorizontal (bool *flipped*) [static], [inherited]

Flip the display horizontally or set it back to normal.

Parameters

flipped	`true` will set horizontal flip mode. `false` will set normal horizontal orientation.

Calling this function with a value of `true` will cause the X coordinate to start at the left edge of the display instead of the right, effectively flipping the display horizontally.

Once in horizontal flip mode, it will remain this way until normal horizontal mode is set by calling this function with a value of `false`.

See also

flipVertical()

Definition at line 426 of file Arduboy2Core.cpp.

6.3.2.44 void Arduboy2Core::flipVertical (bool *flipped*) [static], [inherited]

Flip the display vertically or set it back to normal.

Parameters

flipped	`true` will set vertical flip mode. `false` will set normal vertical orientation.

Calling this function with a value of `true` will cause the Y coordinate to start at the bottom edge of the display instead of the top, effectively flipping the display vertically.

Once in vertical flip mode, it will remain this way until normal vertical mode is set by calling this function with a value of `false`.

See also

flipHorizontal()

Definition at line 420 of file Arduboy2Core.cpp.

6.3.2.45 void Arduboy2Core::freeRGBled ( ) [static], [inherited]

Relinquish analog control of the RGB LED.

Using the RGB LED in analog mode prevents further use of the LED in digital mode. This function will restore the pins used for the LED, so it can be used in digital mode.

See also

digitalWriteRGB() setRGBled()

Definition at line 479 of file Arduboy2Core.cpp.

6.3.2.46 uint8_t * Arduboy2Base::getBuffer ( )

Get a pointer to the display buffer in RAM.

Returns

A pointer to the display buffer array in RAM.

The location of the display buffer in RAM, which is displayed using `display()`, can be gotten using this function. The buffer can then be read and directly manipulated.

Note

The display buffer array, `sBuffer`, is public. A sketch can access it directly. Doing so may be more efficient than accessing it via the pointer returned by `getBuffer()`.

See also

sBuffer

Definition at line 1011 of file Arduboy2.cpp.

6.3.2.47 uint8_t Arduboy2Base::getPixel (uint8_t *x*, uint8_t *y*)

Returns the state of the given pixel in the screen buffer.

Parameters

x	The X coordinate of the pixel.
y	The Y coordinate of the pixel.

Returns

WHITE if the pixel is on or BLACK if the pixel is off.

Definition at line 360 of file Arduboy2.cpp.

6.3.2.48 uint8_t Arduboy2Core::height ( ) [static], [inherited]

Get the height of the display in pixels.

Returns

The height of the display in pixels.

Note

In most cases, the defined value `HEIGHT` would be better to use instead of this function.

Definition at line 301 of file Arduboy2Core.cpp.

6.3.2.49 void Arduboy2Core::idle ( ) [static],[inherited]

Idle the CPU to save power.

This puts the CPU in *idle* sleep mode. You should call this as often as you can for the best power savings. The timer 0 overflow interrupt will wake up the chip every 1ms, so even at 60 FPS a well written app should be able to sleep maybe half the time in between rendering it's own frames.

Definition at line 266 of file Arduboy2Core.cpp.

6.3.2.50 void Arduboy2Base::initRandomSeed ( )

Seed the random number generator with a random value.

The Arduino random number generator is seeded with a random value derived from entropy from an ADC reading of a floating pin combined with the microseconds since boot.

This method is still most effective when called after a semi-random time, such as after a user hits a button to start a game or other semi-random event.

Definition at line 278 of file Arduboy2.cpp.

6.3.2.51 void Arduboy2Core::invert (bool *inverse*) [static],[inherited]

Invert the entire display or set it back to normal.

Parameters

inverse	`true` will invert the display. `false` will set the display to no-inverted.

Calling this function with a value of `true` will set the display to inverted mode. A pixel with a value of 0 will be on and a pixel set to 1 will be off.

Once in inverted mode, the display will remain this way until it is set back to non-inverted mode by calling this function with `false`.

Definition at line 407 of file Arduboy2Core.cpp.

6.3.2.52 bool Arduboy2Base::justPressed (uint8_t *button*)

Check if a button has just been pressed.

Parameters

button	The button to test for. Only one button should be specified.

Returns

`true` if the specified button has just been pressed.

Return `true` if the given button was pressed between the latest call to `pollButtons()` and previous call to `pollButtons()`. If the button has been held down over multiple polls, this function will return `false`.

There is no need to check for the release of the button since it must have been released for this function to return `true` when pressed again.

This function should only be used to test a single button.

See also

pollButtons() justReleased()

Definition at line 1032 of file Arduboy2.cpp.

6.3.2.53 bool Arduboy2Base::justReleased (uint8_t *button*)

Check if a button has just been released.

Parameters

button	The button to test for. Only one button should be specified.

Returns

`true` if the specified button has just been released.

Return `true` if the given button, having previously been pressed, was released between the latest call to `pollButtons()` and previous call to `pollButtons()`. If the button has remained released over multiple polls, this function will return `false`.

There is no need to check for the button having been pressed since it must have been previously pressed for this function to return `true` upon release.

This function should only be used to test a single button.

Note

There aren't many cases where this function would be needed. Wanting to know if a button has been released, without knowing when it was pressed, is uncommon.

See also

pollButtons() justPressed()

Definition at line 1037 of file Arduboy2.cpp.

6.3.2.54 void Arduboy2Core::LCDCommandMode ( ) [static], [inherited]

Put the display into command mode.

When placed in command mode, data that is sent to the display will be treated as commands.

See the SSD1306 controller and OLED display documents for available commands and command sequences.

Links:

- https://www.adafruit.com/datasheets/SSD1306.pdf
- http://www.buydisplay.com/download/manual/ER-OLED013-1_Series_Datasheet.pdf

Note

This is a low level function that is not intended for general use in a sketch. It has been made public and documented for use by derived classes.

See also

LCDDataMode() sendLCDCommand() SPItransfer()

Definition at line 220 of file Arduboy2Core.cpp.

6.3.2.55 void Arduboy2Core::LCDDataMode ( ) [static], [inherited]

Put the display into data mode.

When placed in data mode, data that is sent to the display will be considered as data to be displayed.

Note

This is a low level function that is not intended for general use in a sketch. It has been made public and documented for use by derived classes.

See also

LCDCommandMode() SPItransfer()

Definition at line 215 of file Arduboy2Core.cpp.

6.3.2.56 bool Arduboy2Base::nextFrame ( )

Indicate that it's time to render the next frame.

Returns

`true` if it's time for the next frame.

When this function returns `true`, the amount of time has elapsed to display the next frame, as specified by `setFrameRate()`.

This function will normally be called at the start of the rendering loop which would wait for `true` to be returned before rendering and displaying the next frame.

example:

```
void loop() {
  if (!arduboy.nextFrame()) {
    return; // go back to the start of the loop
  }
  // render and display the next frame
}
```

See also

setFrameRate() setFrameDuration() nextFrameDEV()

Definition at line 232 of file Arduboy2.cpp.

6.3.2.57 bool Arduboy2Base::nextFrameDEV ( )

Indicate that it's time to render the next frame, and visually indicate if the code is running slower than the desired frame rate. **FOR USE DURING DEVELOPMENT**

Returns

`true` if it's time for the next frame.

This function is intended to be used in place of `nextFrame()` during the development of a sketch. It does the same thing as `nextFrame()` but additionally will light the yellow TX LED (at the bottom, to the left of the USB connector) whenever a frame takes longer to generate than the time allotted per frame, as determined by the `setFrameRate()` function.

Therefore, whenever the TX LED comes on (while not communicating over USB), it indicates that the sketch is running slower than the desired rate set by `setFrameRate()`. In this case the developer may wish to set a slower frame rate, or reduce or optimize the code for such frames.

Note

Once a sketch is ready for release, it would be expected that `nextFrameDEV()` calls be restored to `nextFrame()`.

See also

nextFrame() cpuLoad() setFrameRate()

Definition at line 260 of file Arduboy2.cpp.

6.3.2.58 bool Arduboy2Base::notPressed (uint8_t *buttons*)

Test if the specified buttons are not pressed.

Parameters

buttons	A bit mask indicating which buttons to test. (Can be a single button)

Returns

`true` if *all* buttons in the provided mask are currently released.

Read the state of the buttons and return `true` if all the buttons in the specified mask are currently released.

Example: `if (notPressed(UP_BUTTON))`

Note

This function does not perform any button debouncing.

Definition at line 1021 of file Arduboy2.cpp.

6.3.2.59 void Arduboy2Core::paint8Pixels (uint8_t *pixels*) [static],[inherited]

Paint 8 pixels vertically to the display.

Parameters

pixels	A byte whose bits specify a vertical column of 8 pixels.

A byte representing a vertical column of 8 pixels is written to the display at the current page and column address. The address is then incremented. The page/column address will wrap to the start of the display (the top left) when it increments past the end (lower right).

The least significant bit represents the top pixel in the column. A bit set to 1 is lit, 0 is unlit.

Example:

```
X = lit pixels, . = unlit pixels

blank()                          paint8Pixels() 0xFF, 0, 0xF0, 0, 0x0F
v TOP LEFT corner (8x9)          v TOP LEFT corner
. . . . . . . . (page 1)         X . . . X . . . (page 1)
. . . . . . . .                  X . . . X . . .
. . . . . . . .                  X . . . X . . .
. . . . . . . .                  X . . . X . . .
. . . . . . . .                  X . X . . . . .
. . . . . . . .                  X . X . . . . .
. . . . . . . .                  X . X . . . . .
. . . . . . . . (end of page 1)  X . X . . . . . (end of page 1)
. . . . . . . . (page 2)         . . . . . . . . (page 2)
```

Definition at line 306 of file Arduboy2Core.cpp.

6.3.2.60 void Arduboy2Core::paintScreen (const uint8_t * *image*) [static],[inherited]

Paints an entire image directly to the display from program memory.

Parameters

image	A byte array in program memory representing the entire contents of the display.

The contents of the specified array in program memory is written to the display. Each byte in the array represents a vertical column of 8 pixels with the least significant bit at the top. The bytes are written starting at the top left, progressing horizontally and wrapping at the end of each row, to the bottom right. The size of the array must exactly match the number of pixels in the entire display.

See also

paint8Pixels()

Definition at line 311 of file Arduboy2Core.cpp.

6.3.2.61 void Arduboy2Core::paintScreen (uint8_t *image[ ],* bool *clear =* `false`) [static], [inherited]

Paints an entire image directly to the display from an array in RAM.

Parameters

image	A byte array in RAM representing the entire contents of the display.
clear	If `true` the array in RAM will be cleared to zeros upon return from this function. If `false` the RAM buffer will remain unchanged. (optional; defaults to `false`)

The contents of the specified array in RAM is written to the display. Each byte in the array represents a vertical column of 8 pixels with the least significant bit at the top. The bytes are written starting at the top left, progressing horizontally and wrapping at the end of each row, to the bottom right. The size of the array must exactly match the number of pixels in the entire display.

If parameter `clear` is set to `true` the RAM array will be cleared to zeros after its contents are written to the display.

See also

paint8Pixels()

Definition at line 325 of file Arduboy2Core.cpp.

6.3.2.62 void Arduboy2Base::pollButtons ( )

Poll the buttons and track their state over time.

Read and save the current state of the buttons and also keep track of the button state when this function was previouly called. These states are used by the `justPressed()` and `justReleased()` functions to determine if a button has changed state between now and the previous call to `pollButtons()`.

This function should be called once at the start of each new frame.

The `justPressed()` and `justReleased()` functions rely on this function.

example:

```
void loop() {
  if (!arduboy.nextFrame()) {
    return;
  }
  arduboy.pollButtons();

  // use justPressed() as necessary to determine if a button was just pressed
```

Note

As long as the elapsed time between calls to this function is long enough, buttons will be naturally debounced. Calling it once per frame at a frame rate of 60 or lower (or possibly somewhat higher), should be sufficient.

See also

justPressed() justReleased()

Definition at line 1026 of file Arduboy2.cpp.

6.3.2.63 bool Arduboy2Base::pressed (uint8_t *buttons*)

Test if the specified buttons are pressed.

Parameters

buttons	A bit mask indicating which buttons to test. (Can be a single button)

Returns

`true` if *all* buttons in the provided mask are currently pressed.

Read the state of the buttons and return `true` if all the buttons in the specified mask are being pressed.

Example: `if (pressed(LEFT_BUTTON + A_BUTTON))`

Note

This function does not perform any button debouncing.

Definition at line 1016 of file Arduboy2.cpp.

6.3.2.64 bool Arduboy2Base::readShowBootLogoFlag ( )

Read the "Show Boot Logo" flag in system EEPROM.

Returns

`true` if the flag is set to indicate that the boot logo sequence should be displayed. `false` if the flag is set to not display the boot logo sequence.

The "Show Boot Logo" flag is used to determine whether the system boot logo sequence is to be displayed when the system boots up. This function returns the value of this flag.

See also

writeShowBootLogoFlag() bootLogo()

Definition at line 1104 of file Arduboy2.cpp.

6.3.2.65 bool Arduboy2Base::readShowBootLogoLEDsFlag ( )

Read the "Show LEDs with boot logo" flag in system EEPROM.

Returns

`true` if the flag is set to indicate that the RGB LEDs should be flashed. `false` if the flag is set to leave the LEDs off.

The "Show LEDs with boot logo" flag is used to determine whether the RGB LEDs should be flashed in sequence while the boot logo is being displayed. This function returns the value of this flag.

See also

writeShowBootLogoLEDsFlag()

Definition at line 1130 of file Arduboy2.cpp.

6.3.2.66 bool Arduboy2Base::readShowUnitNameFlag ( )

Read the "Show Unit Name" flag in system EEPROM.

Returns

`true` if the flag is set to indicate that the unit name should be displayed. `false` if the flag is set to not display the unit name.

The "Show Unit Name" flag is used to determine whether the system unit name is to be displayed at the end of the boot logo sequence. This function returns the value of this flag.

See also

writeShowUnitNameFlag() writeUnitName() readUnitName() Arduboy2::bootLogoExtra()

Definition at line 1117 of file Arduboy2.cpp.

6.3.2.67 uint16_t Arduboy2Base::readUnitID ( )

Read the unit ID from system EEPROM.

Returns

The value of the unit ID stored in system EEPROM.

This function reads the unit ID that has been set in system EEPROM. The ID can be any value. It is intended to allow different units to be uniquely identified.

See also

writeUnitID() readUnitName()

Definition at line 1056 of file Arduboy2.cpp.

6.3.2.68 uint8_t Arduboy2Base::readUnitName (char * *name*)

Read the unit name from system EEPROM.

Parameters

name	A pointer to a string array variable where the unit name will be placed. The string will be up to 6 characters and terminated with a null (0x00) character, so the provided array must be at least 7 bytes long.

Returns

The length of the string (0-6).

This function reads the unit name that has been set in system EEPROM. The name is in ASCII and can contain any values except 0xFF and the null (0x00) terminator value.

The name can be used for any purpose. It could identify the owner or give the unit itself a nickname. A sketch could use it to automatically fill in a name or initials in a high score table, or display it as the "player" when the opponent is the computer.

Note

Sketches can use the defined value `ARDUBOY_UNIT_NAME_LEN` instead of hard coding a 6 when working with the unit name. For example, to allocate a buffer and read the unit name into it:

```
// Buffer for maximum name length plus the terminator
char unitName[ARDUBOY_UNIT_NAME_LEN + 1];

// The actual name length
byte unitNameLength;

unitNameLength = arduboy.readUnitName(unitName);
```

See also

writeUnitName() readUnitID() Arduboy2::bootLogoExtra()

Definition at line 1068 of file Arduboy2.cpp.

6.3.2.69 void Arduboy2Core::safeMode ( ) [static], [inherited]

Allow upload when the bootloader "magic number" could be corrupted.

If the UP button is held when this function is entered, the RGB LED will be lit and timer 0 will be disabled, then the sketch will remain in a tight loop. This is to address a problem with uploading a new sketch, for sketches that interfere with the bootloader "magic number". The problem occurs with certain sketches that use large amounts of RAM.

This function should be called after `boot()` in sketches that potentially could cause the problem.

It is intended to replace the `flashlight()` function when more program space is required. If possible, it is more desirable to use `flashlight()`, so that the actual flashlight feature isn't lost.

See also

Arduboy2Base::flashlight() boot()

Definition at line 247 of file Arduboy2Core.cpp.

6.3.2.70 void Arduboy2Core::sendLCDCommand (uint8_t *command*) [static], [inherited]

Send a single command byte to the display.

Parameters

command	The command byte to send to the display.

The display will be set to command mode then the specified command byte will be sent. The display will then be set to data mode. Multi-byte commands can be sent by calling this function multiple times.

Note

Sending improper commands to the display can place it into invalid or unexpected states, possibly even causing physical damage.

Definition at line 398 of file Arduboy2Core.cpp.

6.3.2.71 void Arduboy2Base::setFrameDuration (uint8_t *duration*)

Set the frame rate, used by the frame control functions, by giving the duration of each frame.

Parameters

duration	The desired duration of each frame in milliseconds.

Set the frame rate by specifying the duration of each frame in milliseconds. This is used by `nextFrame()` to update frames at a given rate. If this function or `setFrameRate()` isn't used, the default will be 16ms per frame.

Normally, the frame rate would be set to the desired value once, at the start of the game, but it can be changed at any time to alter the frame update rate.

See also

nextFrame() setFrameRate()

Definition at line 222 of file Arduboy2.cpp.

6.3.2.72 void Arduboy2Base::setFrameRate (uint8_t *rate*)

Set the frame rate used by the frame control functions.

Parameters

rate	The desired frame rate in frames per second.

Set the frame rate, in frames per second, used by `nextFrame()` to update frames at a given rate. If this function or `setFrameDuration()` isn't used, the default rate will be 60 (actually 62.5, see note below).

Normally, the frame rate would be set to the desired value once, at the start of the game, but it can be changed at any time to alter the frame update rate.

Note

The given rate is internally converted to a frame duration in milliseconds, rounded down to the nearest integer. Therefore, the actual rate will be equal to or higher than the rate given.
For example, 60 FPS would be 16.67ms per frame. This will be rounded down to 16ms, giving an actual frame rate of 62.5 FPS.

See also

nextFrame() setFrameDuration()

Definition at line 217 of file Arduboy2.cpp.

6.3.2.73 void Arduboy2Core::setRGBled (uint8_t *red,* uint8_t *green,* uint8_t *blue*) [static], [inherited]

Set the light output of the RGB LED.

Parameters

red,green,blue	The brightness value for each LED.

The RGB LED is actually individual red, green and blue LEDs placed very close together in a single package. By setting the brightness of each LED, the RGB LED can show various colors and intensities. The brightness of each LED can be set to a value from 0 (fully off) to 255 (fully on).

Note

Certain libraries that take control of the hardware timers may interfere with the ability of this function to properly control the RGB LED. *ArduboyPlaytune* is one such library known to do this. The digitalWriteRGB() function will still work properly in this case.

Note

Many of the Kickstarter Arduboys were accidentally shipped with the RGB LED installed incorrectly. For these units, the green LED cannot be lit. As long as the green led is set to off, setting the red LED will actually control the blue LED and setting the blue LED will actually control the red LED. If the green LED is turned fully on, none of the LEDs will light.

See also

setRGBled(uint8_t, uint8_t) digitalWriteRGB() freeRGBled()

Definition at line 433 of file Arduboy2Core.cpp.

6.3.2.74 void Arduboy2Core::setRGBled (uint8_t *color,* uint8_t *val*) [static], [inherited]

Set the brightness of one of the RGB LEDs without affecting the others.

Parameters

color	The name of the LED to set. The value given should be one of RED_LED, GREEN_LED or BLUE_LED.
val	The brightness value for the LED, from 0 to 255.

Note

In order to use this function, the 3 parameter version must first be called at least once, in order to initialize the hardware.

This 2 parameter version of the function will set the brightness of a single LED within the RGB LED without affecting the current brightness of the other two. See the description of the 3 parameter version of this function for more details on the RGB LED.

See also

setRGBled(uint8_t, uint8_t, uint8_t) digitalWriteRGB() freeRGBled()

Definition at line 455 of file Arduboy2Core.cpp.

6.3.2.75 void Arduboy2Core::SPItransfer (uint8_t *data*) [static], [inherited]

Transfer a byte to the display.

Parameters

data	The byte to be sent to the display.

Transfer one byte to the display over the SPI port and wait for the transfer to complete. The byte will either be interpreted as a command or as data to be placed on the screen, depending on the command/data mode.

See also

LCDDataMode() LCDCommandMode() sendLCDCommand()

Definition at line 234 of file Arduboy2Core.cpp.

6.3.2.76 void Arduboy2Base::systemButtons ( )

Handle buttons held on startup for system control.

This function is called by `begin()` and can be called by a sketch after `boot()`.

Hold the B button when booting to enter system control mode. The B button must be held continuously to remain in this mode. Then, pressing other buttons will perform system control functions:

- UP: Set "sound enabled" in EEPROM
- DOWN: Set "sound disabled" (mute) in EEPROM

See also

begin() boot()

Definition at line 76 of file Arduboy2.cpp.

6.3.2.77 void Arduboy2Base::waitNoButtons ( )

Wait until all buttons have been released.

This function is called by `begin()` and can be called by a sketch after `boot()`.

It won't return unless no buttons are being pressed. A short delay is performed each time before testing the state of the buttons to do a simple button debounce.

This function is called at the end of `begin()` to make sure no buttons used to perform system start up actions are still being pressed, to prevent them from erroneously being detected by the sketch code itself.

See also

begin() boot()

Definition at line 209 of file Arduboy2.cpp.

6.3.2.78 uint8_t Arduboy2Core::width ( ) [static], [inherited]

Get the width of the display in pixels.

Returns

The width of the display in pixels.

Note

In most cases, the defined value `WIDTH` would be better to use instead of this function.

Definition at line 299 of file Arduboy2Core.cpp.

6.3.2.79 void Arduboy2Base::writeShowBootLogoFlag (bool *val*)

Write the "Show Boot Logo" flag in system EEPROM.

Parameters

val	If `true` the flag is set to indicate that the boot logo sequence should be displayed. If `false` the flag is set to not display the boot logo sequence.

The "Show Boot Logo" flag is used to determine whether the system boot logo sequence is to be displayed when the system boots up. This function allows the flag to be saved with the desired value.

See also

readShowBootLogoFlag() bootLogo()

Definition at line 1109 of file Arduboy2.cpp.

6.3.2.80 void Arduboy2Base::writeShowBootLogoLEDsFlag (bool *val*)

Write the "Show LEDs with boot logo" flag in system EEPROM.

Parameters

val	If `true` the flag is set to indicate that the RGB LEDs should be flashed. If `false` the flag is set to leave the LEDs off.

The "Show LEDs with boot logo" flag is used to determine whether the RGB LEDs should be flashed in sequence while the boot logo is being displayed. This function allows the flag to be saved with the desired value.

See also

readShowBootLogoLEDsFlag()

Definition at line 1135 of file Arduboy2.cpp.

6.3.2.81 void Arduboy2Base::writeShowUnitNameFlag (bool *val*)

Write the "Show Unit Name" flag in system EEPROM.

Parameters

val	If `true` the flag is set to indicate that the unit name should be displayed. If `false` the flag is set to not display the unit name.

The "Show Unit Name" flag is used to determine whether the system unit name is to be displayed at the end of the boot logo sequence. This function allows the flag to be saved with the desired value.

See also

readShowUnitNameFlag() writeUnitName() readUnitName() Arduboy2::bootLogoExtra()

Definition at line 1122 of file Arduboy2.cpp.

6.3.2.82 void Arduboy2Base::writeUnitID (uint16_t *id*)

Write a unit ID to system EEPROM.

Parameters

id	The value of the unit ID to be stored in system EEPROM.

This function writes a unit ID to a reserved location in system EEPROM. The ID can be any value. It is intended to allow different units to be uniquely identified.

See also

readUnitID() writeUnitName()

Definition at line 1062 of file Arduboy2.cpp.

6.3.2.83 void Arduboy2Base::writeUnitName (char * *name*)

Write a unit name to system EEPROM.

Parameters

name	A pointer to a string array variable containing the unit name to be saved. The string can be up to 6 characters and must be terminated with a null (0x00) character. It can contain any values except 0xFF.

This function writes a unit name to a reserved area in system EEPROM. The name is in ASCII and can contain any values except 0xFF and the null (0x00) terminator value. The newline character (LF, \n, 0x0A) and carriage return character (CR, \r, 0x0D) should also be avoided.

The name can be used for any purpose. It could identify the owner or give the unit itself a nickname. A sketch could use it to automatically fill in a name or initials in a high score table, or display it as the "player" when the opponent is the computer.

Note

Sketches can use the defined value `ARDUBOY_UNIT_NAME_LEN` instead of hard coding a 6 when working with the unit name.

See also

readUnitName() writeUnitID() Arduboy2::bootLogoExtra()

Definition at line 1088 of file Arduboy2.cpp.

6.3.3 Member Data Documentation

6.3.3.1 Arduboy2Audio Arduboy2Base::audio

An object created to provide audio control functions within this class.

This object is created to eliminate the need for a sketch to create an Arduboy2Audio class object itself.

See also

Arduboy2Audio

Definition at line 182 of file Arduboy2.h.

6.3.3.2 uint16_t Arduboy2Base::frameCount

A counter which is incremented once per frame.

This counter is incremented once per frame when using the `nextFrame()` function. It will wrap to zero when it reaches its maximum value.

It could be used to have an event occur for a given number of frames, or a given number of frames later, in a way that wouldn't be quantized the way that using `everyXFrames()` might.

example:

```
// move for 10 frames when right button is pressed, if not already moving
if (!moving) {
  if (arduboy.justPressed(RIGHT_BUTTON)) {
    endMoving = arduboy.frameCount + 10;
    moving = true;
  }
} else {
  movePlayer();
  if (arduboy.frameCount == endMoving) {
    moving = false;
  }
}
```

This counter could also be used to determine the number of frames that have elapsed between events but the possibility of the counter wrapping would have to be accounted for.

See also

nextFrame() everyXFrames()

Definition at line 1240 of file Arduboy2.h.

6.3.3.3 uint8_t Arduboy2Base::sBuffer [static]

The display buffer array in RAM.

The display buffer (also known as the screen buffer) contains an image bitmap of the desired contents of the display, which is written to the display using the `display()` function. The drawing functions of this library manipulate the contents of the display buffer. A sketch can also access the display buffer directly.

See also

getBuffer()

Definition at line 1254 of file Arduboy2.h.

The documentation for this class was generated from the following files:

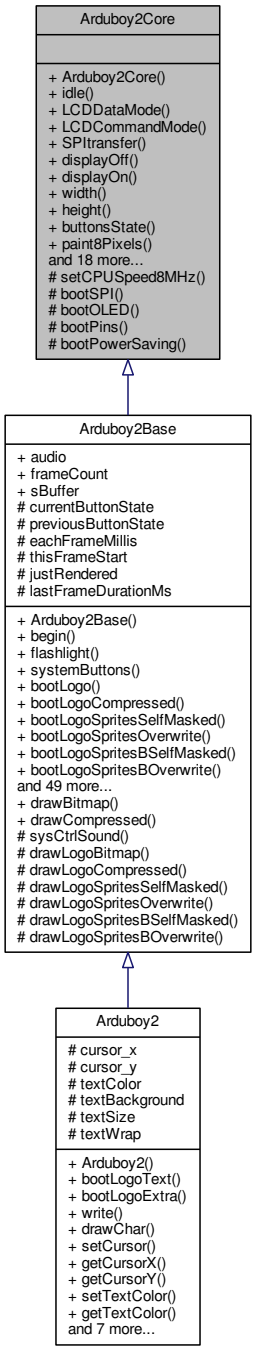
- src/Arduboy2.h
- src/Arduboy2.cpp

6.4 Arduboy2Core Class Reference

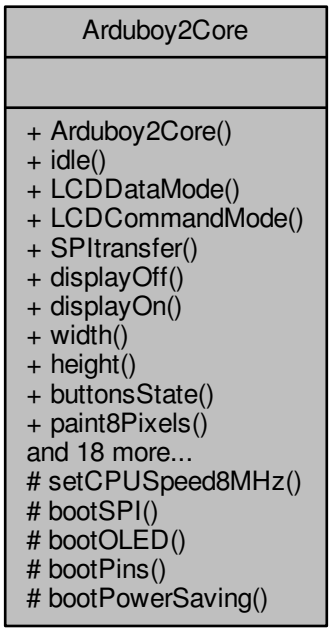
Lower level functions generally dealing directly with the hardware.

```
#include <Arduboy2Core.h>
```

Inheritance diagram for Arduboy2Core:



Collaboration diagram for Arduboy2Core:



Static Public Member Functions

- static void idle ()

  Idle the CPU to save power.
- static void LCDDataMode ()

  Put the display into data mode.
- static void LCDCommandMode ()

  Put the display into command mode.
- static void SPItransfer (uint8_t data)

  Transfer a byte to the display.
- static void displayOff ()

  Turn the display off.
- static void displayOn ()

  Turn the display on.
- static uint8_t width ()

  Get the width of the display in pixels.
- static uint8_t height ()

  Get the height of the display in pixels.
- static uint8_t buttonsState ()

  Get the current state of all buttons as a bitmask.
- static void paint8Pixels (uint8_t pixels)

  Paint 8 pixels vertically to the display.
- static void paintScreen (const uint8_t *image)

Paints an entire image directly to the display from program memory.

- static void paintScreen (uint8_t image[ ], bool clear=false)

  Paints an entire image directly to the display from an array in RAM.
- static void blank ()

  Blank the display screen by setting all pixels off.
- static void invert (bool inverse)

  Invert the entire display or set it back to normal.
- static void allPixelsOn (bool on)

  Turn all display pixels on or display the buffer contents.
- static void flipVertical (bool flipped)

  Flip the display vertically or set it back to normal.
- static void flipHorizontal (bool flipped)

  Flip the display horizontally or set it back to normal.
- static void sendLCDCommand (uint8_t command)

  Send a single command byte to the display.
- static void setRGBled (uint8_t red, uint8_t green, uint8_t blue)

  Set the light output of the RGB LED.
- static void setRGBled (uint8_t color, uint8_t val)

  Set the brightness of one of the RGB LEDs without affecting the others.
- static void freeRGBled ()

  Relinquish analog control of the RGB LED.
- static void digitalWriteRGB (uint8_t red, uint8_t green, uint8_t blue)

  Set the RGB LEDs digitally, to either fully on or fully off.
- static void digitalWriteRGB (uint8_t color, uint8_t val)

  Set one of the RGB LEDs digitally, to either fully on or fully off.
- static void boot ()

  Initialize the Arduboy's hardware.
- static void safeMode ()

  Allow upload when the bootloader "magic number" could be corrupted.
- static void delayShort (uint16_t ms) __attribute__((noinline))

  Delay for the number of milliseconds, specified as a 16 bit value.
- static void exitToBootloader ()

  Exit the sketch and start the bootloader.

6.4.1 Detailed Description

Lower level functions generally dealing directly with the hardware.

This class is inherited by Arduboy2Base and thus also Arduboy2, so wouldn't normally be used directly by a sketch.

Note

A friend class named *Arduboy2Ex* is declared by this class. The intention is to allow a sketch to create an *Arduboy2Ex* class which would have access to the private and protected members of the Arduboy2Core class. It is hoped that this may eliminate the need to create an entire local copy of the library, in order to extend the functionality, in most circumstances.

Definition at line 333 of file Arduboy2Core.h.

6.4.2 Member Function Documentation

6.4.2.1 void Arduboy2Core::allPixelsOn (bool *on*) [static]

Turn all display pixels on or display the buffer contents.

Parameters

on	`true` turns all pixels on. `false` displays the contents of the hardware display buffer.

Calling this function with a value of `true` will override the contents of the hardware display buffer and turn all pixels on. The contents of the hardware buffer will remain unchanged.

Calling this function with a value of `false` will set the normal state of displaying the contents of the hardware display buffer.

Note

All pixels will be lit even if the display is in inverted mode.

See also

invert()

Definition at line 414 of file Arduboy2Core.cpp.

6.4.2.2 void Arduboy2Core::blank ( ) [static]

Blank the display screen by setting all pixels off.

All pixels on the screen will be written with a value of 0 to turn them off.

Definition at line 392 of file Arduboy2Core.cpp.

6.4.2.3 void Arduboy2Core::boot ( ) [static]

Initialize the Arduboy's hardware.

This function initializes the display, buttons, etc.

This function is called by begin() so isn't normally called within a sketch. However, in order to free up some code space, by eliminating some of the start up features, it can be called in place of begin(). The functions that begin() would call after boot() can then be called to add back in some of the start up features, if desired. See the README file or documentation on the main page for more details.

See also

Arduboy2Base::begin()

Definition at line 76 of file Arduboy2Core.cpp.

6.4.2.4 uint8_t Arduboy2Core::buttonsState ( ) [static]

Get the current state of all buttons as a bitmask.

Returns

A bitmask of the state of all the buttons.

The returned mask contains a bit for each button. For any pressed button, its bit will be 1. For released buttons their associated bits will be 0.

The following defined mask values should be used for the buttons:

LEFT_BUTTON, RIGHT_BUTTON, UP_BUTTON, DOWN_BUTTON, A_BUTTON, B_BUTTON

Definition at line 528 of file Arduboy2Core.cpp.

6.4.2.5 void Arduboy2Core::delayShort (uint16_t *ms*) [static]

Delay for the number of milliseconds, specified as a 16 bit value.

Parameters

ms	The delay in milliseconds.

This function works the same as the Arduino `delay()` function except the provided value is 16 bits long, so the maximum delay allowed is 65535 milliseconds (about 65.5 seconds). Using this function instead of Arduino `delay()` will save a few bytes of code.

Definition at line 557 of file Arduboy2Core.cpp.

6.4.2.6 void Arduboy2Core::digitalWriteRGB (uint8_t *red,* uint8_t *green,* uint8_t *blue*) [static]

Set the RGB LEDs digitally, to either fully on or fully off.

Parameters

red,green,blue	Use value RGB_ON or RGB_OFF to set each LED.

The RGB LED is actually individual red, green and blue LEDs placed very close together in a single package. This 3 parameter version of the function will set each LED either on or off, to set the RGB LED to 7 different colors at their highest brightness or turn it off.

The colors are as follows:

```
RED_LED   GREEN_LED   BLUE_LED   COLOR
-------   ---------   --------   -----
RGB_OFF    RGB_OFF    RGB_OFF    OFF
RGB_OFF    RGB_OFF    RGB_ON     Blue
RGB_OFF    RGB_ON     RGB_OFF    Green
RGB_OFF    RGB_ON     RGB_ON     Cyan
RGB_ON     RGB_OFF    RGB_OFF    Red
RGB_ON     RGB_OFF    RGB_ON     Magenta
RGB_ON     RGB_ON     RGB_OFF    Yellow
RGB_ON     RGB_ON     RGB_ON     White
```

Note

Using the RGB LED in analog mode will prevent digital control of the LED. To restore the ability to control the LED digitally, use the `freeRGBled()` function.

Note

Many of the Kickstarter Arduboys were accidentally shipped with the RGB LED installed incorrectly. For these units, the green LED cannot be lit. As long as the green led is set to off, turning on the red LED will actually light the blue LED and turning on the blue LED will actually light the red LED. If the green LED is turned on, none of the LEDs will light.

See also

digitalWriteRGB(uint8_t, uint8_t) setRGBled() freeRGBled()

Definition at line 488 of file Arduboy2Core.cpp.

6.4.2.7 void Arduboy2Core::digitalWriteRGB (uint8_t *color,* uint8_t *val*) [static]

Set one of the RGB LEDs digitally, to either fully on or fully off.

Parameters

color	The name of the LED to set. The value given should be one of RED_LED, GREEN_LED or BLUE_LED.
val	Indicates whether to turn the specified LED on or off. The value given should be RGB_ON or RGB_OFF.

This 2 parameter version of the function will set a single LED within the RGB LED either fully on or fully off. See the description of the 3 parameter version of this function for more details on the RGB LED.

See also

digitalWriteRGB(uint8_t, uint8_t, uint8_t) setRGBled() freeRGBled()

Definition at line 502 of file Arduboy2Core.cpp.

6.4.2.8 void Arduboy2Core::displayOff ( ) [static]

Turn the display off.

The display will clear and be put into a low power mode. This can be used to extend battery life when a game is paused or when a sketch doesn't require anything to be displayed for a relatively long period of time.

See also

displayOn()

Definition at line 283 of file Arduboy2Core.cpp.

6.4.2.9 void Arduboy2Core::displayOn ( ) [static]

Turn the display on.

Used to power up and reinitialize the display after calling `displayOff()`.

Note

The previous call to `displayOff()` will have caused the display's buffer contents to be lost. The display will have to be re-painted, which is usually done by calling `display()`.

See also

displayOff()

Definition at line 294 of file Arduboy2Core.cpp.

6.4.2.10 void Arduboy2Core::exitToBootloader ( ) [static]

Exit the sketch and start the bootloader.

The sketch will exit and the bootloader will be started in command mode. The effect will be similar to pressing the reset button.

This function is intended to be used to allow uploading a new sketch, when the USB code has been removed to gain more code space. Ideally, the sketch would present a "New Sketch Upload" menu or prompt telling the user to "Press and hold the DOWN button when the procedure to upload a new sketch has been initiated". The sketch would then wait for the DOWN button to be pressed and then call this function.

See also

ARDUBOY_NO_USB

Definition at line 562 of file Arduboy2Core.cpp.

6.4.2.11 void Arduboy2Core::flipHorizontal (bool *flipped*) [static]

Flip the display horizontally or set it back to normal.

Parameters

flipped	`true` will set horizontal flip mode. `false` will set normal horizontal orientation.

Calling this function with a value of `true` will cause the X coordinate to start at the left edge of the display instead of the right, effectively flipping the display horizontally.

Once in horizontal flip mode, it will remain this way until normal horizontal mode is set by calling this function with a value of `false`.

See also

flipVertical()

Definition at line 426 of file Arduboy2Core.cpp.

6.4.2.12 void Arduboy2Core::flipVertical (bool *flipped*) [static]

Flip the display vertically or set it back to normal.

Parameters

flipped	`true` will set vertical flip mode. `false` will set normal vertical orientation.

Calling this function with a value of `true` will cause the Y coordinate to start at the bottom edge of the display instead of the top, effectively flipping the display vertically.

Once in vertical flip mode, it will remain this way until normal vertical mode is set by calling this function with a value of `false`.

See also

flipHorizontal()

Definition at line 420 of file Arduboy2Core.cpp.

6.4.2.13 void Arduboy2Core::freeRGBled ( ) [static]

Relinquish analog control of the RGB LED.

Using the RGB LED in analog mode prevents further use of the LED in digital mode. This function will restore the pins used for the LED, so it can be used in digital mode.

See also

digitalWriteRGB() setRGBled()

Definition at line 479 of file Arduboy2Core.cpp.

6.4.2.14 uint8_t Arduboy2Core::height ( ) [static]

Get the height of the display in pixels.

Returns

The height of the display in pixels.

Note

In most cases, the defined value `HEIGHT` would be better to use instead of this function.

Definition at line 301 of file Arduboy2Core.cpp.

6.4.2.15 void Arduboy2Core::idle ( ) [static]

Idle the CPU to save power.

This puts the CPU in *idle* sleep mode. You should call this as often as you can for the best power savings. The timer 0 overflow interrupt will wake up the chip every 1ms, so even at 60 FPS a well written app should be able to sleep maybe half the time in between rendering it's own frames.

Definition at line 266 of file Arduboy2Core.cpp.

6.4.2.16 void Arduboy2Core::invert (bool *inverse*) [static]

Invert the entire display or set it back to normal.

Parameters

inverse	`true` will invert the display. `false` will set the display to no-inverted.

Calling this function with a value of `true` will set the display to inverted mode. A pixel with a value of 0 will be on and a pixel set to 1 will be off.

Once in inverted mode, the display will remain this way until it is set back to non-inverted mode by calling this function with `false`.

Definition at line 407 of file Arduboy2Core.cpp.

6.4.2.17 void Arduboy2Core::LCDCommandMode ( ) [static]

Put the display into command mode.

When placed in command mode, data that is sent to the display will be treated as commands.

See the SSD1306 controller and OLED display documents for available commands and command sequences.

Links:

- https://www.adafruit.com/datasheets/SSD1306.pdf
- http://www.buydisplay.com/download/manual/ER-OLED013-1_Series_Datasheet.pdf

Note

This is a low level function that is not intended for general use in a sketch. It has been made public and documented for use by derived classes.

See also

LCDDataMode() sendLCDCommand() SPItransfer()

Definition at line 220 of file Arduboy2Core.cpp.

6.4.2.18 void Arduboy2Core::LCDDataMode ( ) [static]

Put the display into data mode.

When placed in data mode, data that is sent to the display will be considered as data to be displayed.

Note

This is a low level function that is not intended for general use in a sketch. It has been made public and documented for use by derived classes.

See also

LCDCommandMode() SPItransfer()

Definition at line 215 of file Arduboy2Core.cpp.

6.4.2.19 void Arduboy2Core::paint8Pixels (uint8_t *pixels*) [static]

Paint 8 pixels vertically to the display.

Parameters

pixels	A byte whose bits specify a vertical column of 8 pixels.

A byte representing a vertical column of 8 pixels is written to the display at the current page and column address. The address is then incremented. The page/column address will wrap to the start of the display (the top left) when it increments past the end (lower right).

The least significant bit represents the top pixel in the column. A bit set to 1 is lit, 0 is unlit.

Example:

```
X = lit pixels, . = unlit pixels

blank()                          paint8Pixels() 0xFF, 0, 0xF0, 0, 0x0F
v TOP LEFT corner (8x9)          v TOP LEFT corner
. . . . . . . . (page 1)         X . . . X . . . (page 1)
. . . . . . . .                  X . . . X . . .
. . . . . . . .                  X . . . X . . .
. . . . . . . .                  X . . . X . . .
. . . . . . . .                  X . X . . . . .
. . . . . . . .                  X . X . . . . .
. . . . . . . .                  X . X . . . . .
. . . . . . . . (end of page 1)  X . X . . . . . (end of page 1)
. . . . . . . . (page 2)         . . . . . . . . (page 2)
```

Definition at line 306 of file Arduboy2Core.cpp.

6.4.2.20 void Arduboy2Core::paintScreen (const uint8_t * *image*) [static]

Paints an entire image directly to the display from program memory.

Parameters

image	A byte array in program memory representing the entire contents of the display.

The contents of the specified array in program memory is written to the display. Each byte in the array represents a vertical column of 8 pixels with the least significant bit at the top. The bytes are written starting at the top left, progressing horizontally and wrapping at the end of each row, to the bottom right. The size of the array must exactly match the number of pixels in the entire display.

See also

paint8Pixels()

Definition at line 311 of file Arduboy2Core.cpp.

6.4.2.21 void Arduboy2Core::paintScreen (uint8_t *image[ ]*, bool *clear* = false) [static]

Paints an entire image directly to the display from an array in RAM.

Parameters

image	A byte array in RAM representing the entire contents of the display.
clear	If `true` the array in RAM will be cleared to zeros upon return from this function. If `false` the RAM buffer will remain unchanged. (optional; defaults to `false`)

The contents of the specified array in RAM is written to the display. Each byte in the array represents a vertical column of 8 pixels with the least significant bit at the top. The bytes are written starting at the top left, progressing horizontally and wrapping at the end of each row, to the bottom right. The size of the array must exactly match the number of pixels in the entire display.

If parameter `clear` is set to `true` the RAM array will be cleared to zeros after its contents are written to the display.

See also

paint8Pixels()

Definition at line 325 of file Arduboy2Core.cpp.

6.4.2.22 void Arduboy2Core::safeMode ( ) [static]

Allow upload when the bootloader "magic number" could be corrupted.

If the UP button is held when this function is entered, the RGB LED will be lit and timer 0 will be disabled, then the sketch will remain in a tight loop. This is to address a problem with uploading a new sketch, for sketches that interfere with the bootloader "magic number". The problem occurs with certain sketches that use large amounts of RAM.

This function should be called after `boot()` in sketches that potentially could cause the problem.

It is intended to replace the `flashlight()` function when more program space is required. If possible, it is more desirable to use `flashlight()`, so that the actual flashlight feature isn't lost.

See also

Arduboy2Base::flashlight() boot()

Definition at line 247 of file Arduboy2Core.cpp.

6.4.2.23 void Arduboy2Core::sendLCDCommand (uint8_t *command*) [static]

Send a single command byte to the display.

Parameters

command	The command byte to send to the display.

The display will be set to command mode then the specified command byte will be sent. The display will then be set to data mode. Multi-byte commands can be sent by calling this function multiple times.

Note

Sending improper commands to the display can place it into invalid or unexpected states, possibly even causing physical damage.

Definition at line 398 of file Arduboy2Core.cpp.

6.4.2.24 void Arduboy2Core::setRGBled (uint8_t *red,* uint8_t *green,* uint8_t *blue*) [static]

Set the light output of the RGB LED.

Parameters

red,green,blue	The brightness value for each LED.

The RGB LED is actually individual red, green and blue LEDs placed very close together in a single package. By setting the brightness of each LED, the RGB LED can show various colors and intensities. The brightness of each LED can be set to a value from 0 (fully off) to 255 (fully on).

Note

Certain libraries that take control of the hardware timers may interfere with the ability of this function to properly control the RGB LED. *ArduboyPlaytune* is one such library known to do this. The digitalWriteRGB() function will still work properly in this case.

Note

Many of the Kickstarter Arduboys were accidentally shipped with the RGB LED installed incorrectly. For these units, the green LED cannot be lit. As long as the green led is set to off, setting the red LED will actually control the blue LED and setting the blue LED will actually control the red LED. If the green LED is turned fully on, none of the LEDs will light.

See also

setRGBled(uint8_t, uint8_t) digitalWriteRGB() freeRGBled()

Definition at line 433 of file Arduboy2Core.cpp.

6.4.2.25 void Arduboy2Core::setRGBled (uint8_t *color,* uint8_t *val*) [static]

Set the brightness of one of the RGB LEDs without affecting the others.

Parameters

color	The name of the LED to set. The value given should be one of RED_LED, GREEN_LED or BLUE_LED.
val	The brightness value for the LED, from 0 to 255.

Note

In order to use this function, the 3 parameter version must first be called at least once, in order to initialize the hardware.

This 2 parameter version of the function will set the brightness of a single LED within the RGB LED without affecting the current brightness of the other two. See the description of the 3 parameter version of this function for more details on the RGB LED.

See also

setRGBled(uint8_t, uint8_t, uint8_t) digitalWriteRGB() freeRGBled()

Definition at line 455 of file Arduboy2Core.cpp.

6.4.2.26 void Arduboy2Core::SPItransfer (uint8_t *data*) [static]

Transfer a byte to the display.

Parameters

data	The byte to be sent to the display.

Transfer one byte to the display over the SPI port and wait for the transfer to complete. The byte will either be interpreted as a command or as data to be placed on the screen, depending on the command/data mode.

See also

LCDDataMode() LCDCommandMode() sendLCDCommand()

Definition at line 234 of file Arduboy2Core.cpp.

6.4.2.27 uint8_t Arduboy2Core::width ( ) [static]

Get the width of the display in pixels.

Returns

The width of the display in pixels.

Note

In most cases, the defined value `WIDTH` would be better to use instead of this function.

Definition at line 299 of file Arduboy2Core.cpp.

The documentation for this class was generated from the following files:

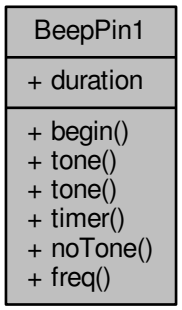
- src/Arduboy2Core.h
- src/Arduboy2Core.cpp

6.5 BeepPin1 Class Reference

Play simple square wave tones using speaker pin 1.

```
#include <Arduboy2Beep.h>
```

Collaboration diagram for BeepPin1:



Static Public Member Functions

- static void begin ()

  Set up the hardware.
- static void tone (uint16_t count)

  Play a tone continually, until replaced by a new tone or stopped.
- static void tone (uint16_t count, uint8_t dur)

  Play a tone for a given duration.
- static void timer ()

  Handle the duration that a tone plays for.
- static void noTone ()

  Stop a tone that is playing.
- static constexpr uint16_t freq (const float hz)

  Convert a frequency to the required count.

Static Public Attributes

- static uint8_t duration = 0

  The counter used by the `timer()` function to time the duration of a tone.

6.5.1 Detailed Description

Play simple square wave tones using speaker pin 1.

Note

Class `BeepPin2` provides identical functions for playing tones on speaker pin 2. Both classes can be used in the same sketch to allow playing two tones at the same time. To do this, the `begin()` and `timer()` functions of both classes must be used.

This class can be used to play square wave tones on speaker pin 1. The functions are designed to produce very small and efficient code.

A tone can be set to play for a given duration, or continuously until stopped or replaced by a new tone. No interrupts are used. A tone is generated by a hardware timer/counter directly toggling the pin, so once started, no CPU cycles are used to actually play the tone. The program continues to run while a tone is playing. However, a small amount of code is required to time and stop a tone after a given duration.

Tone frequencies can range from 15.26Hz to 1000000Hz.

Although there's no specific code to handle mute control, the `Arduboy2Audio` class will work since it has code to mute sound by setting the speaker pins to input mode and unmute by setting the pins as outputs. The `BeepPin1` class doesn't interfere with this operation.

In order to avoid needing to use interrupts, the duration of tones is timed by calling the `timer()` function continuously at a fixed interval. The duration of a tone is given by specifying the number of times `timer()` will be called before stopping the tone.

For sketches that use `Arduboy2::nextFrame()`, or some other method to generate frames at a fixed rate, `timer()` can be called once per frame. Tone durations will then be given as the number of frames to play the tone for. For example, with a rate of 60 frames per second a duration of 30 would be used to play a tone for half a second.

The variable named `duration` is the counter that times the duration of a tone. A sketch can determine if a tone is currently playing by testing if the `duration` variable is non-zero (assuming it's a timed tone, not a continuous tone).

To keep the code small and efficient, the frequency of a tone is specified by the actual count value to be loaded into to timer/counter peripheral. The frequency will be determined by the count provided and the clock rate of the timer/counter. In order to allow a tone's frequency to be specified in hertz (cycles per second) the `freq()` helper function is provided, which converts a given frequency to the required count value.

NOTE that it is intended that `freq()` only be called with constant values. If `freq()` is called with a variable, code to perform floating point math will be included in the sketch, which will likely greatly increase the sketch's code size unless the sketch also uses floating point math for other purposes.

The formulas for frequency/count conversion are:

```
count=(1000000/frequency)-1
frequency=1000000/(count+1)
```

Counts must be between 0 and 65535.

All members of the class are static, so it's not necessary to create an instance of the class in order to use it. However, creating an instance doesn't produce any more code and it may make the source code smaller and make it easier to switch to the `BeepPin2` class if it becomes necessary.

The following is a basic example sketch, which will generate a tone when a button is pressed.

```
#include <Arduboy2.h>
// There's no need to #include <Arduboy2Beep.h>
// It will be included in Arduboy2.h

Arduboy2 arduboy;
BeepPin1 beep; // class instance for speaker pin 1

void setup() {
  arduboy.begin();
  arduboy.setFrameRate(50);
  beep.begin(); // set up the hardware for playing tones
}

void loop() {
  if (!arduboy.nextFrame()) {
    return;
  }

  beep.timer(); // handle tone duration

  arduboy.pollButtons();

  if (arduboy.justPressed(A_BUTTON)) {
    // play a 1000Hz tone for 100 frames (2 seconds at 50 FPS)
    // beep.freq(1000) is used to convert 1000Hz to the required count
    beep.tone(beep.freq(1000), 100);
  }
}
```

Note

These functions, and the equivalents in class `BeepPin2`, will not work with a DevKit Arduboy because the speaker pins used cannot be directly controlled by a timer/counter. "Dummy" functions are provided so a sketch will compile and work properly but no sound will be produced.

See also

BeepPin2

Definition at line 120 of file Arduboy2Beep.h.

6.5.2 Member Function Documentation

6.5.2.1 void BeepPin1::begin ( ) [static]

Set up the hardware.

Prepare the hardware for playing tones. This function must be called (usually in `setup()`) before using any of the other functions in this class.

Definition at line 16 of file Arduboy2Beep.cpp.

6.5.2.2 static constexpr uint16_t BeepPin1::freq (const float *hz*) [inline], [static]

Convert a frequency to the required count.

Parameters

hz	The frequency, in hertz (cycles per second), to be converted to a count.

Returns

The required count to be loaded into the timer/counter for the given frequency.

This helper function will convert a desired tone frequency to the closest value required by the `tone()` function's `count` parameter. The calculated count is rounded up or down to the nearest integer, if necessary.

Example:

```
beep.tone(beep.freq(440)); // play a 440Hz tone until stopped or replaced
```

Note

It is intended that `freq()` only be called with constant values. If `freq()` is called with a variable, code to perform floating point math will be included in the sketch, which will likely greatly increase the sketch's code size unless the sketch also uses floating point math for other purposes.

Definition at line 250 of file Arduboy2Beep.h.

6.5.2.3 void BeepPin1::noTone ( ) [static]

Stop a tone that is playing.

If a tone is playing it will be stopped. It's safe to call this function even if a tone isn't currently playing.

See also

tone()

Definition at line 41 of file Arduboy2Beep.cpp.

6.5.2.4 void BeepPin1::timer ( ) [static]

Handle the duration that a tone plays for.

This function must be called at a constant interval, which would normally be once per frame, in order to stop a tone after the desired tone duration has elapsed.

If the value of the `duration` variable is not 0, it will be decremented. When the `duration` variable is decremented to 0, a playing tone will be stopped.

Definition at line 34 of file Arduboy2Beep.cpp.

6.5.2.5 void BeepPin1::tone (uint16_t *count*) [static]

Play a tone continually, until replaced by a new tone or stopped.

Parameters

count	The count to be loaded into the timer/counter to play the desired frequency.

A tone is played indefinitely, until replaced by another tone or stopped using `noTone()`.

The tone's frequency is determined by the specified count, which is loaded into the timer/counter that generates the tone. A desired frequency can be converted into the required count value using the `freq()` function.

See also

freq() timer() noTone()

Definition at line 22 of file Arduboy2Beep.cpp.

6.5.2.6 void BeepPin1::tone (uint16_t *count*, uint8_t *dur*) [static]

Play a tone for a given duration.

Parameters

count	The count to be loaded into the timer/counter to play the desired frequency.
dur	The duration of the tone, used by `timer()`.

A tone is played for the specified duration, or until replaced by another tone or stopped using `noTone()`.

The tone's frequency is determined by the specified count, which is loaded into the timer/counter that generates the tone. A desired frequency can be converted into the required count value using the `freq()` function.

The duration value is the number of times the `timer()` function must be called before the tone is stopped.

See also

freq() timer() noTone()

Definition at line 27 of file Arduboy2Beep.cpp.

6.5.3 Member Data Documentation

6.5.3.1 uint8_t BeepPin1::duration = 0 [static]

The counter used by the `timer()` function to time the duration of a tone.

This variable is set by the `dur` parameter of the `tone()` function. It is then decremented each time the `timer()` function is called, if its value isn't 0. When `timer()` decrements it to 0, a tone that is playing will be stopped.

A sketch can determine if a tone is currently playing by testing if this variable is non-zero (assuming it's a timed tone, not a continuous tone).

Example:

```
beep.tone(beep.freq(1000), 15);
while (beep.duration != 0) { } // wait for the tone to stop playing
```

It can also be manipulated directly by the sketch, although this should seldom be necessary.

Definition at line 146 of file Arduboy2Beep.h.

The documentation for this class was generated from the following files:

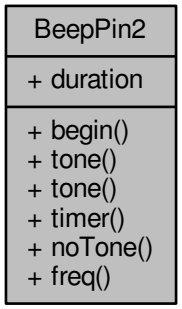
- src/Arduboy2Beep.h
- src/Arduboy2Beep.cpp

6.6 BeepPin2 Class Reference

Play simple square wave tones using speaker pin 2.

`#include <Arduboy2Beep.h>`

Collaboration diagram for BeepPin2:



Static Public Member Functions

- static void begin ()

  Set up the hardware for playing tones using speaker pin 2.
- static void tone (uint16_t count)

  Play a tone on speaker pin 2 continually, until replaced by a new tone or stopped.
- static void tone (uint16_t count, uint8_t dur)

  Play a tone on speaker pin 2 for a given duration.
- static void timer ()

  Handle the duration that a tone on speaker pin 2 plays for.
- static void noTone ()

  Stop a tone that is playing on speaker pin 2.
- static constexpr uint16_t freq (const float hz)

  Convert a frequency to the required count for speaker pin 2.

Static Public Attributes

- static uint8_t duration = 0

  The counter used by the `timer()` function to time the duration of a tone played on speaker pin 2.

6.6.1 Detailed Description

Play simple square wave tones using speaker pin 2.

This class contains the same functions as class `BeepPin1` except they use speaker pin 2 instead of speaker pin 1.

Using `BeepPin1` is more desirable, as it uses a 16 bit Timer, which can produce a greater frequency range and resolution than the 10 bit Timer used by `BeepPin2`. However, if the sketch also includes other sound generating code that uses speaker pin 1, `BeepPin2` can be used to avoid conflict.

Tone frequencies on speaker pin 2 can range from 61.04Hz to 15625Hz using allowed counts from 3 to 1023.

The formulas for frequency/count conversion are:

```
count=(62500/frequency)-1
frequency=62500/(count+1)
```

See the documentation for `BeepPin1` for more details.

See also

BeepPin1

Definition at line 282 of file Arduboy2Beep.h.

6.6.2 Member Function Documentation

6.6.2.1 void BeepPin2::begin ( ) [static]

Set up the hardware for playing tones using speaker pin 2.

For details see `BeepPin1::begin()`.

Definition at line 52 of file Arduboy2Beep.cpp.

6.6.2.2 static constexpr uint16_t BeepPin2::freq (const float *hz*) [inline], [static]

Convert a frequency to the required count for speaker pin 2.

Parameters

hz	The frequency, in hertz (cycles per second), to be converted to a count.

Returns

The required count to be loaded into the timer/counter for the given frequency.

For details see `BeepPin1::freq()`.

Definition at line 355 of file Arduboy2Beep.h.

6.6.2.3 void BeepPin2::noTone ( ) [static]

Stop a tone that is playing on speaker pin 2.

For details see `BeepPin1::noTone()`.

Definition at line 81 of file Arduboy2Beep.cpp.

6.6.2.4 void BeepPin2::timer ( ) [static]

Handle the duration that a tone on speaker pin 2 plays for.

For details see `BeepPin1::timer()`.

Definition at line 74 of file Arduboy2Beep.cpp.

6.6.2.5 void BeepPin2::tone (uint16_t *count*) [static]

Play a tone on speaker pin 2 continually, until replaced by a new tone or stopped.

Parameters

count	The count to be loaded into the timer/counter to play the desired frequency.

For details see `BeepPin1::tone(uint16_t)`.

Definition at line 61 of file Arduboy2Beep.cpp.

6.6.2.6 void BeepPin2::tone (uint16_t *count*, uint8_t *dur*) [static]

Play a tone on speaker pin 2 for a given duration.

Parameters

count	The count to be loaded into the timer/counter to play the desired frequency.
dur	The duration of the tone, used by `timer()`.

For details see `BeepPin1::tone(uint16_t, uint8_t)`.

Definition at line 66 of file Arduboy2Beep.cpp.

6.6.3 Member Data Documentation

6.6.3.1 uint8_t BeepPin2::duration = 0 [static]

The counter used by the `timer()` function to time the duration of a tone played on speaker pin 2.

For details see `BeepPin1::duration`.

Definition at line 293 of file Arduboy2Beep.h.

The documentation for this class was generated from the following files:

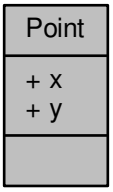
- src/Arduboy2Beep.h
- src/Arduboy2Beep.cpp

6.7 Point Struct Reference

An object to define a single point for collision functions.

`#include <Arduboy2.h>`

Collaboration diagram for Point:



Public Attributes

- int16_t x
- int16_t y

6.7.1 Detailed Description

An object to define a single point for collision functions.

The location of the point is given by X and Y coordinates.

See also

Arduboy2Base::collide(Point, Rect)

Definition at line 117 of file Arduboy2.h.

6.7.2 Member Data Documentation

6.7.2.1 int16_t Point::x

The X coordinate of the point

Definition at line 119 of file Arduboy2.h.

6.7.2.2 int16_t Point::y

The Y coordinate of the point

Definition at line 120 of file Arduboy2.h.

The documentation for this struct was generated from the following file:

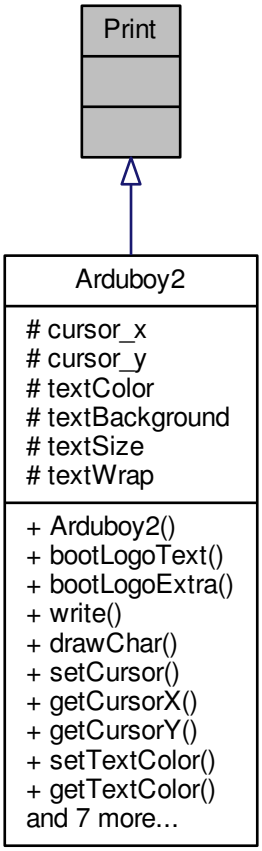
- src/Arduboy2.h

6.8 Print Class Reference

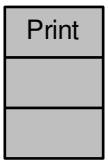
The Arduino `Print` class is available for writing text to the screen buffer.

`#include <Arduboy2.h>`

Inheritance diagram for Print:



Collaboration diagram for Print:



6.8.1 Detailed Description

The Arduino `Print` class is available for writing text to the screen buffer.

For an `Arduboy2` class object, functions provided by the Arduino `Print` class can be used to write text to the screen buffer, in the same manner as the Arduino `Serial.print()`, etc., functions.

Print will use the `write()` function to actually draw each character in the screen buffer.

See: https://www.arduino.cc/en/Serial/Print

Example:

```
int value = 42;

arduboy.println("Hello World"); // Prints "Hello World" and then moves the
                               // text cursor to the start of the next line
arduboy.print(value);  // Prints "42"
arduboy.print('\n');   // Moves the text cursor to the start of the next line
arduboy.print(78, HEX) // Prints "4E" (78 in hexadecimal)
```

See also

Arduboy2::write()

The documentation for this class was generated from the following file:

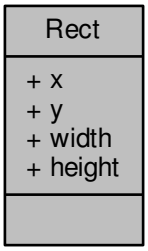
- src/Arduboy2.h

6.9 Rect Struct Reference

A rectangle object for collision functions.

`#include <Arduboy2.h>`

Collaboration diagram for Rect:



Public Attributes

- int16_t x
- int16_t y
- uint8_t width
- uint8_t height

6.9.1 Detailed Description

A rectangle object for collision functions.

The X and Y coordinates specify the top left corner of a rectangle with the given width and height.

See also

Arduboy2Base::collide(Point, Rect) Arduboy2Base::collide(Rect, Rect)

Definition at line 101 of file Arduboy2.h.

6.9.2 Member Data Documentation

6.9.2.1 uint8_t Rect::height

The height of the rectangle

Definition at line 106 of file Arduboy2.h.

6.9.2.2 uint8_t Rect::width

The width of the rectangle

Definition at line 105 of file Arduboy2.h.

6.9.2.3 int16_t Rect::x

The X coordinate of the top left corner

Definition at line 103 of file Arduboy2.h.

6.9.2.4 int16_t Rect::y

The Y coordinate of the top left corner

Definition at line 104 of file Arduboy2.h.

The documentation for this struct was generated from the following file:

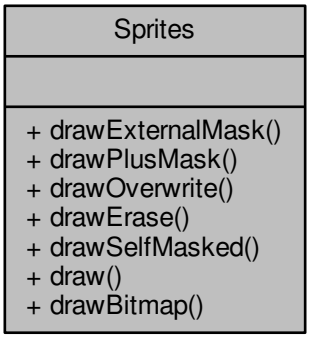
- src/Arduboy2.h

6.10 Sprites Class Reference

A class for drawing animated sprites from image and mask bitmaps.

```
#include <Sprites.h>
```

Collaboration diagram for Sprites:



Static Public Member Functions

- static void drawExternalMask (int16_t x, int16_t y, const uint8_t *bitmap, const uint8_t *mask, uint8_t frame, uint8_t mask_frame)

  Draw a sprite using a separate image and mask array.
- static void drawPlusMask (int16_t x, int16_t y, const uint8_t *bitmap, uint8_t frame)

  Draw a sprite using an array containing both image and mask values.
- static void drawOverwrite (int16_t x, int16_t y, const uint8_t *bitmap, uint8_t frame)

  Draw a sprite by replacing the existing content completely.
- static void drawErase (int16_t x, int16_t y, const uint8_t *bitmap, uint8_t frame)

  "Erase" a sprite.
- static void drawSelfMasked (int16_t x, int16_t y, const uint8_t *bitmap, uint8_t frame)

  Draw a sprite using only the bits set to 1.

6.10.1 Detailed Description

A class for drawing animated sprites from image and mask bitmaps.

The functions in this class will draw to the screen buffer an image contained in an array located in program memory. A mask can also be specified or implied, which dictates how existing pixels in the buffer, within the image boundaries, will be affected.

A sprite or mask array contains one or more "frames". Each frame is intended to show whatever the sprite represents in a different position, such as the various poses for a running or jumping character. By specifying a different frame each time the sprite is drawn, it can be animated.

Each image array begins with values for the width and height of the sprite, in pixels. The width can be any value. The height must be a multiple of 8 pixels, but with proper masking, a sprite of any height can be created.

For a separate mask array, as is used with `drawExternalMask()`, the width and height are not included but must contain data of the same dimensions as the corresponding image array.

Following the width and height values for an image array, or from the beginning of a separate mask array, the array contains the image and/or mask data for each frame. Each byte represents a vertical column of 8 pixels with the least significant bit (bit 0) at the top. The bytes are drawn as 8 pixel high rows from left to right, top to bottom. When the end of a row is reached, as specified by the width value, the next byte in the array will be the start of the next row.

Data for each frame after the first one immediately follows the previous frame. Frame numbers start at 0.

Note

A separate `SpritesB` class is available as an alternative to this class. The only difference is that the `SpritesB` class is optimized for small code size rather than for execution speed. One or the other can be used depending on whether size or speed is more important.
Even if the speed is acceptable when using `SpritesB`, you should still try using `Sprites`. In some cases `Sprites` will produce less code than `SpritesB`, notably when only one of the functions is used.
You can easily switch between using the `Sprites` class or the `SpritesB` class by using one or the other to create an object instance:

```
Sprites sprites;  // Use this to optimize for execution speed
SpritesB sprites; // Use this to (likely) optimize for code size
```

Note

In the example patterns given in each Sprites function description, a # character represents a bit set to 1 and a - character represents a bit set to 0.

See also

SpritesB

Definition at line 75 of file Sprites.h.

6.10.2 Member Function Documentation

6.10.2.1 void Sprites::drawErase (int16_t *x,* int16_t *y,* const uint8_t * *bitmap,* uint8_t *frame*) [static]

"Erase" a sprite.

Parameters

x,y	The coordinates of the top left pixel location.
bitmap	A pointer to the array containing the image frames.
frame	The frame number of the image to erase.

The data from the specified frame in the array is used to erase a sprite. To "erase" a sprite, bits set to 1 in the frame will set the corresponding pixel in the buffer to 0. Frame bits set to 0 will remain unchanged in the buffer.

```
image  before  after  (# = 1, - = 0)

-----  -----   -----
--#--  -----   -----
##-##  -----   -----
--#--  -----   -----
-----  -----   -----

image  before  after

-----  #####   #####
--#--  #####   ##-##
##-##  #####   --#--
--#--  #####   ##-##
-----  #####   #####
```

Definition at line 20 of file Sprites.cpp.

6.10.2.2 void Sprites::drawExternalMask (int16_t *x,* int16_t *y,* const uint8_t * *bitmap,* const uint8_t * *mask,* uint8_t *frame,* uint8_t *mask_frame*) [static]

Draw a sprite using a separate image and mask array.

Parameters

x,y	The coordinates of the top left pixel location.
bitmap	A pointer to the array containing the image frames.
mask	A pointer to the array containing the mask frames.
frame	The frame number of the image to draw.
mask_frame	The frame number for the mask to use (can be different from the image frame number).

An array containing the image frames, and another array containing corresponding mask frames, are used to draw a sprite.

Bits set to 1 in the mask indicate that the pixel will be set to the value of the corresponding image bit. Bits set to 0 in the mask will be left unchanged.

```
image  mask   before  after  (# = 1, - = 0)
```

```
-----  -###-  -----   -----
--#--  #####  -----   --#--
##-##  ##-##  -----   ##-##
--#--  #####  -----   --#--
-----  -###-  -----   -----

image  mask   before  after

-----  -###-  #####   #---#
--#--  #####  #####   --#--
##-##  #####  #####   ##-##
--#--  #####  #####   --#--
-----  -###-  #####   #---#
```

Definition at line 9 of file Sprites.cpp.

6.10.2.3 void Sprites::drawOverwrite (int16_t *x*, int16_t *y*, const uint8_t * *bitmap*, uint8_t *frame*) [static]

Draw a sprite by replacing the existing content completely.

Parameters

x,y	The coordinates of the top left pixel location.
bitmap	A pointer to the array containing the image frames.
frame	The frame number of the image to draw.

A sprite is drawn by overwriting the pixels in the buffer with the data from the specified frame in the array. No masking is done. A bit set to 1 in the frame will set the pixel to 1 in the buffer, and a 0 in the array will set a 0 in the buffer.

```
image  before  after  (# = 1, - = 0)

-----  -----   -----
--#--  -----   --#--
##-##  -----   ##-##
--#--  -----   --#--
-----  -----   -----

image  before  after

-----  #####   -----
--#--  #####   --#--
##-##  #####   ##-##
--#--  #####   --#--
-----  #####   -----
```

Definition at line 15 of file Sprites.cpp.

6.10.2.4 void Sprites::drawPlusMask (int16_t *x*, int16_t *y*, const uint8_t * *bitmap*, uint8_t *frame*) [static]

Draw a sprite using an array containing both image and mask values.

Parameters

x,y	The coordinates of the top left pixel location.
bitmap	A pointer to the array containing the image/mask frames.
frame	The frame number of the image to draw.

An array containing combined image and mask data is used to draw a sprite. Bytes are given in pairs with the first byte representing the image pixels and the second byte specifying the corresponding mask. The width given in the array still specifies the image width, so each row of image and mask bytes will be twice the width value.

Bits set to 1 in the mask indicate that the pixel will be set to the value of the corresponding image bit. Bits set to 0 in the mask will be left unchanged.

```
image  mask   before  after  (# = 1, - = 0)

-----  -###-  -----   -----
--#--  #####  -----   --#--
##-##  ##-##  -----   ##-##
--#--  #####  -----   --#--
-----  -###-  -----   -----

image  mask   before  after

-----  -###-  #####   #---#
--#--  #####  #####   --#--
##-##  #####  #####   ##-##
--#--  #####  #####   --#--
-----  -###-  #####   #---#
```

Definition at line 30 of file Sprites.cpp.

6.10.2.5 void Sprites::drawSelfMasked (int16_t *x*, int16_t *y*, const uint8_t * *bitmap*, uint8_t *frame*) [static]

Draw a sprite using only the bits set to 1.

Parameters

x,y	The coordinates of the top left pixel location.
bitmap	A pointer to the array containing the image frames.
frame	The frame number of the image to draw.

Bits set to 1 in the frame will be used to draw the sprite by setting the corresponding pixel in the buffer to 1. Bits set to 0 in the frame will remain unchanged in the buffer.

```
image  before  after  (# = 1, - = 0)

-----  -----   -----
--#--  -----   --#--
##-##  -----   ##-##
--#--  -----   --#--
-----  -----   -----

image  before  after

-----  #####   #####  (no change because all pixels were
--#--  #####   #####  already white)
##-##  #####   #####
--#--  #####   #####
-----  #####   #####
```

Definition at line 25 of file Sprites.cpp.

The documentation for this class was generated from the following files:

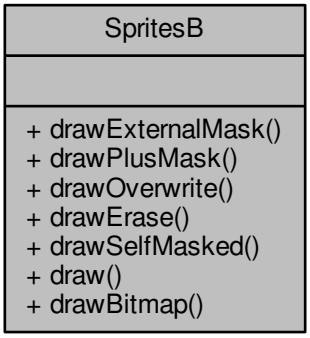
- src/Sprites.h
- src/Sprites.cpp

6.11 SpritesB Class Reference

A class for drawing animated sprites from image and mask bitmaps. Optimized for small code size.

`#include <SpritesB.h>`

Collaboration diagram for SpritesB:



Static Public Member Functions

- static void drawExternalMask (int16_t x, int16_t y, const uint8_t *bitmap, const uint8_t *mask, uint8_t frame, uint8_t mask_frame)

  Draw a sprite using a separate image and mask array.
- static void drawPlusMask (int16_t x, int16_t y, const uint8_t *bitmap, uint8_t frame)

  Draw a sprite using an array containing both image and mask values.
- static void drawOverwrite (int16_t x, int16_t y, const uint8_t *bitmap, uint8_t frame)

  Draw a sprite by replacing the existing content completely.
- static void drawErase (int16_t x, int16_t y, const uint8_t *bitmap, uint8_t frame)

  "Erase" a sprite.
- static void drawSelfMasked (int16_t x, int16_t y, const uint8_t *bitmap, uint8_t frame)

  Draw a sprite using only the bits set to 1.

6.11.1 Detailed Description

A class for drawing animated sprites from image and mask bitmaps. Optimized for small code size.

The functions in this class are identical to the `Sprites` class. The only difference is that the functions in this class are optimized for smaller code size rather than execution speed.

See the `Sprites` class documentation for details on the use of the functions in this class.

Even if the speed is acceptable when using `SpritesB`, you should still try using `Sprites`. In some cases `Sprites` will produce less code than `SpritesB`, notably when only one of the functions is used.

You can easily switch between using the `Sprites` class or the `SpritesB` class by using one or the other to create an object instance:

```
Sprites sprites;  // Use this to optimize for execution speed
SpritesB sprites; // Use this to (likely) optimize for code size
```

See also

Sprites

Definition at line 40 of file SpritesB.h.

6.11.2 Member Function Documentation

6.11.2.1 void SpritesB::drawErase (int16_t *x*, int16_t *y*, const uint8_t * *bitmap*, uint8_t *frame*) [static]

"Erase" a sprite.

Parameters

x,y	The coordinates of the top left pixel location.
bitmap	A pointer to the array containing the image frames.
frame	The frame number of the image to erase.

See also

Sprites::drawErase()

Definition at line 21 of file SpritesB.cpp.

6.11.2.2 void SpritesB::drawExternalMask (int16_t *x*, int16_t *y*, const uint8_t * *bitmap*, const uint8_t * *mask*, uint8_t *frame*, uint8_t *mask_frame*) [static]

Draw a sprite using a separate image and mask array.

Parameters

x,y	The coordinates of the top left pixel location.
bitmap	A pointer to the array containing the image frames.
mask	A pointer to the array containing the mask frames.
frame	The frame number of the image to draw.
mask_frame	The frame number for the mask to use (can be different from the image frame number).

See also

Sprites::drawExternalMask()

Definition at line 10 of file SpritesB.cpp.

6.11.2.3 void SpritesB::drawOverwrite (int16_t *x*, int16_t *y*, const uint8_t * *bitmap*, uint8_t *frame*) [static]

Draw a sprite by replacing the existing content completely.

Parameters

x,y	The coordinates of the top left pixel location.
bitmap	A pointer to the array containing the image frames.
frame	The frame number of the image to draw.

See also

Sprites::drawOverwrite()

Definition at line 16 of file SpritesB.cpp.

6.11.2.4 void SpritesB::drawPlusMask (int16_t *x*, int16_t *y*, const uint8_t * *bitmap*, uint8_t *frame*) [static]

Draw a sprite using an array containing both image and mask values.

Parameters

x,y	The coordinates of the top left pixel location.
bitmap	A pointer to the array containing the image/mask frames.
frame	The frame number of the image to draw.

See also

Sprites::drawPlusMask()

Definition at line 31 of file SpritesB.cpp.

6.11.2.5 void SpritesB::drawSelfMasked (int16_t *x*, int16_t *y*, const uint8_t * *bitmap*, uint8_t *frame*) [static]

Draw a sprite using only the bits set to 1.

Parameters

x,y	The coordinates of the top left pixel location.
bitmap	A pointer to the array containing the image frames.
frame	The frame number of the image to draw.

See also

Sprites::drawSelfMasked()

Definition at line 26 of file SpritesB.cpp.

The documentation for this class was generated from the following files:

- src/SpritesB.h
- src/SpritesB.cpp

Chapter 7

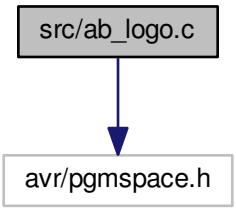
File Documentation

7.1 src/ab_logo.c File Reference

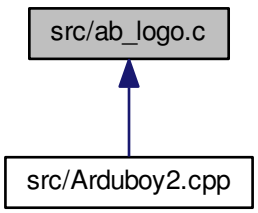
The ARDUBOY logo bitmap.

```
#include <avr/pgmspace.h>
```

Include dependency graph for ab_logo.c:



This graph shows which files directly or indirectly include this file:



7.1.1 Detailed Description

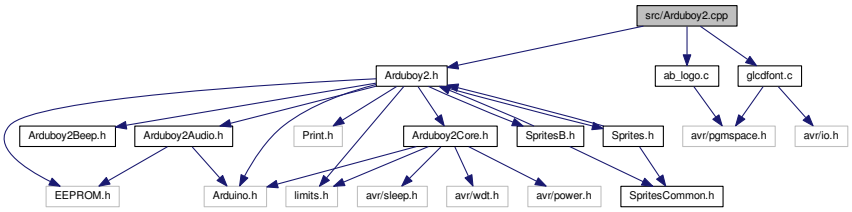
The ARDUBOY logo bitmap.

7.2 src/Arduboy2.cpp File Reference

The Arduboy2Base and Arduboy2 classes and support objects and definitions.

```
#include "Arduboy2.h"
#include "ab_logo.c"
#include "glcdfont.c"
```

Include dependency graph for Arduboy2.cpp:



7.2.1 Detailed Description

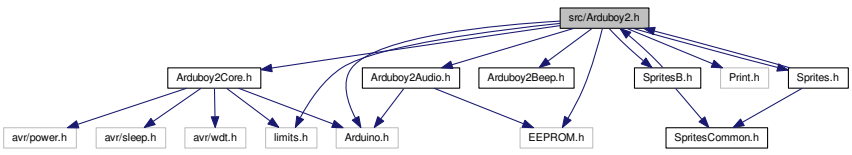
The Arduboy2Base and Arduboy2 classes and support objects and definitions.

7.3 src/Arduboy2.h File Reference

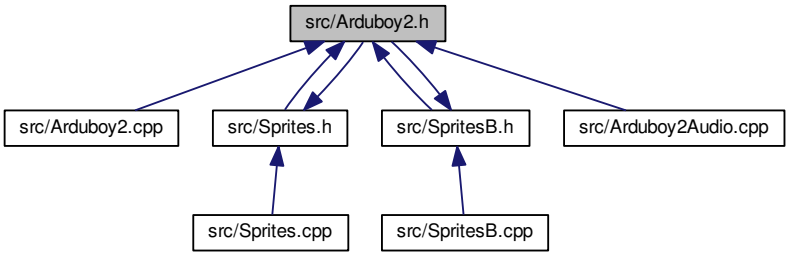
The Arduboy2Base and Arduboy2 classes and support objects and definitions.

```
#include <Arduino.h>
#include <EEPROM.h>
#include "Arduboy2Core.h"
#include "Arduboy2Beep.h"
#include "Sprites.h"
#include "SpritesB.h"
#include <Print.h>
#include <limits.h>
#include "Arduboy2Audio.h"
```

Include dependency graph for Arduboy2.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct Rect

  A rectangle object for collision functions.
- struct Point

  An object to define a single point for collision functions.
- class Arduboy2Base

  The main functions provided for writing sketches for the Arduboy, minus text output.
- class Arduboy2

  The main functions provided for writing sketches for the Arduboy, including text output.

Macros

- #define ARDUBOY_LIB_VER 50100

  Library version.
- #define ARDUBOY_UNIT_NAME_LEN 6
- #define EEPROM_STORAGE_SPACE_START 16

  Start of EEPROM storage space for sketches.
- #define BLACK 0
- #define WHITE 1
- #define INVERT 2

  Color value to indicate pixels are to be inverted.
- #define CLEAR_BUFFER true

7.3.1 Detailed Description

The Arduboy2Base and Arduboy2 classes and support objects and definitions.

7.3.2 Macro Definition Documentation

7.3.2.1 #define ARDUBOY_LIB_VER 50100

Library version.

For a version number in the form of x.y.z the value of the define will be ((x ∗ 10000) + (y ∗ 100) + (z)) as a decimal number. So, it will read as xxxyyzz, with no leading zeros on x.

A user program can test this value to conditionally compile based on the library version. For example:

```
// If the library is version 2.1.0 or higher
#if ARDUBOY_LIB_VER >= 20100
  // ... code that make use of a new feature added to V2.1.0
#endif
```

Definition at line 37 of file Arduboy2.h.

7.3.2.2 #define ARDUBOY_UNIT_NAME_LEN 6

The maximum length of the unit name string.

Definition at line 40 of file Arduboy2.h.

7.3.2.3 #define BLACK 0

Color value for an unlit pixel for draw functions.

Definition at line 76 of file Arduboy2.h.

7.3.2.4 #define CLEAR_BUFFER true

Value to be passed to `display()` to clear the screen buffer.

Definition at line 89 of file Arduboy2.h.

7.3.2.5 #define EEPROM_STORAGE_SPACE_START 16

Start of EEPROM storage space for sketches.

An area at the start of EEPROM is reserved for system use. This define specifies the first EEPROM location past the system area. Sketches can use locations from here to the end of EEPROM space.

Definition at line 66 of file Arduboy2.h.

7.3.2.6 #define INVERT 2

Color value to indicate pixels are to be inverted.

BLACK pixels will become WHITE and WHITE will become BLACK.

Note

Only function Arduboy2Base::drawBitmap() currently supports this value.

Definition at line 87 of file Arduboy2.h.

7.3.2.7 #define WHITE 1

Color value for a lit pixel for draw functions.

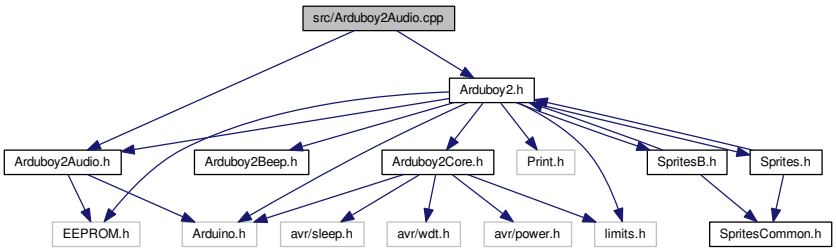
Definition at line 77 of file Arduboy2.h.

7.4 src/Arduboy2Audio.cpp File Reference

The Arduboy2Audio class for speaker and sound control.

```
#include "Arduboy2.h"
#include "Arduboy2Audio.h"
```

Include dependency graph for Arduboy2Audio.cpp:



7.4.1 Detailed Description

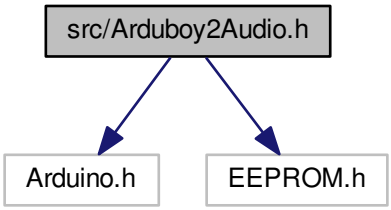
The Arduboy2Audio class for speaker and sound control.

7.5 src/Arduboy2Audio.h File Reference

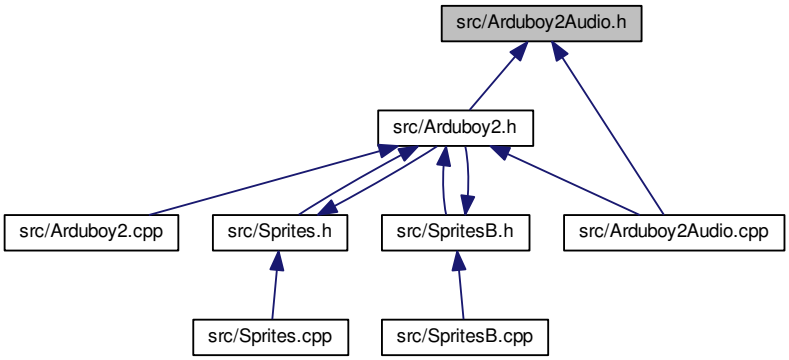
The Arduboy2Audio class for speaker and sound control.

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

Include dependency graph for Arduboy2Audio.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Arduboy2Audio

  Provide speaker and sound control.

7.5.1 Detailed Description

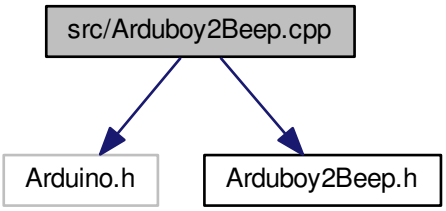
The Arduboy2Audio class for speaker and sound control.

7.6 src/Arduboy2Beep.cpp File Reference

Classes to generate simple square wave tones on the Arduboy speaker pins.

```
#include <Arduino.h>
#include "Arduboy2Beep.h"
```

Include dependency graph for Arduboy2Beep.cpp:



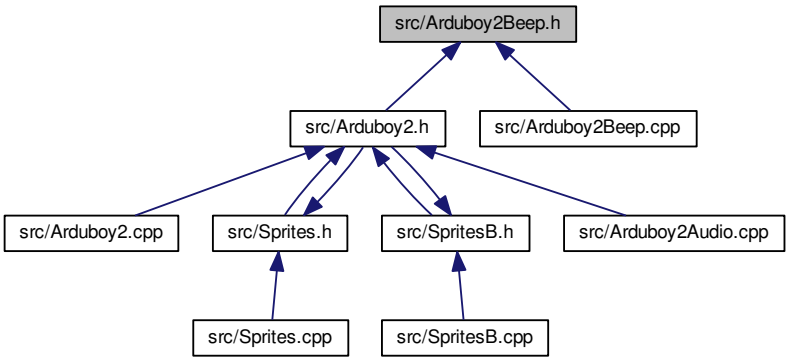
7.6.1 Detailed Description

Classes to generate simple square wave tones on the Arduboy speaker pins.

7.7 src/Arduboy2Beep.h File Reference

Classes to generate simple square wave tones on the Arduboy speaker pins.

This graph shows which files directly or indirectly include this file:



Classes

- class BeepPin1

  Play simple square wave tones using speaker pin 1.
- class BeepPin2

  Play simple square wave tones using speaker pin 2.

7.7.1 Detailed Description

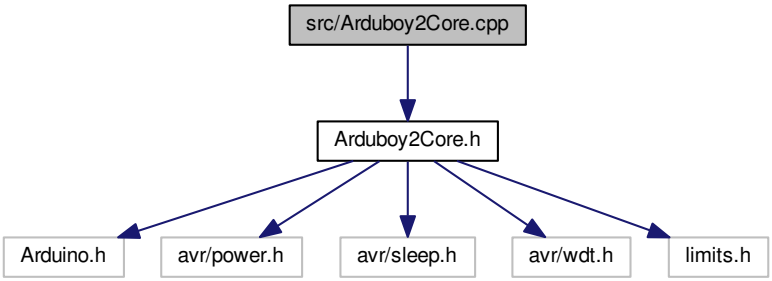
Classes to generate simple square wave tones on the Arduboy speaker pins.

7.8 src/Arduboy2Core.cpp File Reference

The Arduboy2Core class for Arduboy hardware initilization and control.

```
#include "Arduboy2Core.h"
```

Include dependency graph for Arduboy2Core.cpp:



7.8.1 Detailed Description

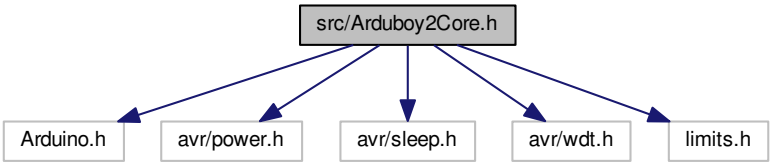
The Arduboy2Core class for Arduboy hardware initilization and control.

7.9 src/Arduboy2Core.h File Reference

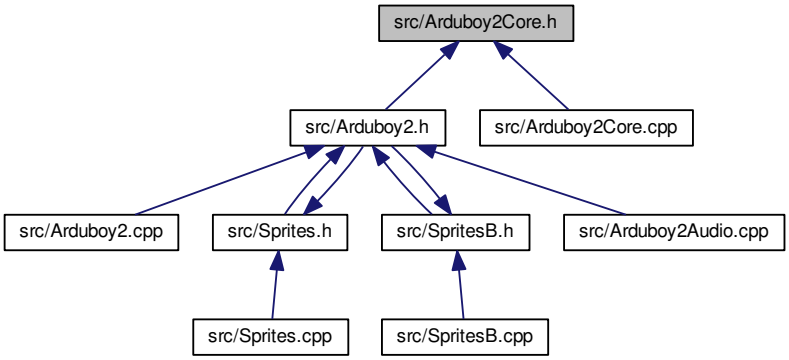
The Arduboy2Core class for Arduboy hardware initilization and control.

```
#include <Arduino.h>
#include <avr/power.h>
#include <avr/sleep.h>
#include <avr/wdt.h>
#include <limits.h>
```

Include dependency graph for Arduboy2Core.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Arduboy2Core

  Lower level functions generally dealing directly with the hardware.

Macros

- #define RGB_ON LOW
- #define RGB_OFF HIGH
- #define RED_LED 10
- #define GREEN_LED 11
- #define BLUE_LED 9
- #define LEFT_BUTTON _BV(5)
- #define RIGHT_BUTTON _BV(6)
- #define UP_BUTTON _BV(7)

- #define DOWN_BUTTON _BV(4)
- #define A_BUTTON _BV(3)
- #define B_BUTTON _BV(2)
- #define PIN_SPEAKER_1 5
- #define PIN_SPEAKER_2 13
- #define WIDTH 128
- #define HEIGHT 64
- #define ARDUBOY_NO_USB

  Eliminate the USB stack to free up code space.

7.9.1 Detailed Description

The Arduboy2Core class for Arduboy hardware initilization and control.

7.9.2 Macro Definition Documentation

7.9.2.1 #define A_BUTTON _BV(3)

The A button value for functions requiring a bitmask

Definition at line 73 of file Arduboy2Core.h.

7.9.2.2 #define ARDUBOY_NO_USB

Value:

```
int main() __attribute__ ((OS_main)); \
int main() { \
  Arduboy2Core::mainNoUSB(); \
  return 0; \
}
```

Eliminate the USB stack to free up code space.

Note

WARNING: Removing the USB code will make it impossible for sketch uploader programs to automatically force a reset into the bootloader! This means that a user will manually have to invoke a reset in order to upload a new sketch, after one without USB has be been installed. Be aware that the timing for the point that a reset must be initiated can be tricky, which could lead to some frustration on the user's part.

This macro will cause the USB code, normally included in the sketch as part of the standard Arduino environment, to be eliminated. This will free up a fair amount of program space, and some RAM space as well, at the expense of disabling all USB functionality within the sketch (except as power input).

The macro should be placed before the `setup()` function definition:

```
#include <Arduboy2.h>

Arduboy2 arduboy;

// (Other variable declarations, etc.)

// Eliminate the USB stack
ARDUBOY_NO_USB

void setup() {
  arduboy.begin();
  // any additional setup code
}
```

As stated in the warning above, without the USB code an uploader program will be unable to automatically force a reset into the bootloader to upload a new sketch. The user will have to manually invoke a reset. In addition to eliminating the USB code, this macro will check if the DOWN button is held when the sketch first starts and, if so, will call `exitToBootloader()` to start the bootloader for uploading. This makes it easier for the user than having to press the reset button.

However, to make it even more convenient for a user to invoke the bootloader it is highly recommended that a sketch using this macro include a menu or prompt that allows the user to press the DOWN button within the sketch, which should cause `exitToBootloader()` to be called.

At a minimum, the documentation for the sketch should clearly state that a manual reset will be required, and give detailed instructions on what the user must do to upload a new sketch.

See also

Arduboy2Core::exitToBootloader()

Definition at line 312 of file Arduboy2Core.h.

7.9.2.3 #define B_BUTTON _BV(2)

The B button value for functions requiring a bitmask

Definition at line 74 of file Arduboy2Core.h.

7.9.2.4 #define BLUE_LED 9

The pin number for the blue color in the RGB LED.

Definition at line 56 of file Arduboy2Core.h.

7.9.2.5 #define DOWN_BUTTON _BV(4)

The Down button value for functions requiring a bitmask

Definition at line 72 of file Arduboy2Core.h.

7.9.2.6 #define GREEN_LED 11

The pin number for the greem color in the RGB LED.

Definition at line 55 of file Arduboy2Core.h.

7.9.2.7 #define HEIGHT 64

The height of the display in pixels

Definition at line 251 of file Arduboy2Core.h.

7.9.2.8 #define LEFT_BUTTON _BV(5)

The Left button value for functions requiring a bitmask

Definition at line 69 of file Arduboy2Core.h.

7.9.2.9 #define PIN_SPEAKER_1 5

The pin number of the first lead of the speaker

Definition at line 112 of file Arduboy2Core.h.

7.9.2.10 #define PIN_SPEAKER_2 13

The pin number of the second lead of the speaker

Definition at line 113 of file Arduboy2Core.h.

7.9.2.11 #define RED_LED 10

The pin number for the red color in the RGB LED.

Definition at line 54 of file Arduboy2Core.h.

7.9.2.12 #define RGB_OFF HIGH

For digitially setting an RGB LED off using digitalWriteRGB()

Definition at line 37 of file Arduboy2Core.h.

7.9.2.13 #define RGB_ON LOW

For digitially setting an RGB LED on using digitalWriteRGB()

Definition at line 36 of file Arduboy2Core.h.

7.9.2.14 #define RIGHT_BUTTON _BV(6)

The Right button value for functions requiring a bitmask

Definition at line 70 of file Arduboy2Core.h.

7.9.2.15 #define UP_BUTTON _BV(7)

The Up button value for functions requiring a bitmask

Definition at line 71 of file Arduboy2Core.h.

7.9.2.16 #define WIDTH 128

The width of the display in pixels

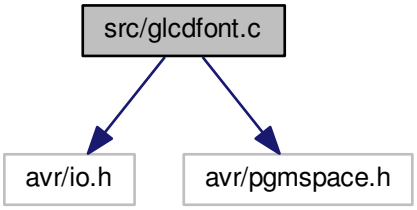
Definition at line 250 of file Arduboy2Core.h.

7.10 src/glcdfont.c File Reference

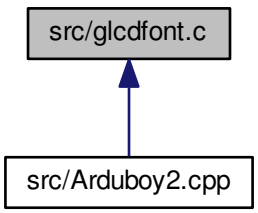
The font definitions used to display text characters.

```
#include <avr/io.h>
#include <avr/pgmspace.h>
```

Include dependency graph for glcdfont.c:



This graph shows which files directly or indirectly include this file:



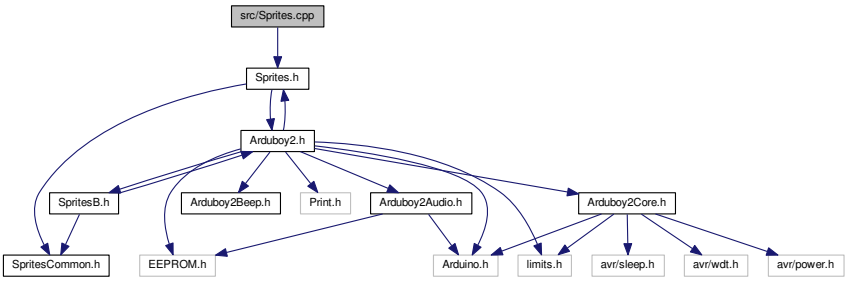
7.10.1 Detailed Description

The font definitions used to display text characters.

7.11 src/Sprites.cpp File Reference

A class for drawing animated sprites from image and mask bitmaps.

```
#include "Sprites.h"
```
Include dependency graph for Sprites.cpp:



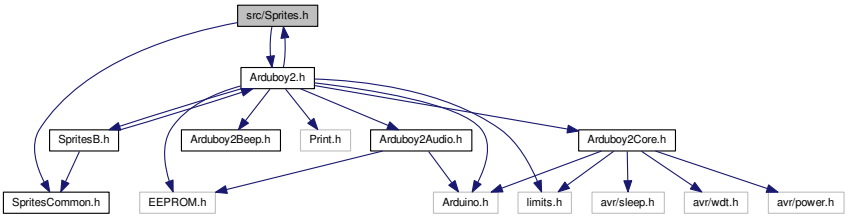
7.11.1 Detailed Description

A class for drawing animated sprites from image and mask bitmaps.

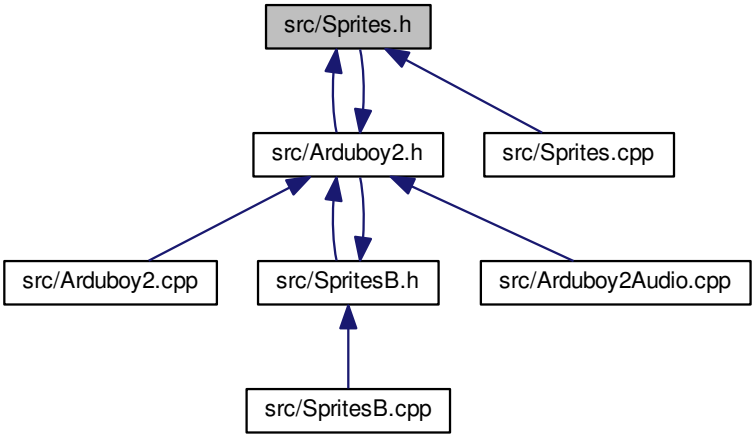
7.12 src/Sprites.h File Reference

A class for drawing animated sprites from image and mask bitmaps.

```
#include "Arduboy2.h"
#include "SpritesCommon.h"
```
Include dependency graph for Sprites.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Sprites

  A class for drawing animated sprites from image and mask bitmaps.

7.12.1 Detailed Description

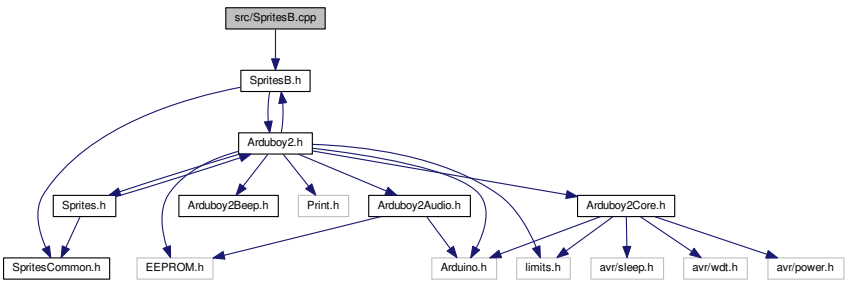
A class for drawing animated sprites from image and mask bitmaps.

7.13 src/SpritesB.cpp File Reference

A class for drawing animated sprites from image and mask bitmaps. Optimized for small code size.

```
#include "SpritesB.h"
```

Include dependency graph for SpritesB.cpp:



7.13.1 Detailed Description

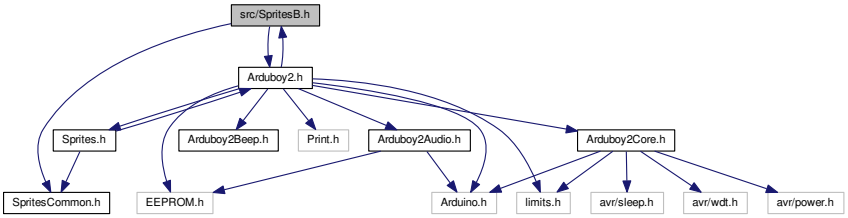
A class for drawing animated sprites from image and mask bitmaps. Optimized for small code size.

7.14 src/SpritesB.h File Reference

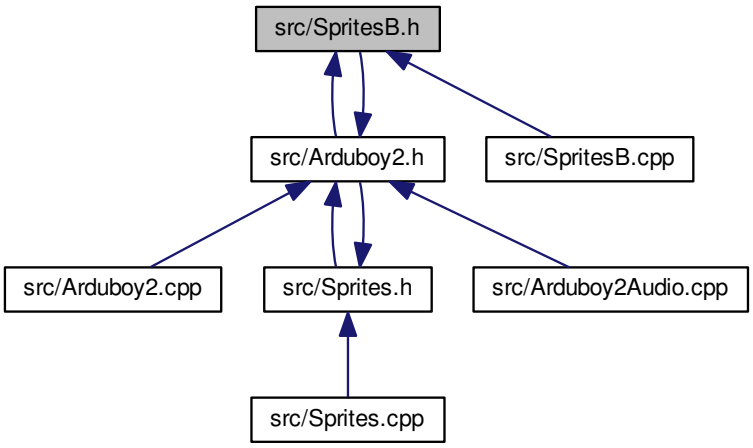
A class for drawing animated sprites from image and mask bitmaps. Optimized for small code size.

```
#include "Arduboy2.h"
#include "SpritesCommon.h"
```

Include dependency graph for SpritesB.h:



This graph shows which files directly or indirectly include this file:



Classes

- class SpritesB

  A class for drawing animated sprites from image and mask bitmaps. Optimized for small code size.

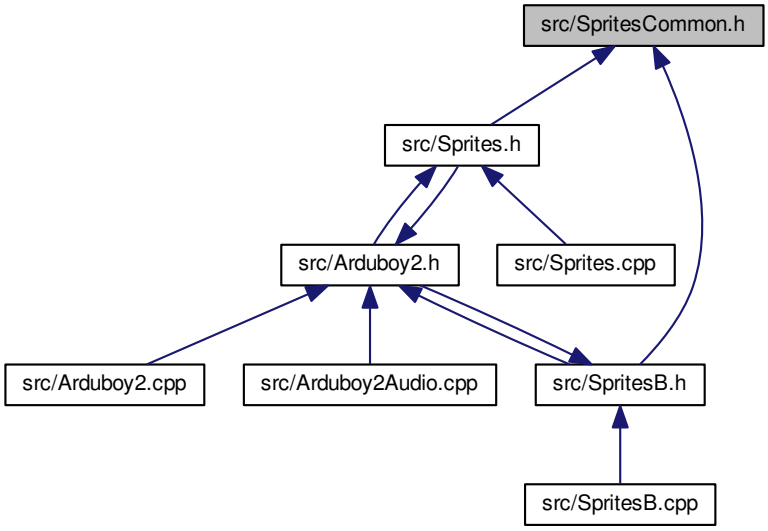
7.14.1 Detailed Description

A class for drawing animated sprites from image and mask bitmaps. Optimized for small code size.

7.15 src/SpritesCommon.h File Reference

Common header file for sprite functions.

This graph shows which files directly or indirectly include this file:



7.15.1 Detailed Description

Common header file for sprite functions.

Index