

Vorwort

Diese Einführung in die Programmierung eines Arduino-basierenden Systems wie den MAKERbuino soll lediglich Geschmack auf mehr machen. Sie erhebt nicht den Anspruch auf Vollständigkeit. Beim Programmieren kommt man nicht wirklich um die englische Sprache herum. Viele Begriffe und vor allem die Befehle sind englisch. Daher wird hier versucht die Zusammenhänge zwischen den englischen Begriffen herzustellen. Es ist wichtig, dass man diese Begriffe versteht, damit man versteht was passiert und wo man eingreifen muss.

Viel Spaß damit

Die Entwicklungsumgebung

Um Programme für den MAKERbuino oder andere Arduino-basierten Systems schreiben zu können benötigt man ein paar Programme. Zunächst benötigt man einen sogenannten Editor in dem man seinen Quelltext oder in Englisch Sourcecode schreiben kann. Ein Editor ist also ein Programm in dem man Texte schreiben kann. Das ist der Ausgangspunkt für jedes Programm, dass man schreiben möchte. Der Sourcecode ist die Beschreibung der eigentlichen Abläufe im Programm. Er wird in einer für uns Menschen „verständlichen“ Sprache der sogenannten Programmiersprache geschrieben. Leider kann der Prozessor diese Sprachen nicht direkt verstehen. Er spricht nur „binären Op-Code“ ¹⁾. Damit der Prozessor trotzdem weiß, was er zu tun hat, muss der Sourcecode in binären Op-Code übersetzt werden. Diese Aufgabe übernimmt der ogenannte Compiler. Er versteht den Sourcecode und kann diesen dann in die nötigen Op-Code-Befehle, die der Prozessor benötigt übersetzen.

Es gibt noch weitere Helfer, aber dazu später mehr. Zunächst müssen wir also einen Editor und einen Compiler für den verwendeten Prozessor, den **Arduino**, installieren. Nun könnten wir alles einzeln installieren oder wir verwenden eine sogenannte IDE ²⁾

Als IDE verwenden wir die Ardunio-IDE die [hier heruntergeladen](#) werden kann. Aktuell ist die Version 1.8.5 (Stand: Januar 2018). Im zweiten Abschnitt befinden sich die Download-Links für die verschiedenen Betriebssysteme. Weg zum Download: arduino.cc → Software

Download the Arduino IDE



ARDUINO 1.8.5
The open-source Arduino Software (IDE) makes it easy to write code and upload it to the board. It runs on Windows, Mac OS X, and Linux. The environment is written in Java and based on Processing and other open-source software.
This software can be used with any Arduino board. Refer to the [Getting Started](#) page for installation instructions.

Windows Installer
Windows ZIP file for non-admin install

Windows app Requires Win 8.1 or 10
[Get](#)

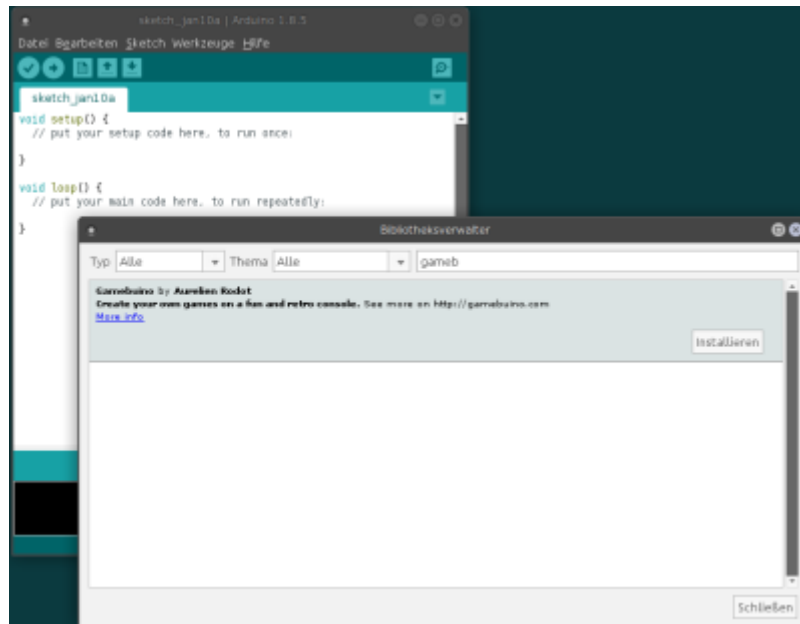
Mac OS X 10.7 Lion or newer

Linux 32 bits
Linux 64 bits
Linux ARM

[Release Notes](#)
[Source Code](#)
[Checksums \(SHA512\)](#)

Die Arduino-IDE wird wie jedes Programm einfach installiert. Anschließend kann die Arduino-IDE gestartet werden. Es sollte der Dialog erscheinen, der sketch_... Arduino 1.8.5 im Titel anzeigt. Damit wir für den MAKERbuino einer speziellen Form des Arduino Programme entwickeln können, benötigen wir noch eine sogenannte Bibliothek oder im Englischen Library in der spezielle bereits vorbereitete Programmteile zusammengefasst sind, die uns das Leben mit dem MAKERbuino

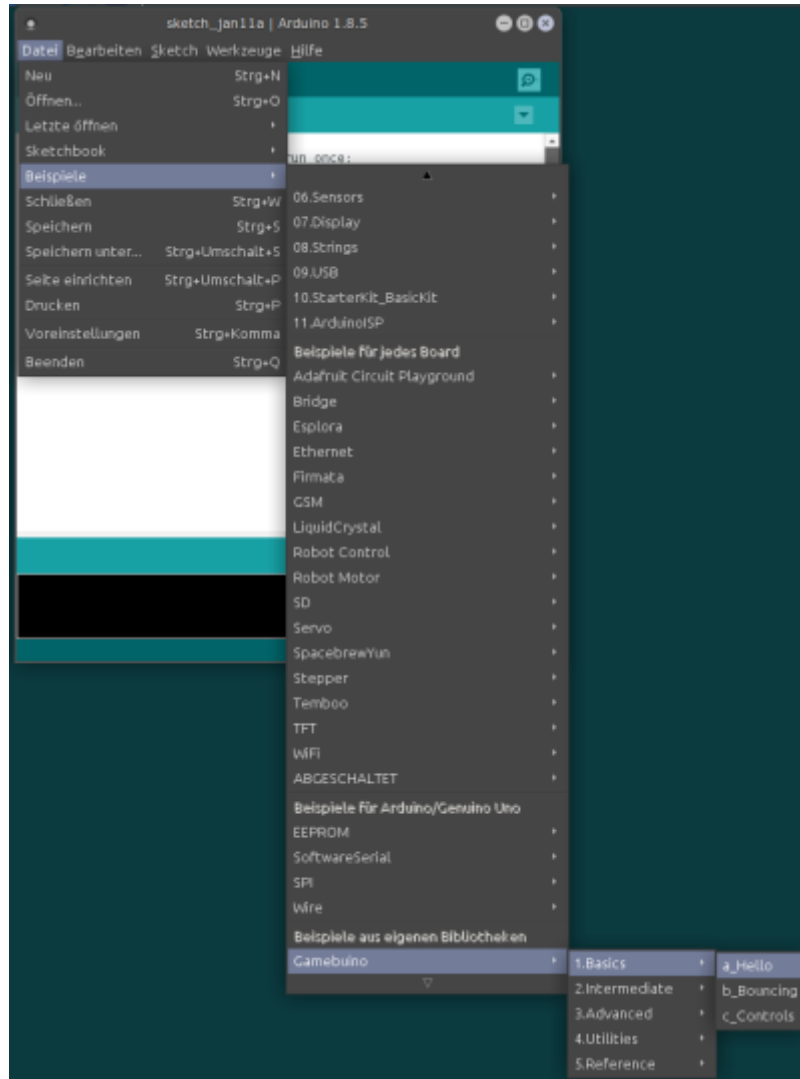
vereinfachen. Der MAKERbuino ist kompatibel, also verhält sich genauso, zum Gamebuino. Daher benötigen wir die Bibliothek des Gamebuino. Diese kann einfach nachinstalliert werden. Über Sketch → Bibliothek einbinden → Bibliotheken verwalten... Hier kann im oberen rechten Bereich gamebuino eingegeben werden. Es sollte dann wie im Bild unten eine kurze Information zur Bibliothek angezeigt werden. Klickt man in diesen Bereich, dann erscheint ein Installieren-Button, den man anklickt. Nach einer kurzen Installation steht im oberen Bereich nun INSTALLED.



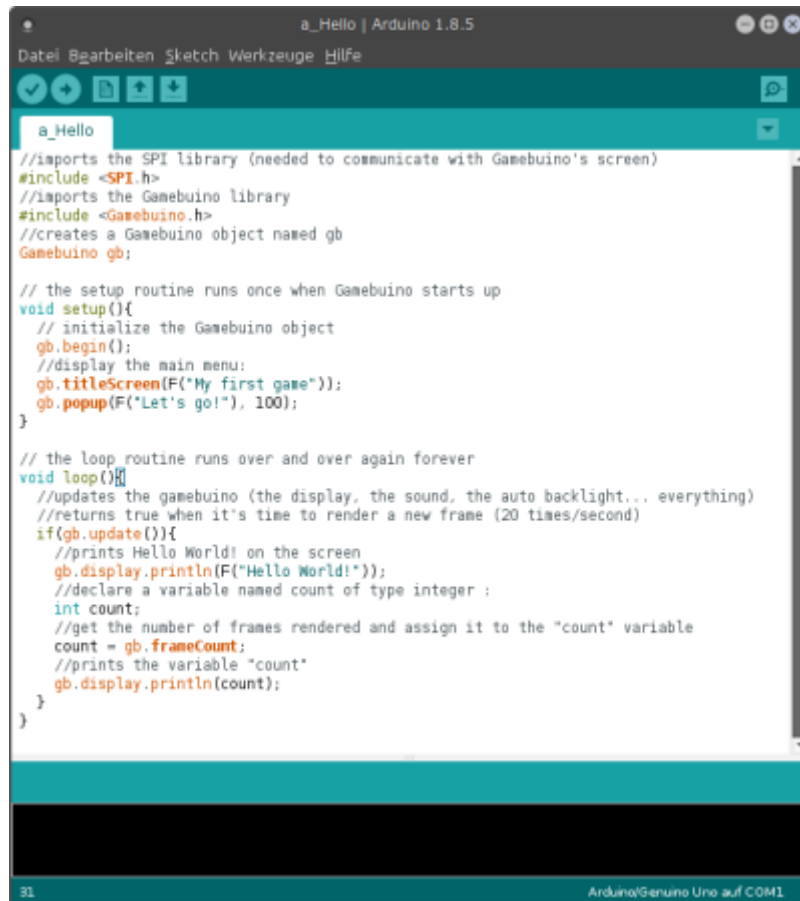
WICHTIG: Nach der Installation der Bibliothek muss die Arduino-IDE ggf. neugestartet werden, damit die neue Bibliothek eingebunden wird.

Das erste Programm "Hello World!"

Jetzt geht es an das erste Programm. Hierzu öffnen wir ein Beispielprogramm des Gamebuino.



Folgender Sourcecode sollte erscheinen:



```
a_Hello | Arduino 1.8.5
Datei Bearbeiten Sketch Werkzeuge Hilfe

a_Hello
//imports the SPI library (needed to communicate with Gamebuino's screen)
#include <SPI.h>
//imports the Gamebuino library
#include <Gamebuino.h>
//creates a Gamebuino object named gb
Gamebuino gb;

// the setup routine runs once when Gamebuino starts up
void setup(){
  // initialize the Gamebuino object
  gb.begin();
  //display the main menu:
  gb.titleScreen(F("My first game"));
  gb.popup(F("Let's go!"), 100);
}

// the loop routine runs over and over again forever
void loop(){
  //updates the gamebuino (the display, the sound, the auto backlight... everything)
  //returns true when it's time to render a new frame (20 times/second)
  if(gb.update()){
    //prints Hello World! on the screen
    gb.display.println(F("Hello World!"));
    //declare a variable named count of type integer :
    int count;
    //get the number of frames rendered and assign it to the "count" variable
    count = gb.frameCount;
    //prints the variable "count"
    gb.display.println(count);
  }
}
```

Kurze Erklärung der Abschnitte (kann übersprungen werden)

Im Folgenden werden die Grundlagen einer Sourcecode-Datei erklärt.

Der erste Abschnitt: Einbinden von fremdem Sourcecode

Der erste Abschnitt des Sourcecodes bindet alle notwendigen Bibliotheken ein. Dies geschieht mit dem Befehl `#include`³⁾. Das Zeichen `#` gibt dem Compiler eine Anweisung, also ist `include` kein Befehl, der in binäre Code für den Prozessor übersetzt wird, sondern lediglich an den Compiler selbst gerichtet ist.

In spitzen Klammern wird nun die sogenannte Header-Datei der Bibliothek angegeben. In diesen Header-Dateien erklärt man normalerweise welche Variablen bzw. Funktionen innerhalb einer Bibliothek zur Verfügung stehen. Durch das `#include` wird der Inhalt genau diese Header-Dateien während der Übersetzung des Sourcecodes in den eigenen Sourcecode „kopiert“. Damit wird alles aus den entsprechenden Bibliotheken im eigenen Sourcecode nutzbar ohne den fremden Sourcecode tatsächlich immer kopieren zu müssen.

Hier werden also die beiden Bibliotheken `SPI`⁴⁾ einen `int`-Wert⁵⁾ und gibt ebenfalls einen `int`-Wert zurück hier `return var1`. Bei `void setup()` bedeutet `void` in diesem Zusammenhang kein Rückgabewert und die leeren runden Klammern bedeuten kein Eingangswert. `==== setup()`: Die Initialisierung des Systems `====` Die `setup()`-Funktion wird genau einmal beim Programmstart

durchlaufen und dient dazu die Grundkonfiguration des Programms vorzunehmen. Hier werden meist die globalen Variablen wie `gb` initialisiert, also mit Werten befüllt `gb.begin()`. Auch kann man hier z.B. den Titelbildschirm `gb.titleScreen` einblenden lassen oder sogenannte popup-Nachrichten `gb.popup` (eingblendete Nachrichten). Wichtig**: Hier gehört nicht hinein, dass mehrfach aufgeführt werden sollen. Also irgendwelche Steuerungen oder sonstige Programmteile. Ansonsten passiert alles genau einmal und danach passiert nichts mehr. ===== `loop()`: Die eigentliche Hauptroutine ===== Die `loop()`-Funktion ist die Hauptroutine, die immer und immer wieder durchlaufen wird. Das bedeutet das alle Befehle, die innerhalb dieser Funktion stehen immer wieder durchlaufen werden. Also sollte darüber gut nachgedacht werden, was hier passieren soll. Es wird ansonsten sehr oft wiederholt. Im Kommentar wird erklärt, dass der Gamebuino 20 mal in der Sekunde ein `update` als eine Aktualisierung durchführt. Innerhalb dieses `update`s wird der Bildschirm, die Soundausgabe, die Hintergrundbeleuchtung usw. aktualisiert. Darum müssen wir also glücklicherweise nicht kümmern (sofern alles wie gewünscht funktioniert). Während eines solchen Aktualisierungszyklus sollte man keine Änderungen durch das eigene Programm durchführen. Dies könnte zu unschönen Ergebnissen führen z.B. ist der obere Teil des Bildschirms schon aktualisiert worden und kurz danach wird ein Wert für diesen Teil geändert. Damit also nicht permanent etwas geschieht, kann man darauf warten, dass ein solcher `update`-Prozess durchlaufen wurde und im Anschluss einige Befehle abarbeiten. Das Objekt `gb` hat hierzu eine Funktion((Methode)) definiert, die mit `gb.update()` aufrufen werden kann. Der `.`-Punkt zwischen `gb` und `update()` deutet daraufhin, dass die Funktion `update()` ein Teil des Objektes `gb` ist.

Mit Hilfe der Funktion `gb.update()` lässt darauf warten, dass der Gamebuino einen `update`-Zyklus beendet hat, in dem man innerhalb der `loop`-Funktion mit Hilfe einer `if`-Anweisung auf den Rückgabe werden `true` prüft.

```
if (gb.update()){
    ...
}
```

1)

Op-Code: Operationcode, also Befehle, die durch Zahlenwerte dargestellt werden

2)

IDE: **i**ntegrated **d**evelopment **e**nviroment; integrierte Entwicklungsumgebungen enthalten alle notwendigen Programme wie Editor, Compiler, Linker etc. und sind meist sehr komfortabel

3)

`include`: Binde ein; macht z.B. eine fremde Bibliothek im eigenen Sourcecode nutzbar

4)

SPI: **s***erial **p***rogramming **i***nterface, **serielle Programmierschnittstelle**) **zum nutzen der Programmierschnittstelle und Gamebuino für den MAKERbuino selbst eingebunden. Weiterhin wird hier noch die Variable gb definiert. Vor dieser Variable steht der sogenannte Type der Variable. Hier Gamebuino. Dieser Variablentype wird in der Header-Datei Gamebuino.h definiert und enthält alles Wichtige um den MAKERbuino zu programmieren. Die Variable existiert zwar zu diesem Zeitpunkt, aber sie kann noch nicht genutzt werden. Um dies zu tun muss sie zunächst initialisiert werden, also quasi mit Leben gefüllt werden. Das heißt das Gerüst der Variablen existiert schon, aber der Inhalt ist noch leer. ===== Der zweite Abschnitt: Der Sourcecode ===== Im zweiten Abschnitt kommt der eigentliche Sourcecode. Hier sind zwei sogenannte Funktionen (`void setup(){...}` und `void loop(){...}`) definiert. Um zu verstehen was hier passiert, muss man wissen, dass es bei vielen Systemen festdefinierte Funktionen gibt, die bestimmte Aufgaben haben. Die beiden `setup` und `loop` gehören dazu. Alle Funktionen haben denselben Aufbau: `<code> RÜCKGABEWERT FUNKTIONSNAME(EINGANGSWERTE){ BEFEHLE, die innerhalb der Funktion abgearbeitet werden }` Beispiel: `int myfunct(int var1)`**

`{ print(var1); var1=var1+1; return var1; }` </code> Die Funktion heißt hier myfunct sie erwartet als Eingangswert((an sie gerichtet
5)

int: **integer**; ganzzahlige Zahl zB. -3, 2 oder 5

From:
<https://www.kopfload.de/> - **kopfload - Lad Dein Hirn auf!**

Permanent link:
https://www.kopfload.de/doku.php?id=allgemein:howto:makerbuino_programmierung&rev=1515686755

Last update: **2025/11/19 16:13**

