DeviceLib in C

voor arduino en raspberry

2020, D. E. Repolev

Creative commons licentiebepaling

Volledige tekst van de licentiebepaling is te vinden op:

https://creativecommons.org/licenses/by-nc-sa/4.0/legalcode.nl

Samengevat ben je vrij om:

- het werk te delen (kopiëren, verspreiden, doorgeven)
- het werk te bewerken (remixen, veranderen, afgeleide werken)

Onder de volgende voorwaarden:

- naamsvermelding van de maker
- een link naar de volledige tekst van de licentie plaatsen
- duidelijk aangeven of het werk veranderd is
- niet de indruk wekken dat de licentiegever instemt met het gebruik van het werk of met de bewerking ervan
- bewerkingen mogen alleen worden verspreid onder dezelfde licentie als het originele werk
- geen gebruik voor commerciële doeleinden
- geen aanvullende restricties (juridisch of technologisch) naast deze licentie toepassen

Inleiding

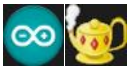
Deze gids vormt de documentatie bij de github repository ***DeviceLib***. *DeviceLib* is een C++ library voor arduino en raspberry met classes voor diverse populaire sensoren en actuatoren.

Het onoverzichtelijke aanbod van sensoren en actuatoren op internet en de uiteenlopende en vaak verwarrende manieren van aansturen vormen een obstakel voor educatief gebruik. *DeviceLib* is bedoeld om daar verandering in te brengen. Waar de bestaande libraries voornamelijk efficiëntie nastreven en kiezen voor optimale aansturing, waarborgt *DeviceLib* in de eerste plaats een universele benadering van de devices. Bovendien biedt *DeviceLib* op de raspberry dezelfe opbouw van een programma als die van de arduino-ide, zodat programma's zonder aanpassing voor beide boards kunnen worden gebruikt.

Behalve het naslagwerk van de classes in *DeviceLib*, schenkt deze gids verder aandacht aan de programmeeromgeving en aan elektronische schakelingen. Bij iedere class wordt een aansluitschema afgebeeld van het bijbehorende device met daarbij een voorbeeldprogramma.

Inhoud

De programmeeromgeving



Voor de arduino en de raspberry gebruik je andere programmeeromgevingen. Deze gids veronderstelt dat je de arduino programmeert met de *arduino-ide* en de raspberry met *geany*. De *arduino-ide* installeer je op een pc (of je gebruikt de online versie) en *geany* is voorgeïnstalleerd wanneer je het besturingssysteem op de raspberry zet.

Arduino: Je programmeert op de pc met de *arduino-ide*. Daar compileer je het programma om het vervolgens naar het arduino-board te uploaden. Het programma start automatisch.

Raspberry: Je programmeert op de raspberry met *geany*. Daar compileer je en bouw je het programma. Het programma start je vanuit *geany*. Wie handig is met de raspberry weet nog andere manieren om het programmate bouwen en te starten.

Programmeeromgeving voor de arduino

1. Download het bestand *DeviceLib Arduino.zip* en pak het uit. Er verschijnt een map *DeviceLib*.
2. Verplaats de map *DeviceLib* naar *Documenten\Arduino\libraries*.
3. Start de *Arduino IDE*.

Programmeeromgeving voor de raspberry

NB. Voor het goed functioneren van de *DeviceLib* library moeten *gtk*, *opencv*, *espeak* en *mysqlclient* worden geïnstalleerd. Dit wordt behandeld in het deel over het voorbereiden van de raspberry. De programmeeromgeving bereid je als volgt voor:

1. Download het bestand *DeviceLib_BCM.tar* of *DeviceLib_SYS.tar* en pak het uit. Er verschijnen twee mappen: *home_pi* en *usr*.
2. Open de commando shell.
3. Type het commando: *sudo pcmanfm*. De verkenner opent met superuser-rechten.
4. Kopiëer de map *home_pi* naar de home-map (meestal */home/pi/*).
5. Kopiëer de map *usr* naar basis-map */*. Kies ervoor om de inhoud te *overschrijven*. Dit zal overigens niet gebeuren, omdat er alleen nieuwe bestanden worden toegevoegd.
6. Sluit de verkenner.
7. Type het commando: *sudo chmod 777 ~/Projecten*.
8. Type het commando: *sudo chmod 777 ~/Projecten/OnzeApp*.
9. Type het commando: *sudo ldconfig*.
10. Type het commando: *sudo nano /boot/config.txt*.
11. Wijzing de regel met *dtoverlay=w1-gpio* als volgt of voeg de regel toe:
    dtoverlay=w1-gpio,gpiopin=17
12. Druk eerst op *Ctrl-O* en *Enter* om op te slaan en daarna op *Ctrl-X* om af te sluiten.
13. Sluit de command shell.

Om ervoor te zorgen dat *geany* een programma op de juiste manier compileert, bouwt en start moeten enkele procedures van deze programmeeromgeving worden aangepast.

1. Kies in het menu *Bouwen >> Bouwcommando's instellen*. Daarmee open je een scherm met *Project instellingen.*
2. Wijzig het item *Compileren* als volgt:
   g++ -Wall -D RPI `pkg-config --cflags gtk+-3.0` -I/usr/include/DeviceLib -c "%f" -o "%e.o"
3. Wijzig het item *Build* als volgt:
   g++ -Wall "%e.o" -o "%e" -lbcm2835 -lDeviceLib -lmysqlclient -lpthread
   `pkg-config --cflags --libs opencv` `pkg-config --libs gtk+-3.0`
4. Wijzig het item *Execute* als volgt:
   clear; sudo "./%e"
5. Vergeet niet de aanpassingen op te slaan door op *OK* te clicken.

Gebruik het bestand *OnzeApp.geany* en de map *OnzeApp* als sjabloon voor je programma's. Je kunt ze kopiëren en *OnzeApp* voor de naam van je eigen programma vervangen. Open het bestand *OnzeApp.geany* in kladbok en vervang ook daar alle woorden *OnzeApp* met de naam van je programma.

Programmeren met *DeviceLib*

De basis van een programma

Een leeg programma ziet er als volgt uit:

Arduino:

```
void setup()
{
}

void loop()
{
}
```

Raspberry:

```
#include <Arduino.h>

void setup()
{
}

void loop()
{
}
```

Eventueel mag een programma voor de arduino ook beginnen met `#include <Arduino.h>`. Wanneer op de raspberry met de scherm-module wordt gewerkt, moet de eerste regel worden vervangen door `#include <ArduinoGtk.h>`.

De routine *setup* wordt éénmaal aan het begin van het programma uitgevoerd. De routine *loop* wordt eindeloos herhaald.

Een eenvoudig programma dat een led laat knipperen ziet er als volgt uit:

Arduino:

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

Led led;

void setup()
{
  led.setPin( 11);
  led.setBlink( 500, 500);
  led.setOn();
}

void loop()
{
}
```

Raspberry:

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

Led led;

void setup()
{
  led.setPin( 11);
  led.setBlink( 500, 500);
  led.setOn();
}

void loop()
{
}
```

Beide programma's zijn volledig identiek en hebben hetzelfde resultaat. De instructie `led.setBlink( 500, 500)` stelt de led in op knipperen, maar er gebeurt nog niets. Pas bij de instructie `led.setOn()` begint de led daadwerkelijk te knipperen.

Devices aan je programma toevoegen

Om een device in een programma te kunnen aansturen, moet het eerst aan een programma bekend worden gemaakt. Daartoe moet de betreffende class uit *DeviceLib* aan je programma worden toegevoegd met een *include* instructie. Hier doet zich echter het probleem voor dat vergelijkbare devices (bijv. stappenmotoren) elektronisch net even anders moeten worden aangestuurd, maar verder wel dezelfde functionaliteit bezitten (zoals vooruit of achteruit draaien, met een bepaalde snelheid draaien, enz). In zo'n geval biedt *DeviceLib* wel dezelfde class aan (bijv. *Stepper*), maar via verschillende *include* instructies (zoals `#include <a4988.h>` en `#include <unistepper.h>`). Om in een programma verwarring te voorkomen, is dan een extra *using namespace* instructie nodig (zoals `using namespace A4884` of `using namespace UniStepper`).

Motor met A4988-driver:

```
#include <Arduino.h>
#include <a4988.h>
using namespace A4944;

Stepper mtr;

void setup()
{
  mtr.setPin( 10, 11, 12);
  mtr.init();
  mtr.turn( 360);
}

void loop()
{
}
```

Unipolar stappenmotor:

```
#include <Arduino.h>
#include <unistepper.h>
using namespace UniStepper;

Stepper mtr;

void setup()
{
  mtr.setPin( 10, 11, 12, 13);
  mtr.init();
  mtr.turn( 360);
}

void loop()
{
}
```

Beide programma's hebben hetzelfde resultaat. De stappenmotor zal één rondje draaien: 360°.

De *namespace* maakt het mogelijk om beide soorten stappenmotoren in één programma aan te sturen:

```
#include <Arduino.h>
#include <a4988.h>
#include <unistepper.h>
using namespace A4944;
using namespace UniStepper;

A4944::Stepper mtr1;
UniStepper::Stepper mtr2;

void setup()
{
  mtr1.setPin( 10, 11, 12);
  mtr2.setPin( 10, 11, 12, 13);
  mtr1.init();
  mtr2.init();
  mtr1.turn( 360);
  mtr2.turn( 360);
}

void loop()
{
}
```

De algemene opzet van *DeviceLib*

Zoals in de inleiding al werd gesteld, moet de library *DeviceLib* eenheid brengen in het aansturen van actuatoren en sensoren. Iedere device heeft zijn eigen class, maar een aantal routines vind je in alle classes terug. Uiteraard verschillen daarbij de classes voor sensoren en actuatoren.

Sensoren	Actuatoren	Functie
`setPin`	`setPin`	Geef door op welke pinnen het device is aangesloten.
`init`	`init`	Voor sommige devices is initialiseren nodig.
`setXxxx`	`setXxxx`	Optionele instellingen van het device. Deze verschillen per device.
`read`		Haalt sensor-data van het device.
`dataReady`		Geeft terug of er nieuwe sensor-data aanwezig is.
`[eenheid]`, bijvoorbeeld `cm`		Geeft de sensorwaarde in de betreffende eenheid.
	`setXxxx`	Kiest een waarde (bijv. snelheid) voor een actuator, `m`eestal als een percentage van 'volledig aan'.
	`setOn`	Zet een actuator met de vooringestelde waarden aan.
	`setOff`	Zet een actuator uit.
	`isOn`	Geeft terug of een actuator 'aan' dan wel 'uit' staat.

In de beschrijving van de classes voor de sensoren en actuatoren staan aanwijzingen welke routines in de *setup* moeten worden aangeroepen. De *setup* wordt één keer aan het begin van het programma uitgevoerd. Daarna wordt de code in de *loop* steeds herhaald.

De meeste classes in *DeviceLib* zijn afgeleid van ofwel de class *Actuator* dan we de class *Sensor*. Beiden zijn op hun beurt afgeleid van de basis class *Device*. De class *Device* biedt de mogelijkheid om delen van een programma in zogenaamde *events* af te handelen. Een *event* komt in actie na het verstrijken van een bepaalde tijd.

Interfaces

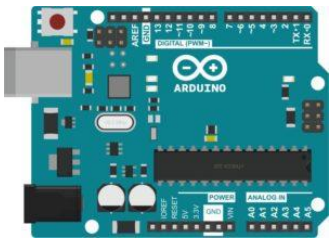
De arduino en raspberry hebben ieder hun eigen indeling met pinnen voor de aansluiting van devices. Bij een arduino staan de pin-nummers op de printplaat gedrukt. Dit is niet het geval bij een raspberry en bovendien zijn er bij de raspberry verschillende soorten nummering in omloop. De hier gebruikte nummering volgt de *GPIO*-nummering. Een aantal pinnen hebben een speciale functie, omdat ze voor een interface worden gebruikt.

De meeste boards bieden vier interfaces aan: *uart*, *i2c*, *spi* (*twi*) en *w1* (*onewire*). Sensoren of actuatoren die van deze interfaces gebruik maken, moeten worden aangesloten op de daarvoor gereserveerde pinnen. De tabel hieronder toont welke pinnen dat zijn. In de aansluitschema's bij de beschrijving van de classes worden de betreffende pinnen niet met een pinnummer maar met hun naam uit de tabel aangeduid. Bij het programmeren hoeft hier verder niet op te worden gelet.

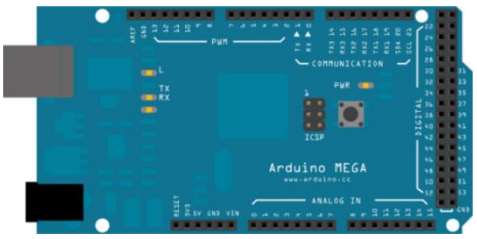
NB. Op de raspberry zit standaard de pin voor *onewire* (*GPIO-4*) tussen de *i2c*-pinnen en een *GND*-pin. Met de installatie als hiervoor beschreven (zie regel 12 onder het kopje *Programmeeromgeving voor de raspberry*), wordt de *onewire*-pin gewijzigd in *GPIO-17*. Dit is gedaan om *i2c*-apparaten met één enkele connector aan te kunnen sluiten. *DeviceLib* verwacht deze pin dan ook op *GPIO-17*.

BOARD	UART			I2C/TWI			W1		SPI			
	RX receive	TX transmit		SDA data	SCL clock		one wire		MOSI slave in	MISO slave out	SCK clock	CS/SS select
Arduino UNO	0	1		A4	A5		soft		11	12	13	10
Arduino MEGA	0	1		20	21		soft		51	50	52	53
Raspberry	gpio15	gpio14		gpio2	gpio3		gpio17		gpio10	gpio9	gpio11	gpio8

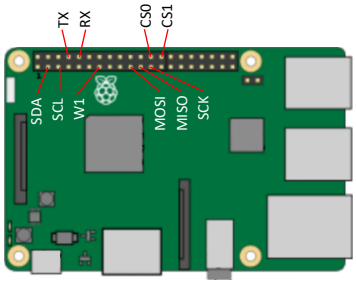
Arduino UNO



Arduino MEGA



Raspberry



DeviceLib referentiegids

Deze referentiegids beschrijft alle classes in de library *DeviceLib*. Bij de classes voor de actuatoren en sensoren wordt een programmavoorbeeld en het bijbehorende aansluitschema getoond.

De gids bestaat uit twee delen.

- In het eerste deel worden de classes en hun routines (programmeerbare taken) beschreven. Ze staan gegroepeerd naar hun functionaliteit. Achter de routinenaam staan tussen haakjes de eventuele parameters die moeten worden meegegeven. Een parameter in een kleiner lettertype mag worden weggelaten. In dat geval wordt de standaardwaarde gebruikt, die achter de parameter met '=' wordt aangegeven. Als een routine een waarde terug geeft, dan wordt dit met een '=' vóór de routinenaam aangegeven. In de beschrijving staat dan wat de routine terug geeft. De beschrijving van een routine begint soms met het woordje *setup*. In dat geval wordt de routine eenmalig aan het begin van een programma gebruikt (in de *setup*-routine) om de aansluiting van een apparaat te configureren.

- In het tweede deel staan de classes en bijbehorende routines alfabetisch gerangschikt. Terwijl het eerste deel beschrijvend is, worden in dit tweede deel alleen de data types van de parameters en de return waardes vermeld. In wezen is dit een lijst met de declaraties van de routines.

Device

De class *Device* biedt via inheritance voor alle classes de mogelijkheid om taken via een timer herhaald uit te voeren. Hiervoor wordt aan het begin van een programma eenmalig *Device::begin()* aangeroepen en vervolgens per afgeleide class de routine *startTimerEvent*. Met *startTimerEvent* wordt de routine opgegeven die het apparaat bedient. Zo'n routine wordt de timer event genoemd. Via een timer event kunnen opdrachten herhaald worden uitgevoerd, zonder dat daarmee de hoofdroutine wordt belast. Bij de classes *Actuator* en *Sensor* wordt een speciale toepassing hiervan beschreven.

Dit is wat er op de achtergrond gebeurt: De timer wordt gestart met de aanroep van de routine *Device::begin()* en eventueel weer gestopt met *Device::end()*. Dit zijn zogenaamde *static* routines. De timer loopt iedere milliseconde af en controleert van alle afgeleide classes de verstreken tijd om eventueel hun routine *onTimerEvent* [1] aan te roepen. Vervolgens voert *onTimerEvent* op zijn beurt weer de bij *startTimerEvent* opgegeven event routine uit. De routine *startTimerEvent* start dit proces voor een apparaat en de routine *stopTimerEvent* stop het weer.

startTimerEvent(msec, event)

Geef bij *event* de routine op, die aan de timer events moet worden toegevoegd. Deze routine zal automatisch om de zoveel tijd worden uitgevoerd. Geef bij *msec* op om de hoeveel milliseconden de routine moet worden uitgevoerd.

stopTimerEvent()

Verwijdert de event routine van *startTimerEvent* uit de timer events. De routine zal niet langer automatisch worden opgeroepen.

= millisTimer()

Geeft de tijd in milliseconden vanaf het moment dat de event-timer werd gestart.

[1] Een programmeur die een eigen class rechtstreeks van *Device* dan wel via *Actuator* of *Sensor* afleidt, kan de routine *onEventTimer* overloaden.

Actuator

Alle actuator-classes zijn afgeleid van de class *Actuator*. Daarmee wordt een standaard aangebracht in het activeren (*setOn*) en stoppen (*setOff*) van een actuator.
Eventueel kunnen de routines *setOn* en *setOff* in een timer event worden gebruikt. Dit kan handig zijn als het aan en uit zetten elkaar op regelmatige wijze afwisselen. Bijvoorbeeld maakt de routine *setBlink* van de class *Led* hiervan gebruik. Vervolgens kan met de routine *isOn* in de hoofdroutine worden gecheckt of een apparaat op een bepaald moment aan dan wel uit is.

setOn()

Activeert het apparaat.

setOff()

Stopt het apparaat.

= isOn()

Geeft *true* terug als het apparaat actief is en anders *false*. Gebruik *isOn* in de hoofdroutine als *setOn* of *setOff* in een timer event routine wordt aangeroepen.

Sensor

Alle sensor-classes zijn afgeleid van de class *Sensor*. Daarmee wordt een standaard aangebracht in hoe de waardes van de sensoren worden uitgelezen. Roep eerst de routine *read* aan om de actuele meetwaarde van een sensor op te vragen. Roep vervolgens een sensor-afhankelijke routine aan, die deze waarde in een specifieke eenheid terug geeft (zoals de routines *celcius* en *fahrenheit* van de class *Temperature*).
Eventueel kan de routine *read* in een timer event worden gebruikt. Dit kan handig zijn om een sensor op regelmatige tijden uit te lezen. Bijvoorbeeld maakt de routine *continueRead* van de class *Ibutton* hiervan gebruik. In de hoofdroutine kan dan met de routine *dataReady* worden gecheckt of er data van de sensor beschikbaar is.

read()

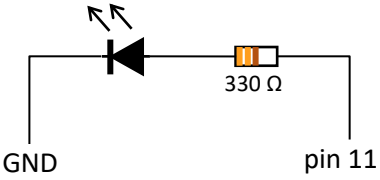
Vraagt een waarde van de sensor op.

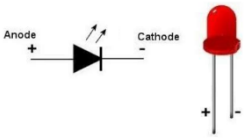
= dataReady()

Geeft *true* terug als er opnieuw een waarde van de sensor is ingelezen. Geeft *false* terug als dat niet het geval is. Gebruik *dataReady* in de hoofdroutine als *read* in een timer event routine wordt aangeroepen.

Led

Een led sluit je met twee draden aan: één aan een pin en één aan de GND (ground, 0 V). Zorg ervoor dat tussen de pin en de led een weerstand van 330 Ω wordt aangesloten. Zonder weerstand gaat er teveel stroom lopen en gaan de led en de pin stuk. Om bij een arduino de helderheid in te kunnen stellen, moet je de led op een pwm-pin aansluiten. Bij een raspberry maakt het niet uit, omdat de pwm softwarematig wordt geregeld. De kwaliteit is daardoor wel minder (soms geknipper).






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

Led led;

void setup()
{
  led.setPin( 11);
  led.setBlink( 300, 200);
}

void loop()
{
  led.setOn();
  delay( 2000);
  led.setOff()
  delay( 2000);
}
```

#include <led.h>

setPin(pin, pwm = false)

Setup. Geef het pin-nummer door, waarop de led is aangesloten. Geef bij 'pwm' (pulse-width-modulation) de waarde *true* door om de helderheid in te kunnen stellen. Geef bij 'pwm' de waarde *false* door als je de led alleen aan of uit hoeft te zetten.

setInversion(inversion = true)

Setup. De standaardinstelling van de class *Led* gaat ervanuit dat een led aan gaat als de pin 'hoog' wordt en uit als de pin 'laag' wordt. Dit is zo wanneer de led tussen de pin and de GND wordt aangesloten. Dit werkt net andersom wanneer de led tussen de pin en VCC wordt aangesloten. In dat geval moet de routine *setInversion* worden aangeroepen.

setBlink (on = NOBLINK, off = NOBLINK)

Laat de led knipperen wanneer *setOn* wordt gebruikt. Geef de tijd in milliseconden op, dat de led aan en uit moet zijn. Gebruik *setBlink* zonder parameters om het knipperen uit te zetten.

setBrightness(on, off = 0)

Werkt alleen als de led op een pwm-pin is aangesloten (zie *setPin*). Stelt de helderheid in. Geef de helderheid in procenten op. Gebruik eventueel het woord BRIGHT in plaats van 100%.

Op de Raspberry kan *setBrightness* samenwerken met de routine *setBlink*. De led knippert dan met twee verschillende helderheden. Geef bij parameter *off* een waarde voor de 'uit'-stand door. Dit kan niet op de arduino.

setOn()

Hiermee gaat de led aan of (met *setBlink*) gaat knipperen.

setOff()

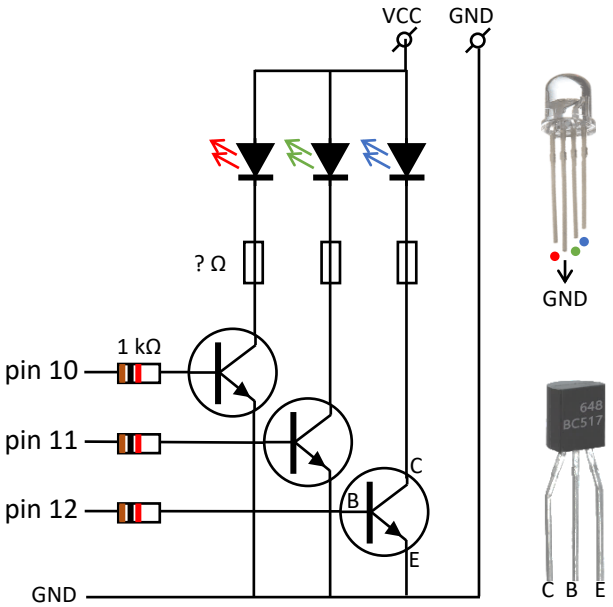
Hiermee gaat de led uit.

= isOn()

Geeft terug of de led aan is.

RgbLed

Er zijn diverse types op de markt. De meeste RgbLeds hebben een speciale aansluiting nodig. De RgbLed class gaat ervan uit dat de kleuren rood, groen en blauw ieder op een eigen pin worden aangesloten. Deze kleuren worden afzonderlijk als een rode led, groene led en blauwe led beschouwd. Zie verder de beschrijving van de class *Led*.



#include <rgbled.h>

setPin(red, green, blue, pwm = true)

Setup. Geef de pin-nummers door, zoals de kleuren van de led zijn aangesloten. Vul bij 'pwm' (pulse-width-modulation) *true* in om met *setColor(red, green, blue)* alle kleuren in te kunnen stellen. Wanneer de pinnen geen pwm ondersteunen kun je met *setColor(color)* alleen de standaard-kleuren instellen. Vul dan *false* in bij 'pwm'.

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

RgbLed led;

void setup()
{
  led.setPin( 10, 11, 12, false);
  led.setOn();
}

void loop()
{
  led.setColor( led.Red);
  delay( 500);
  led.setColor( led.Yellow);
  delay( 500);
}
```

setInversion(inversion = false)

Setup. Er bestaan twee type kleurenleds of ledstrips. Bij de ene hebben de interne leds een gemeenschappelijke VCC (wel *inversie*) en bij de andere een gemeenschappelijke GND (geen *inversie*). De standaardinstelling van de class *RgbLed* gaat uit van een gemeenschappelijke VCC. Leds met een gemeenschappelijke GND reageren precies omgekeerd bij deze instelling. Bij *setOn* gaat de led uit en bij *setOff* licht deze wit op. Voor het correcte resultaat moet bij dit type eerst *setInversion* worden aangeroepen.

setColor(color)

Kiest de kleur voor de rgb-led. Gebruik de vooringestelde kleuren zoals RgbLed::Red of RgbLed::Green om de kleur in te stellen. Kies uit de kleuren Red, Green, Blue, Magenta, Yellow, Cyan en White. De kleur wordt direct verwerkt en de rgb-led gaat aan.

setColor(red, green, blue)

Kiest de kleur voor de rgb-led door de helderheid van rood, groen en blauw in te stellen. De helderheid wordt met een percentage opgegeven. 0% is volledig uit. 100% is volledig aan. De kleur wordt direct verwerkt en de rgb-led gaat aan.

setOn()

Hiermee gaat de rgb-led aan of (met *setBlink*) gaat knipperen.

setOff()

Hiermee gaat de rgb-led uit.

= isOn()

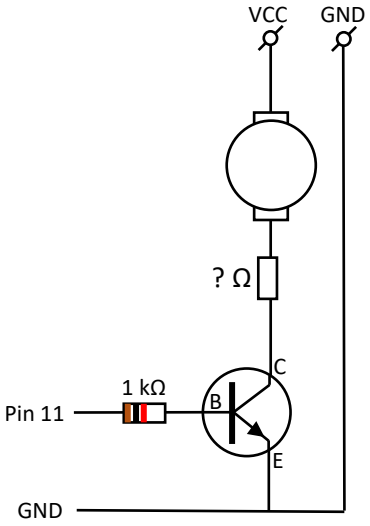
Geeft terug of de led aan is.

= redLed()
= greenLed()
= blueLed()

Deze drie routines geven de kleuren rood, groen en blauw terug als afzonderlijke led. Technisch gesproken geven de routines voor de betreffende kleur een pointer naar de class *Led* terug.

DcMotor

Er zijn diverse types op de markt. Je mag een motor nooit rechtstreeks op de pinnen aansluiten. Ook al gebruik je een 5V motortje dat minder dan 40 mA neemt, bij het aan- en uitschakelen kan er even veel stroom worden getrokken. Gebruik bijvoorbeeld een darlington transistor (BC517) of een chip (ULN2003) als buffer.



#include <dcmotor.h>

setPin(pin, pwm = true)

Setup. Geef het pin-nummer door, waarop de motor is aangesloten. Om een motor met *setSpeed* harder of zachter te laten draaien, moet je hem op een pin aansluiten die pulse-width-modulation (pwm) ondersteunt. Vul dan *true* in bij 'pwm'. Anders kan de motor alleen aan of uit worden geschakeld.

setSpeed(speed)

Werkt alleen als de motor op een pwm-pin is aangesloten (zie *setPin*). Stelt de snelheid in van 0% tot 100%.

setOn()

Hiermee gaat de dc-motor aan.

setOff()

Hiermee gaat de dc-motor uit.

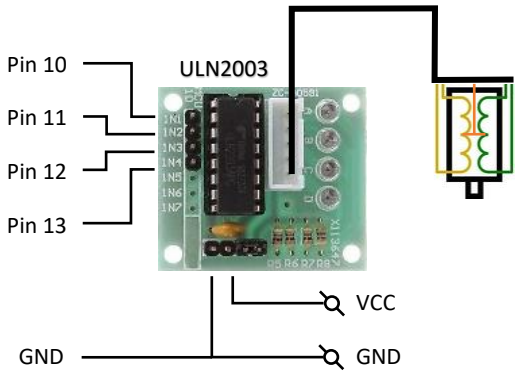
= isOn()

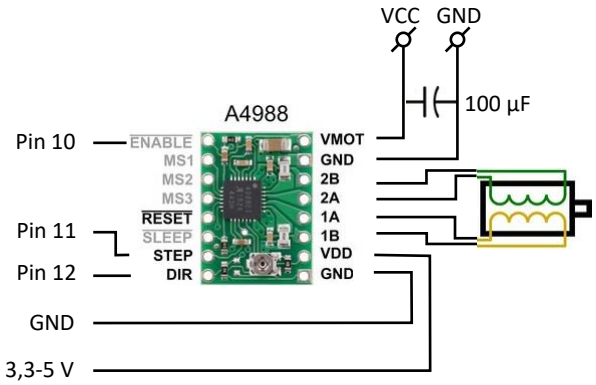
Geeft terug of de motor aan is.

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

DcMotor mtr;
int spd = 0;
bool acc = true;

void setup()
{
  mtr.setPin( 11);
  mtr.setOn();
}

void loop()
{
  if ( acc) spd = spd + 10;
  else spd = spd - 10;

  if ( spd < 0 ) {
    spd = 0;
    acc = true;
  }
  if ( spd > 100 ) {
    spd = 100;
    acc = false;
  }

  mtr.setSpeed( spd);
  delay( 100);
}
```

Stepper

Er zijn twee veel gebruikte types op de markt: unipolaire en bipolaire stappen-motoren. Gebruik een darlington-array (ULN2003) voor de aandrijving van een unipolaire stappenmotor. De aansturing wordt met vier pinnen geregeld. Gebruik een driver-board (A4988, DRV8825, enz.) om een bipolaire stappenmotor aan te sluiten. De meeste driver-boards voor bipolaire motoren worden met twee pinnen aangestuurd: één voor het draaien (step) en één voor de draairichting (dir).






```
#include <unistepper.h>
using namespace UniStepper;
```

```
#include <a4988.h>
using namespace A4988;
```

init(stepsPerRotation = 200)

Setup. Geef hier op hoeveel stappen de motor bij een volledige omwenteling maakt.

setPin(pin1, pin2, pin3, pin4) *

Setup. Geef de pin-nummers door, zoals de ingangen *in1*, *in2*, *in3* en *in4* van de stappenmotor zijn aangesloten.

setPin(step, direction, enable, reset) *

Setup. Geef de pin-nummers door, zoals de ingangen *step*, *dir*, *enab* en *reset* van het driver-board zijn aangesloten.

* Alleen geschikt voor gebruik met een *A4988* driver of een compatible type.

```
#include <Arduino.h>
#include <a4988.h>
using namespace A4988;

Stepper mtr;

void setup()
{
  mtr.setPin( 11, 12, 10);
  mtr.init();
}

void loop()
{
  mtr.start( SC_ROTATE, 360);

  // wacht tot de motor klaar is
  while ( mtr.isOn() );

  delay( 100);
}
```

setForward(forward = true)

Stelt de richting van de motor in op vooruit als je *true* doorgeeft en op achteruit als je *false* doorgeeft.

setReverse()

Stelt de richting van de motor in op achteruit.

setHold(hold = true)

Laat de motor zijn positie vasthouden als je *true* doorgeeft en geeft de motor vrije loop als je *false* doorgeeft. Pas op! Sommige motoren branden door als je ze lang hun positie laat vasthouden.

setRelease()

Geeft de motor vrije loop.

turn(steps, speed = 100)
rotate(angle, speed = 100)
move(msec, speed = 100)
go(speed = 100) *

Laat de motor respectievelijk een aantal stappen, graden of milliseconden draaien. Geef de snelheid in procenten op. Als je geen snelheid invult, kiest de routine standaard 100%.

Deze routines komen in de plaats van de *setOn* (zoals je in de andere classes van *DeviceLib* gewend bent) en starten de motor meteen.

* Pas op met de routine *go*, omdat deze het programma kan blokkeren. De routine kan alleen met *setOff* worden beëindigd vanuit een *andere thread* of vanuit een *interrupt routine*!

setOff()

Hiermee gaat de stappenmotor uit. Overigens gaat de stappenmotor vanzelf uit na een *turn*, *move* of *rotate* instructie. De routine *setOff* is daarom alleen zinvol, wanneer deze vanuit een andere *thread* of vanuit een *timer interrupt* wordt aangeroepen om de *start* routine te onderbreken.

start(command, parameter, speed = 100) *

Wanneer de routines *turn*, *rotate*, *move* of *go* worden gebruikt, zal het programma pas verder gaan als de motor klaar is of als hij vanuit een *interrupt routine* of een *andere thread* wordt gestopt. Maar na het aanroepen van de *start* routine wacht het programma niet tot de motor klaar is en gaat verder. Ondertussen blijft de motor draaien tot hij klaar is. Geeft als commando de waardes *TURN*, *ROTATE*, *MOVE* of *GO* door om de betreffende actie te starten. Bij *parameter* geeft je de bijpassende parameter voor deze routines door, respectievelijk de *steps*, *angle* of de tijd in *msec*. Geef de snelheid in procenten op.

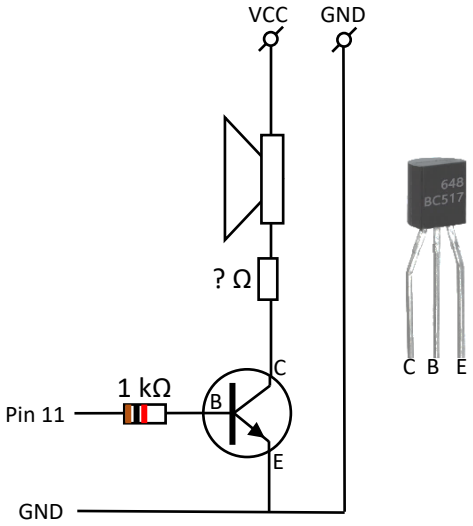
* Alleen beschikbaar voor de raspberry.

= isOn()

Geeft terug of de stappenmotor bezig is.

Buzzer

Een buzzer is een eenvoudig apparaatje dat zoemt als je er stroom door laat gaan. Je mag een buzzer nooit rechtstreeks op de pinnen aansluiten. Ook al gebruik je een 5V buzzer die minder dan 40 mA neemt, bij het aan- en uitschakelen kan er even veel stroom worden getrokken. Gebruik bijvoorbeeld een darlington transistor (BC517) als buffer.



#include <buzzer.h>

setPin(pin)

Setup. Geef het pin-nummer door, waarop de buzzer is aangesloten

setOn()

Hiermee gaat de buzzer aan.

setOff()

Hiermee gaat de buzzer uit.

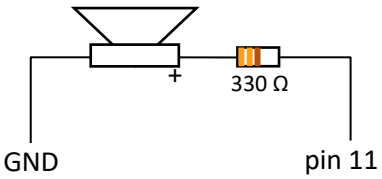
= isOn()

Geeft terug of de buzzer aan is.

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

Buzzer buz;

void setup()
{
  buz.setPin( 11, false);
  buz.setBlink( 300, 200);
}

void loop()
{
  buz.setOn();
  delay( 1000);
  led.setOff()
  buz  ( 4000);
}
```

Beeper

Een beeper sluit je met twee draden aan: met de plus-draad aan een pin en de andere aan de GND (ground, 0 V). Zorg ervoor dat tussen de pin en de beeper een weerstand van 330 Ω wordt aangesloten. Zonder weerstand gaat er teveel stroom lopen en gaan de beeper en de pin stuk. De class *Beeper* biedt de mogelijkheid om een tonenlijst te maken en af te spelen.



De tonen van tien toonladders (C0 tot en met B9) zijn voorgedefinieerd als *PIANO_C0*, *PIANO_C0s*, *PIANO_B*, enz. In de definities wordt *C0#* (Cis), D0# (Dis), enz. geschreven als *PIANO_C0s* (C sharp) en *PIANO_D0s* (D sharp). Dit, omdat het #-teken een speciale betekenis heeft in C++.

Voor de toonduur kunnen de waardes *WHOLE_NOTE*, *HALF_NOTE*, *QUARTER_NOTE*, *EIGHTH_NOTE* en *SIXTEENTH_NOTE* worden gebruikt. Toonduur kan bij elkaar worden opgeteld, bijv. *QUARTER_NOTE+EIGHTH_NOTE* voor een anderhalve tel in kwart maat.

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

Beeper beep;

void setup()
{
  beep.setPin( 11);

  beep.add( 220, 500);
  beep.add( 880, 500);

}

void loop()
{
  beep.play( 5);
  delay( 2000);
}
```

#include <beeper.h>

setPin(pin)

Setup. Geef het pin-nummer door, waarop de beeper is aangesloten

setBeeper(herz, msec)

Bepaal met deze routine de toonhoogte (*herz*) en duur (*msec*) van de beep.

beep()

Laat de beep horen, zoals eerder met *setBeeper* werd ingesteld.

setOn()

Laat de toonhoogte, zoals eerder met *setBeeper* werd ingesteld, voor onbepaalde duur horen.

Let op! De routine *setOn* blokkeert het programma totdat *setOff* vanuit een interrupt of timer event wordt aangeroepen. Gebruik de routine *beep* om een korte beep te laten horen.

= addTone(herz, msec)

Voeg een toon toe aan het einde van de tonenlijst. Parameter *herz* bepaalt de toonhoogte en parameter *msec* bepaalt de duur van de toon. Geeft de indexpositie van de toon in de lijst terug.

insertTone(index, herz, msec)

Voegt een toon tussen de andere tonen van de lijst op de bij *index* aangegeven positie. Parameter *herz* bepaalt de toonhoogte en parameter *msec* bepaalt de duur van de toon.

remove(index, count = 1)

Wist een *count* aantal tonen vanaf de betreffende *index*.

clear()

Wist de lijst met tonen.

play()

Speelt de lijst met tonen af.

Sound *

* Alleen beschikbaar voor de raspberry.

Deze class is gebaseerde op extra software die alleen voor de Raspberry beschikbaar is. Het betreft de programma's *aplay*, *omxplayer* en *espeak*. *Aplay* en *omxplayer* zijn standaard in *Raspbian Buster* opgenomen. *Espeak* moet speciaal worden geïnstalleerd. De class *Sound* werkt met de standaard uitvoerapparaten voor geluid op de Raspberry: de hoofdtelefoon-uitgang ('local' genoemd), de hdmi-uitgang en via bluetooth.

De class *Sound* kan een playlist afspelen of aparte play-opdrachten verwerken.

NB. De class *Sound* maakt bij uitzondering geen gebruik van de standaard routines *setOn* en *setOff*. Het zou niet duidelijk zijn welke manier van afspelen met deze routines wordt bedoeld.

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

Sound snd;

int n = 0;

void setup()
{
  snd.init( snd.Local);
}

void loop()
{
  delay( 1000);
  n++;

  String s( n);
  s += " seconden bezig";
  snd.playText( s);
}
```

#include <sound.h>

init(afspeelapparaat)
init(Sound::Bluetooth, device-adres)

Bepaal met welk afspeelapparaat een geluidbestand of tekst wordt afgespeeld. Dit kan zijn: *Sound::Local*, *Sound::HDMI* of *Sound::Bluetooth*. Wanneer *Bluetooth* als afspeelapparaat wordt gekozen, is het nodig om het device-adres van de speaker/koptelefoon op te geven. Deze vind je met het shell-commando *bluetoothctl* en dan *scan on*. Je gebruikt *bluetoothctl* tevens om de speaker/koptelefoon te pairen en te connecten (*pair <device-adres>*, *trust < device -adres>*, *connect < device -adres>*).

= addFile(geluidbestand)

Geef een geluidbestand door dat door *aplay* kan worden afgespeeld. Dit wordt als laatste in de playlist geplaatst. Geef het bestand als *String* of tussen "" op. De routine geeft het indexnummer in de playlist terug.

= addText(tekst)

Geef een tekst door in een taal die door *espeak* kan worden uitgesproken. Deze wordt als laatste in de playlist geplaatst. Geef de tekst als *String* of tussen "" op. De routine geeft het indexnummer in de playlist terug.

insertFile(index, geluidbestand)
insertText(index, tekst)

Gaat hetzelfde als *add*, maar plaatst het bestand of de tekst op de aangegeven index in de playlist.

remove(index)

Verwijdert het geluidbestand of de tekst op positie *index* uit de playlist.

clear()

Verwijdert alle geluidsbestanden en teksten uit de playlist.

playList()

Speelt de playlist af.

playFile(geluidbestand)

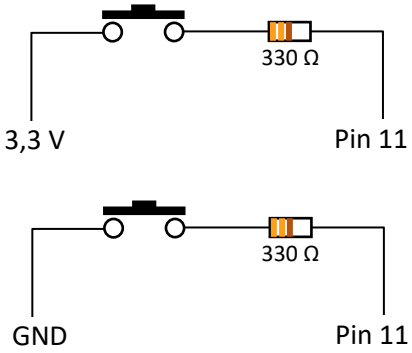
Speelt het geluid uit het geluidbestand af. Geef het bestand als *String* of tussen “” op.

playText(tekst)

Spreekt de opgegeven tekst uit. Geef de tekst als *String* of tussen “” op.

Switch

Een schakelaar sluit je bij de raspberry aan tussen een pin en de VCC (bovenste schakeling hiernaast) en bij de arduino tussen een pin en de GND (onderste schakeling hiernaast). Een raspberry en arduino verschillen namelijk in hoe de boards zijn gemaakt. Bij de arduino zijn de pinnen met een weerstand aan de VCC verbonden (pull-up) en bij een raspberry aan de GND (pull-down). Je mag de schakelaar nooit zonder weerstand aansluiten, want dan ontstaat er kortsluiting. De weerstand moet minimaal 330 Ω bedragen.



Let op! Omdat schakelaars bij het indrukken of loslaten meestal snel achter elkaar meerdere keren contact maken (dit heet *jitter*), is het niet gemakkelijk om je programma zo te maken dat er toch maar één keer wordt gereageerd.

```
#include <Arduino.h>
#include <switch.h>
#include <led.h>

Switch swi;
Led led;

void setup()
{
  swi.setPin( 11, swi.toVcc);
  led.setPin( 12);
}

void loop()
{
  swi.read();
  if ( swi.pressed() )
    led.setOn();
  else
    led.setOff();
}
```

#include <switch.h>

setPin(pin, connect)

Setup. Geef bij parameter *pin* het pin-nummer door waarop de schakelaar is aangesloten. Vul bij parameter *connect* in of de schakelaar aan de VCC is verbonden (*Switch::toVcc*) dan wel aan de GND (*Switch::toGnd*).

setNormalOpen(normalopen)

De meeste schakelaars zijn van het type ‘normal open’. Als je ze indrukt wordt de stroomkring gesloten. Sommige schakelaars werken net andersom (zoals die in een koelkast). De stroomkring blijft gesloten totdat je de schakelaar indrukt. Deze schakelaars zijn van het type ‘normal closed’. Vul *true* in voor normal-open en *false* voor normal-closed schakelaars.

triggerEvent(routine)

Geef hier de naam van een routine op, die zal worden aangeroepen als de schakelaar wordt ingedrukt of losgelaten. Declareer de routine als volgt: *void routinenaam()*. Let op: De schakelaar moet worden aangesloten op één van de speciale pinnen, die een IRQ (interrupt request) kan oproepen. Raadpleeg de beschrijving van de raspberry of de arduino.

read()

Hiermee wordt een stand van de schakelaar ingelezen.

= dataReady()

Geeft terug of er de stand opnieuw is ingelezen.

= pressed()

Nadat eerst *read* is aangeroepen, geeft deze routine *true* terug als de schakelaar is ingedrukt en anders *false*.

= released()

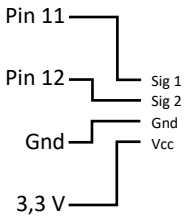
Nadat eerst *read* is aangeroepen, geeft deze routine *false* terug als de schakelaar is ingedrukt en anders *true*.

= changed()

Nadat eerst *read* is aangeroepen, geeft deze routine *true* terug als de stand van de schakelaar is gewijzigd en anders *false*.

Rotary

Een rotary-sensor geeft pulsen op de signaal-lijnen wanneer de as draait. Over het algemeen zijn er twee signaal-lijnen, zodat de sensor kan bepalen of de as vooruit dan wel achteruit draait.



#include <rotary.h>

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

Rotary rot;

void setup()
{
  rot.setPin( 11, 12);
}

void loop()
{
  rot.read();
  if ( rot.dataReady() ) {
    Serial.print( "Rpm: ");
    Serial.print( rot.rps());
  }
}
```

setPin(pin1, pin2)

Setup. Geef het pin-nummers door, waarop de schakelaar is aangesloten. Meestal worden de uitgangen van de rotary *signalen* genoemd. Je sluit dan signaal 1 op *pin1* aan en signaal 2 op *pin2*.

setTicksPerRotation(ticks)

Setup. Geef hier het aantal pulsen op dat de rotary-sensor bij één volledige omwenteling op een pin afgeeft.

setTimeout(msec)

Geef hier op hoeveel millisecondes er op een puls zal worden gewacht voordat de rotary vindt dat er geen beweging meer is.

swapDirection()

Na aanroep van deze routine zal de rotary *vooruit* en *achteruit* met elkaar verwisselen.

read()

Hiermee worden de signalen van de rotary ingelezen.

= dataReady()

Geeft terug of de signalen van de rotary opnieuw zijn ingelezen.

= moving()

Nadat eerst *read* is aangeroepen, geeft deze routine *true* terug als de rotary draait en anders *false*.

= forward()

Nadat eerst *read* is aangeroepen, geeft deze routine *true* terug als de rotary vooruit draait en anders *false*.

= backward()

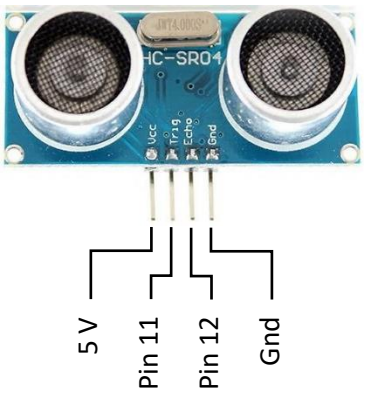
Nadat eerst *read* is aangeroepen, geeft deze routine *true* terug als de rotary achteruit draait en anders *false*.

= rps()
= rpm()

Nadat eerst *read* is aangeroepen, geven deze routines de rotatiesnelheid terug in de gewenste eenheid. De snelheid is een kommagetal. (*Rps* betekent ‘rotaties per seconde’ en *rpm* betekent ‘rotaties per minuut’.

Distance

Dit apparaat meet afstanden door middel van ultrasone geluiden. Pulsen op de echo-pin zenden een geluid uit dat tegen een voorwerp weerkaats. Daarna wordt het weerkaatste geluid opgevangen via de trigger-pin. De tijd die het geluid nodig had is bepalend voor de afstand. Er bestaan diverse types die eruit zien als het voorbeeld hiernaast en op dezelfde manier werken. (Zie verder bij de class *Average*).



#include <hcsr04.h>
using namespace HCSR04;

```
#include <Arduino.h>
#include <hcsr04.h>
using namespace HCSR04;

Distance dist;

void setup()
{
  dist.setPin( 11, 12);
}

void loop()
{
  dist.read();
  if ( dist.dataReady() ) {
    Serial.print( "Afstand: ");
    Serial.print( dist.cm());
    Serial.println( " cm");
  }
}
```

setPin(trigger, echo)

Setup. Geef de pin-nummers door, zoals de afstandsensor is aangesloten.

read()

Hiermee wordt een afstand ingelezen.

= dataReady()

Geeft terug of de afstand opnieuw is ingelezen.

= cm()

Nadat eerst *read* is aangeroepen, geeft deze routine de afstand in hele centimeters terug.

= inch()

Nadat eerst *read* is aangeroepen, geeft deze routine de afstand in hele inches terug.

Wanneer de class Distance wordt geïmporteerd, komt tegelijk de kleine hulp-class Average mee. Bij beweeglijke oppervlakken kan de ultrasone sensor sterk wisselende meetwaardes geven. Gebruik de class Average om deze waardes te stabiliseren.

Average

Samen met de class *Distance* krijg je ook de beschikking over de class *Average*. Gebruik deze class om de meetwaardes te stabiliseren. In de praktijk blijken afstandmetingen door diverse factoren te worden verstoord. De class *Average* neemt het gemiddelde van de laatste twintig metingen. Alle waardes worden als kommagetal verwerkt.

add(value)

Voegt een waarde toe op basis van 'first in, first out'. De waarde komt op de plaats van de eerst toegevoegde waarde uit een lijst van twintig.

clear()

Wist alle waardes uit de lijst.

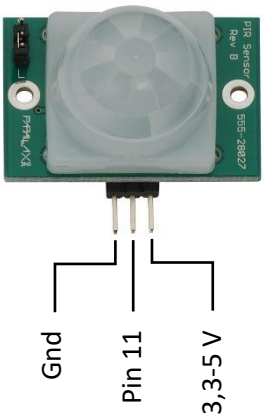
= avg()

Geeft het gemiddelde van de laatste twintig metingen terug.

```
#include <Arduino.h>
#include <hcsr04.h>
using namespace HCSR04;

Distance dist;
Avarage avg;

void setup()
{
  dist.setPin( 11, 12);
}

void loop()
{
  dist.read();
  if ( dist.dataReady() ) {
    avg.add( dist.cm());
    Serial.print( "Afstand: ");
    Serial.print( avg.avg());
    Serial.println( " cm");
  }
}
```

Motion

Deze class ondersteunt *PIR*-sensoren, beter bekend als bewegingsensoren. In de hobbywereld worden vooral de types *HCSR501* en de mini-uitvoering *HCSR505* gebruikt. De *HCSR505* kun je gebruiken alsof het een *HCSR501* is. Houd er rekening mee dat *PIR*-sensoren traag op beweging reageren. Bovendien geeft een sensor pas na meerdere seconden aan dat de beweging is gestopt.




```
#include <hcsr501.h>
using namespace HCSR501;
```

setPin(pin)

Setup. Geef de pin-nummers door, zoals de bewegingsensor is aangesloten.

read()

Hiermee wordt de sensor uitgelezen of er beweging is geregistreerd.

= dataReady()

Geeft terug of de sensor opnieuw is uitgelezen.

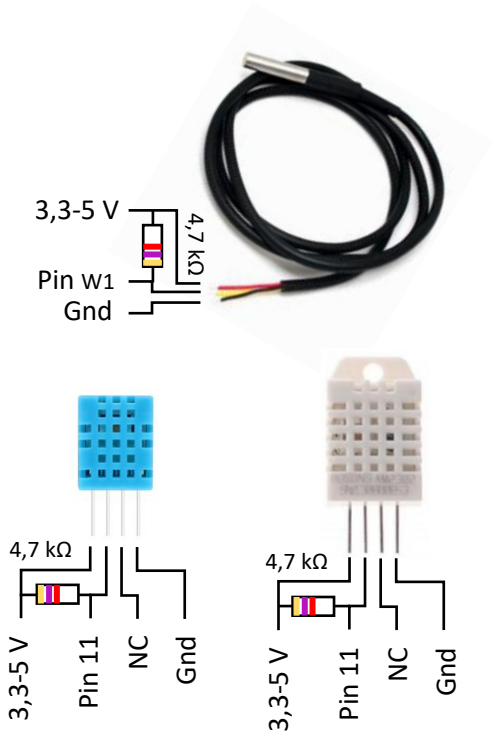
= detected()

Nadat eerst *read* is aangeroepen, geeft deze routine terug of er beweging is geregistreerd.

```
#include <Arduino.h>
#include <hcsr501.h>
using namespace HCSR501;

Motion mot;

void callback()
{
  Serial.println( "Beweging");
}

void setup()
{
  mot.setPin( 11);
  mot.triggerEvent( callback,
                    mot.Detected);
}

void loop()
{
}
```

Temperature

Dit apparaat meet de temperatuur. De class is geschikt voor de twee type sensoren: DS18 (zoals DS18S20 en DS18B20) en DHT (de DHT11 en DHT22). Met een DHT-sensor kan ook de luchtvochtigheid worden gemeten.




```
#include <dht.h>
using namespace DHT;

#include <ds18.h>
using namespace DS18;
```

= getIdList() *

Setup. Deze routine geeft een lijst met sensor-id's terug. Er mogen meerdere DS18-sensoren zijn aangesloten op dezelfde pin.

* Alleen beschikbaar wanneer een DS18-sensor is aangesloten.

setSensor(id, pin = 10)

Bepaalt van welke sensor de waarde wordt gelezen. Vul voor *id* in: DHT11, DHT22 of een sensor-id uit de lijst van *getIdList*. Wanneer er één DS18-sensor is aangesloten, kan een lege waarde ("") worden ingevuld. Op de raspberry is het niet nodig om een pinnummer door te geven voor een DS18-sensor. Automatisch wordt de standaardpin voor *W1* (*onewire*) gebruikt.

read()

Hiermee wordt de temperatuur ingelezen.
Met een DHT-sensor wordt ook de luchtvochtigheid ingelezen.

= dataReady()

Geeft terug of er de temperatuur opnieuw is ingelezen.

```
#include <Arduino.h>
#include <dht.h>
using namespace DHT;

Temperature tmp;

void setup()
{
  tmp.setSensor( DHT22, 11);
}

void loop()
{
  tmp.read();
  if ( tmp.dataReady() ) {
    Serial.print( "Temperatuur: ");
    Serial.print( tmp.celcius());
    Serial.println( " °C");
    Serial.print( "Vochtigheid: ");
    Serial.print( tmp.humidity());
    Serial.println( " %");
  }
}
```

= kelvin()
= celcius()
= fahrenheit()

Nadat eerst *read* is aangeroepen, geven deze routines de temperatuur terug in de gewenste eenheid. De temperatuur is een kommagetal. Arduino: Wanneer de sensor geen meetwaarde kon inlezen, geven de routines NAN terug.

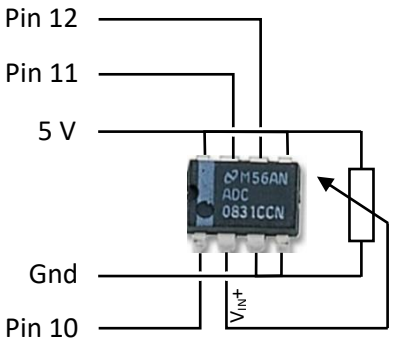
= humidity() *

Nadat eerst *read* is aangeroepen, geeft deze routine de relatieve luchtvochtigheid in procenten terug. De luchtvochtigheid is een kommagetal. Wanneer de sensor geen meetwaarde kon inlezen, geeft de routines NAN terug .

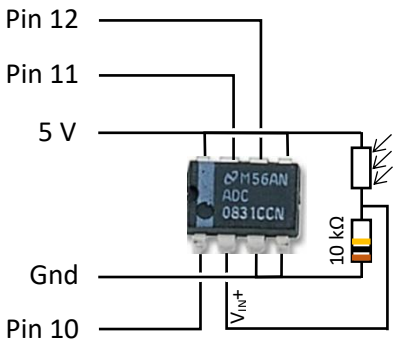
* Alleen beschikbaar wanneer een DHT-sensor is aangesloten.

ADConverter

Een ad-converter zet een analoge spanning om in een 8-bits binaire waarde. De output-waarde van de converter verloopt lineair met het voltage op de input-pin (V_{IN}+). De output-waarde wordt door de class ADConverter ingelezen en omgezet in een meetwaarde volgens de instellingen van *setInputRange* en *setOutputRange*.




```
#include <adc0831.h>
using namespace ADC0831;
```



setPin(cs, clk, output)

Setup. Geef met deze routine door, op welke pinnen de ad-converter is aangesloten.

setInputRange(min, max)

Setup. Stel met deze routine de minimale en maximale output-waarde in, die de converter door kan geven. Standaard staat het bereik ingesteld op 0 tot 255, wat overeenkomt het volledige bereik.

Aangezien de chip volgens de datasheet minimaal 4,5 V vraagt, zou de schakeling hiernaast niet met de raspberry werken. In de praktijk blijkt echter dat de chip bij 3,3 V nog steeds zijn lineaire karakterestiek behoudt, maar met de helft van het bereik: 0 tot 127. Door *setInputRange(0, 127)* te gebruiken werkt de schakelijk hiernaast toch ook met 3,3 V.

setOutputRange(min, max)

Setup. Stel met deze routine de minimale en maximale meetwaarde in, die overeenkomen met de maximale en minimale output-waarde van de converter. Standaard staat het bereik ingesteld op 0 tot 100, oftewel op procenten.

read()

Hiermee wordt de meetwaarde ingelezen.

```
#include <Arduino.h>
#include <adc0831.h>
using namespace ADC0831;

ADConverter adc;

void setup()
{
  adc.setPin( 10, 11, 12);
}

void loop()
{
  adc.read();
  if ( adc.dataReady() ) {
    Serial.print( "Waarde: ");
    Serial.println( adc.value());
  }
}
```

= dataReady()

Geeft terug of de meetwaarde opnieuw is ingelezen.

= value()

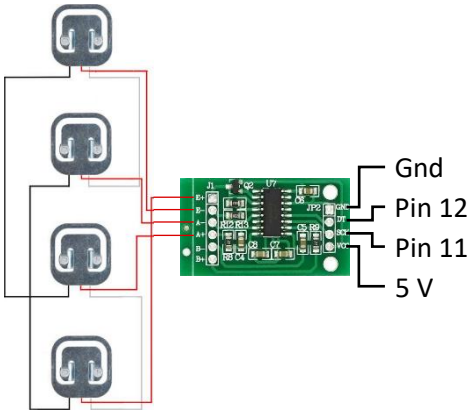
Nadat eerst *read* is aangeroepen, geeft deze routines de meetwaarde terug.

= rawValue()

Nadat eerst *read* is aangeroepen, geeft deze routine de onverwerkte output-waarde terug.

Scale

De class *Scale* gebruikt de Hx711 signaal-versterker en ad-converter om gewichtsensoren aan te sluiten. Meestal worden vier sensoren in een zogenaamde Wheatstone-schakeling geplaatst. Het gewicht wordt in kilogram teruggegeven.




```
#include <hx711.h>
using namespace HX711;
```

setPin(sck, dt)

Setup. Geef met deze routine door, op welke pinnen op het Hx711-boardje is aangesloten.

read()

Hiermee wordt het gewicht ingelezen.

= dataReady()

Geeft terug of er het gewicht opnieuw is ingelezen.

= kg()

Nadat eerst *read* is aangeroepen, geeft deze routine het gewicht in kilogram terug.

Gps

Deze class werkt samen met iedere gps-module die NMEA-zinnen via UART verzendt. De gps-module wordt op de TX- en RX-pin van de arduino of de raspberry aangesloten.

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

Gps gps;

void setup()
{
  gps.setSentences( gps.GPRMC);
}

void loop()
{
  gps.read();
  if ( gps.dataReady() ) {
    Serial.print( "Lon: ");
    Serial.println( gps.longitude());
    Serial.print( "Lat: ");
    Serial.println( gps.latitude());
    Serial.print( "Koers: ");
    Serial.println( gps.course());
    Serial.println( "");
  }
}
```

#include <gps.h>

setPin(rx, tx)

Setup. Geef de pinnen door waarop de gps-module is aangesloten. Alleen nodig als de uart-hardware niet wordt gebruikt.

setBaud(baud)

Geef de baudrate door voor de communicatie met de gps-module. Deze staat standaard op 115200 ingesteld.

setSentences(sentences)

Geef door welke NMEA-zinnen moeten worden gebruikt om de gegevens uit te halen. Dit kunnen zijn: *GPS::GPGGA*, *GPS::GPGLL*, *GPS::GPRMC*. Deze drie zinnen kunnen via 'or' worden gecombineerd, bijvoorbeeld 'GPS::GPGGA | GPS::GPRMC'.

read()

Hiermee worden de gegevens ingelezen.

= dataReady()

Geeft terug of er een nieuwe NMEA-zin is ingelezen.

= latitude()
= longitude()

Nadat eerst *read* is aangeroepen, geven deze routines de locatie gegevens als kommagetal in graden terug.

= altitude()

Nadat eerst *read* is aangeroepen, geeft deze routine de hoogte in meters terug. De hoogte is een kommagetal.

= mps()
= kph()

Nadat eerst *read* is aangeroepen, geven deze routines de snelheid terug in de gewenste eenheid. De snelheid is een kommagetal.

= course()

Nadat eerst *read* is aangeroepen, geeft deze routine de richting van de verplaatsing terug als azimut. De azimut is een kommagetal.

= timestamp()

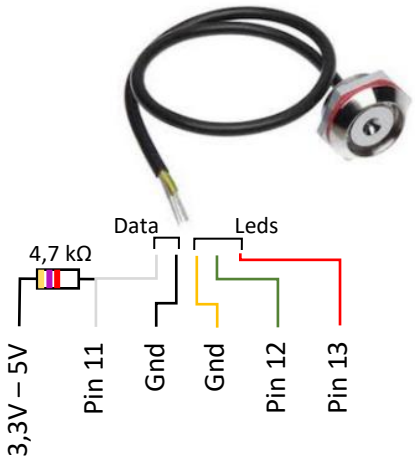
. Nadat eerst *read* is aangeroepen, geeft deze routine de timestamp van de meting terug. De timestamp is een tekst met het format “YYMMDDhhmmss”.

= messages()

Nadat eerst *read* is aangeroepen, geeft deze routine de voor de gegevens gebruikte NMEA-zinnen als tekst terug. De zinnen zijn met een new-line-teken gescheiden.

IButton

I-buttons, ook wel smart-buttons genaamd, worden gebruikt voor toegangcontrole. Ieder i-button heeft een uniek id. Over het algemeen hebben de button-readers een ledje in het midden, soms met meerdere kleuren. Het uitlezen van de i-button en de aansturing van de ledjes gaat via aparte pinnen. Gebruik de class *Led* om de ledjes aan te sturen.



#include <ibutton.h>

setPin(pin)

Setup. Geef het pin-nummer door, waarop de i-button is aangesloten.

setTimeout(msec)

Setup. Geef het milliseconden op voordat de lezer een poging tot het lezen van een button afbreekt.

read()

Hiermee wordt de id van een i-button uitgelezen.

= dataReady()

Geeft terug of er een andere i-button werd gepresenteerd.

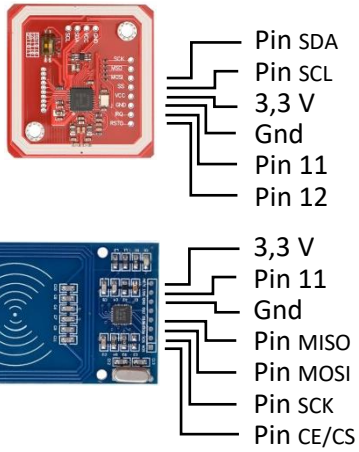
= tag(expireAfter = 0)

Nadat eerst *read* is aangeroepen, geeft deze routine de id van de i-button als tekst terug. Geef bij *expireAfter* door hoe lang (in milliseconden) na een *read* de id mag worden teruggeven. Wanneer de geldigheid is verstreken, geeft de routine de tekst 'No button' terug.

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

IButton ib;

void setup()
{
  ib.setPin( 11);
}

void loop()
{
  ib.read();
  if ( ib.dataReady() ) {
    Serial.print( "Kaart-id: ");
    Serial.println( ib.tag());
  }
}
```

Rfid

Rfid-tags worden onder andere gebruikt voor toegangscontrole. Ieder tag heeft een uniek id en veel tags kunnen informatie opslaan en terug geven. De *Rfid*-class ondersteund tags met een geheugen dat is ingedeeld in 64 blokken van 16 byte (totaal 1 kb). Dit is de standaard Mifare-indeling.




```
#include <pn532.h>
using namespace PN532;
```

```
#include <mfrc522>
using namespace MFRC522;
```

setPin(reset, irq = 0)

Setup. Geef het pin-nummer op waarmee de rfid-lezer kan worden gereset.

= init()

Setup. Deze routine moet direct na *setPin* worden aangeroepen om de rfid-scanner te activeren. De routine geeft terug of het activeren is gelukt.

setTimeout(msec)

Setup. Geef het milliseconden op voordat de rfid-lezer een poging tot het lezen van een kaart afbreekt.

reset()

Hiermee wordt de rfid-lezer gereset.

read()

Hiermee wordt de id van een tag uitgelezen.

done()

Deze routine moet na een succesvolle *read* worden aangeroepen (dus als *dataReady* waar is). Tussen een succesvolle *read* en het aanroepen van *done* kan eventueel het kaartgeheugen worden uitgelezen (*readValue*) of beschreven (*writeValue*).

```
#include <Arduino.h>
#include <pn532.h>
using namespace PN532;

Rfid rfid;

void setup()
{
  rfid.setPin( 12, 11);
  rfid.init();
  rfid.setTimeout( 500);
}

void loop()
{
  rfid.read();
  if ( rfid.dataReady() ) {
    Serial.print( "Kaart-id: ");
    Serial.println( rfid.tag());

    rfid.done();
  }
}
```

= dataReady()

Geeft terug of er een andere tag werd gepresenteerd.

= tag(expireAfter = 0)

Nadat eerst *read* is aangeroepen, geeft deze routine de id van de rfid-tag als *String* terug. Geef bij *expireAfter* op, hoe lang (in milliseconden) na een *read* de tag mag worden teruggeven. Wanneer de geldigheid is verstreken, geeft de routine de tekst “No button” terug.

setAuthentication(key)

Informatie op een rfid-kaart is meestal beveiligd met een authenticatie-sleutel. Geef hier de sleutel op waarmee data van en naar de rfid-kaart mag worden gestuurd. De sleutel bestaat uit een array van zes bytes. Wanneer geen sleutel wordt ingesteld, gebruikt de class *Rfid* de standaard-sleutel [0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF] waarmee rfid-kaarten worden geleverd.

readValue(index, value)

Leest een tekst van 16 tekens uit het geheugen van de rfid-kaart. De tekst wordt als *String* terug gegeven in *value*. Geeft met *index* op van welke positie de tekst moet worden gelezen. Bij een Mifare 1kB kaart loopt de index van 0 tot en met 63, maar niet alle posities zijn toegankelijk. Iedere vierde positie (index 3, 7, 11, enz.) is ontoeganlijk voor lezen. Wanneer de positie is beveiligd met een sleutel, moet eerst de routine *setAuthentication* worden aangeroepen.

writeValue(index, value)

Schrijft een tekst van 16 tekens in het geheugen van de rfid-kaart. De te schrijven tekst wordt als *String* opgegeven in *value*. Geeft met *index* op naar welke positie de tekst moet worden geschreven. Bij een Mifare 1kB kaart loopt de index van 0 tot en met 63, maar niet alle posities zijn toegankelijk. Iedere vierde positie (index 3, 7, 11, enz.) is ontoeganlijk voor schrijven. Wanneer de positie is beveiligd met een sleutel, moet eerst de routine *setAuthentication* worden aangeroepen.

Json

Veel applicaties gebruiken *json* als data-format. Met *json* is het mogelijk om de data in een vrije structuur te bewaren of te verzenden. Dit in tegenstelling tot bijvoorbeeld de tabelstructuur of de boomstructuur. In *json* kun je diverse datastructuren mixen. Het *json*-format wordt in deze beschrijving als bekend verondersteld. De class *Json* voert geen controle uit of de data in een correct format wordt aangeleverd. Zorg ervoor om de data in de juiste volgorde toe te voegen, zodat een correcte *json*-structuur wordt opgebouwd.

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

Json json;

void setup()
{
  json.add( "Binnen", 21.3, 1);
  json.add( "Buiten", 25.5, 2);
  json.add( "Temperatuur",
                      json.array());
  json = json.object();
}

void loop()
{
}
```

De bovenstaande code resulteert in de volgende json-structuur:

{"Temperatuur":["Binnen":21.3, "Buiten":25.50]}

#include <Json.h>

Json add(value, decimals = 2)

Voegt data aan de *json*-structuur toe. Bij een lege *json*-structuur zal deze vervolgens alleen de waarde bevatten. Anders wordt de waarde door een komma gescheiden achter de *json*-structuur toegevoegd. Tekst wordt automatisch tussen aanhalingstekens geplaatst. Bij een getal-waarde kan het aantal decimalen achter de komma (standaard 2) worden opgegeven.

Json add(key, value, decimals = 2)

Voegt data met de bijbehorende sleutel aan de *json*-structuur toe in het format *<key>:<value>*. Bij een lege *json*-structuur zal deze vervolgens alleen de geformatteerde waarde bevatten. Anders wordt de geformatteerde waarde door een komma gescheiden achter de *json*-structuur toegevoegd. Tekst wordt automatisch tussen aanhalingstekens geplaatst. Bij een getal-waarde kan het aantal decimalen achter de komma (standaard 2) worden opgegeven.

Json object(keep = false)

Maakt van de *json*-structuur een *object* door deze tussen accolades te plaatsen. Meestal gaat het om een tijdelijke structuur die aan een uiteindelijke structuur moet worden toegevoegd. Vandaar dat standaard na het teruggeven van het *object* de *json*-structuur wordt gewist om met een nieuwe tijdelijke structuur aan de slag te gaan. Geef *true* door bij paramater *keep* als dit niet gewenst is.

Json array(keep = false)

Maakt van de *json*-structuur een *array* door deze tussen blokhaken te plaatsen. Meestal gaat het om een tijdelijke structuur die aan een uiteindelijke structuur moet worden toegevoegd. Vandaar dat standaard na het teruggeven van de *array* de *json*-structuur wordt gewist om met een nieuwe tijdelijke structuur aan de slag te gaan. Geef *true* door bij paramater *keep* als dit niet gewenst is.

String toString()

Geeft de *json*-structuur als *String* terug.

void clear()

Wist de *json*-structuur.

int count(key)

Geeft terug hoevaak de sleutel *key* in de *json*-structuur voorkomt. Vervolgens kan met de *get*-routine de waardes bij de betreffende sleutel worden opgevraagd.

void get(value, key, occurence = 1)

Vraag data uit de *json*-structuur op aan de hand van een sleutel. Bepaal met de routine *count* hoevaak de sleutel aan de structuur is toegevoegd. Bij parameter *occurence* wordt doorgegeven welke uit deze groep wordt gewenst. De data wordt teruggeven in parameter *value*.

=

Bewerking die de ene *json*-structuur gelijk maakt aan de andere.

TcpServer *

* Alleen beschikbaar voor de Raspberry. [2]

De class *TcpServer* kan meerdere boards met daarop *TcpClient* via een netwerk bedienen en gebruikt daarbij het tcp-protocol. *TcpServer* en *TcpClient* kunnen dan data met elkaar uitwisselen.

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

TcpServer server;
String clientid = "test";

void setup()
{
  if ( !server.connect( 51515) ) {
    Serial.println(
         "Geen contact met het netwerk");
    exit( 1);
  }

  // wacht op de betreffende client
  while ( !server.hasClient( clientid) );
}

void loop()
{
  if ( server.dataReady( clientid) ) {
    Serial.println( server.receive( clientid));
    server.send( "Antwoord");
  }
}
```

#include <tcpserver.h>

= connect(poort)

Setup. Deze routine maakt via wifi contact met het netwerk. Geef als parameter de poort door waarlangs de communicatie met de clients verloopt. De routine geeft terug of de verbinding is gelukt.

= connected()

Geeft terug of er verbinding is met een netwerk.

close()

Hiermee wordt de toegang tot het netwerk gesloten.

getClients(list)

Vraag met deze routine een lijst met verbonden clients op. De *id*'s van de clients worden in de parameter *list* teruggegeven. Parameter *list* moet van het type *StringList* zijn.

= client()

Geeft de *id* van de client terug. Deze routine is alleen zinvol als de server steeds met maar één client contact heeft.

= hasClient(id)

Geeft terug of er verbinding is met de client met het opgegeven *id*.

[2] De arduino library is gebaseerd op het arduino wifi-shield en kan met maar één client communiceren.

stopClient(id)

Deze routine zendt een *stop*-signaal naar de client met het opgegeven *id*.

= dataReady(id)

Test met deze routine of de client met het opgegeven *id* data heeft verzonden.

= receive(id, data)

Geeft in parameter *data* de informatie terug, die door de client met het opgegeven *id* werd verzonden. De paramater *data* moet van het type *String* zijn. Let op! Als de client geen informatie heeft verzonden zal parameter *data* een lege string zijn. Dit is ook het geval als de client daadwerkelijk een lege string als informatie heeft verzonden. Om beide situaties te kunnen onderscheiden geeft de routine *read* terug of de waarde in parameter *data* daadwerkelijk is verzonden.

= send(id, data)

Geef in parameter *data* de informatie door, die naar de client moet worden verzonden. De routine *send* geeft terug of de informatie voor verzending kon worden klaargezet. Omdat het verzenden zelf asynchroon [3] verloopt, kunnen eventuele fouten bij het verzenden alleen met de routines *latestError* of *latestErrors* worden opgepoord. Het is niet mogelijk om te controleren of de *data* bij de client is aangekomen, tenzij het programma van de client een bevestiging stuurt.

= latestError()

Geeft de laatst opgetreden fout als tekst terug. Wanneer er in de laatst opgeroepen routine geen fout optrad, wordt een lege *String* teruggegeven.

= latestErrors()

Geeft een verwijzing (*pointer*) naar de lijst (*StringList*) met de laatste tien foutmeldingen terug. Aan de hand van deze lijst kunnen fouten in de communicatie met de clients worden opgespoord.

[3] 'Asynchroon' betekent 'niet op hetzelfde moment'. *TcpServer* werkt met buffers voor het ontvangen en verzenden van data. De routine *send* plaatst de informatie in de buffer en kan alleen teruggeven of dat gelukt is. Een fractie van een seconde later – terwijl het programma al verder is gegaan – wordt de informatie pas werkelijk verzonden.

TcpClient

De class *TcpClient* maakt via een netwerk contact met de class *TcpServer* op een ander board en gebruikt daarbij het tcp-protocol. Daarna kunnen *TcpClient* en *TcpServer* data naar elkaar verzenden.

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

TcpClient client;
String clientid = "test";

void setup()
{
  if ( !client.wifi( "ssid", "wachtwoord") ||
       !client.connect( "1.0.0.127",
                        51515, clientid) {
    Serial.println( "Geen contact met de server");
    exit( 1);
  }
}

void loop()
{
  client.send( "Vraag");
  String data;
  // wacht op antwoord
  while ( !client.receive( data) );
  Serial.println( data);

  delay( 2000);
}
```

#include <tcpclient.h>

= wifi(ssid, passphrase) *

Setup. Maakt verbinding met een access-point. De routine geeft terug of de verbinding is gelukt.

* Alleen voor de arduino [4].

= connect(host, port, id)

Setup. Deze routine maakt contact met de server. De host kan lokaal zijn ("localhost" of "1.0.0.127") of in een netwerk zitten met een ip-adres. De poort moet overeenkomen met de poort waarlangs de server contact heeft met het netwerk. Geef de client een uniek *id* in het netwerk. De class *TcpServer* onderscheidt de clients met hun *id*. De routine geeft terug of er verbinding is met de server.

= connected()

Geeft terug of er verbinding is met de server.

close()

Hiermee wordt de verbinding met de server gesloten.

[4] Op de raspberry maakt het operating systeem verbinding met een access-point. (Klik op het netwerk-symbool in de taakbalk en maak verbinding met het gewenste wifi-punt). Het is daarom niet nodig dit ook nog eens in een programma te doen.

= receive(data)

Ontvangt *data* van de server. De paramater *data* moet van het type *String* zijn. Let op! Als de client geen informatie heeft verzonden zal parameter *data* een lege string zijn. Dit is ook het geval als de client daadwerkelijk een lege string als informatie heeft verzonden. Om beide situaties te kunnen onderscheiden geeft de routine *read* terug of de waarde in parameter *data* daadwerkelijk is verzonden.

= send(id, data)

Verzendt *data* naar de server. Geeft terug of het verzenden is gelukt. Het is niet mogelijk om te controleren of de *data* bij de server is aangekomen, tenzij het programma van de server een bevestiging stuurt.

= error()

Geeft de laatst opgetreden fout als tekst terug. Wanneer er in de laatst opgeroepen routine geen fout optrad, wordt een lege *String* teruggegeven.

SqlClient *

* Alleen beschikbaar voor de Raspberry.

De class *SqlClient* maakt contact met een mysql-server en voert sql-queries uit. De queries worden als tekst opgegeven en moeten volgen de regels van de mysql-server. Het resultaat van een query wordt per rij uitgelezen.

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

SqlClient sql;

void setup()
{
  // maak contact met de database
  if ( !sql.connect( "1.0.0.127", "db-naam",
                     "gebruiker", "wachtwoord") ) {
    Serial.println( "Geen contact met de server");

    // sluit het programma met foutcode 1
    exit( 1);
  }

  // voer een query uit
  if ( !sql.query( "SELECT * FROM tabel-naam;") ) {
    Serial.println( "Fout in de query");

    // sluit het programma met foutcode 2
    exit( 2);
  }

  // haal de eerste rij
  StringList rij = sql.nextRow();

  // toon alle rijen
  while ( rij.size() ) {

    // toon alle velden van de rij
    for ( int i = 0; i < rij.size(); i++ )
      Serial.print( rij[i] + " ");
    Serial.println( "");

    // haal de volgende rij
    rij = sql.nextRow();
  }

  // sluit het programma zonder foutcode
  exit(0);
}

void loop()
{
}
```

#include <sqlclient.h>

= connect(host, database, user, password)

Setup. Deze routine maakt contact met de database. Alle parameters worden als *String* of tussen "" doorgegeven De host kan lokaal zijn ("localhost" of "1.0.0.127") of in een netwerk zitten met een ip-adres. Een database moet toegankelijk zijn met de gebruikersnaam en het wachtwoord. De routine *connect* geeft terug of de database kon worden geopend.

close()

Hiermee wordt de toegang tot de database gesloten.

= query(sql-query)

Met de routine *query* kan iedere geldige sql-query worden uitgevoerd. De routine geeft terug of de query correct is uitgevoerd. Veel sql-query's zien er als volgt uit:

SELECT <kolom>, <kolom> FROM <tabel> WHERE <kolom> = "<waarde>" AND <kolom> = <waarde> ORDER BY <kolom>;

Geeft de sql-query als *String* of tussen "" door.

= columnHeaders()

De routine *headers* geeft een lijst (*StringList*) met kolom-headers terug. De headers staan gerangschikt in de volgorde zoals in de sql-query genoemd.

= nextRow()

Meestal heeft een sql-query meerdere datarijen als resultaat. De routine *nextRow* geeft bij iedere nieuwe aanroep de volgende datarij in een lijst (*StringList*) terug. De lijst bevat de kolomwaardes, gerangschikt in de volgorde zoals in de sql-query genoemd. Wanneer er alle datarijen zijn uitgelezen, geeft de routine een lege lijst terug (*StringList::size() == 0*).

= filloutQuery(query, value_list)
= filloutQuery(query, value1, value2, …)

Gebruik deze routine om waardes bij een query in te vullen. Bijvoorbeeld zal de aanroep `filloutQuery( "SELECT * FROM test WHERE val = @@1;", "35")` resulteren in de query `SELECT * FROM test WHERE val = 35;`. Het token *@@1* wordt vervangen door *value1* achter de *query* – in het voorbeeld *35*. Overigens hoeven de tokens niet te worden genummerd en wordt het teken acher *@@* als id opgevat. De verschillende tokens in de *query* worden van links naar rechts vervangen door de opeenvolgende waardes achter de *query*. In het volgende voorbeeld wordt een token twee maal gebruikt, waaruit de manier van invullen blijkt:

`filloutQuery( "SELECT * FROM @@4 WHERE @@B = '@@4';", "test", "info");`
resulteert in: `SELECT * FROM test WHERE info = 'test';`

Let er op dat het aantal verschillende tokens in een *query* overeenkomt met het aantal waardes achter de *query*. Wanneer er meer verschillende tokens worden gebruikt dan er waardes in de lijst voorkomen, zal dit tot een programmafout leiden. Ook moet ieder token een vast format hebben, met twee @-tekens gevolgd door één ander teken (letter of cijfer);

De routine *filloutQuery* geeft de ingevulde query terug.

= error()

Vraag met deze routine de laatste foutmelding van de database op. De melding wordt als tekst teruggeven.

Camera *

* Alleen beschikbaar voor de Raspberry.

De class *Camera* is gebaseerd op *OpenCV* en alle door *OpenCV* ondersteunde camera's kunnen worden gebruikt. De class neemt snapshots, maar kan geen beelden streamen. Wel biedt de class enkele manieren om een beeld te analyseren. Wanneer daar gebruik van wordt gemaakt, moet vooraf de routine *setAnalyse* worden aangeroepen.

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

Camera cam;
Mat    img;

void setup()
{
  cam.init( 800, 600);
  cam.start();
}

void loop()
{
  cam.read();
  img = cam.image();
  imshow( "titel", img);
  waitKey( 40);
}
```

#include <camera.h>

init(width, height)

Setup. Stel de *video capture* in op de gewenste afmetingen van het beeld. Het werkelijke beeldformaat hoeft niet precies met de opgegeven waardes overeen te komen. Dit hangt af van de mogelijkheden van de camera. Wanneer een camera het opgegeven formaat niet ondersteunt, zal een snapshot in het meest gelijkende formaat worden weergeven.

setAnalyse(threshold, blocksize)

Setup. Voor de beeldanalyse wordt een beeld in blokjes met een hoogte en breedte van *blocksize* verdeeld. Van de blokjes wordt een gemiddelde kleur vastgesteld. Per kleurelement (rood, groen en blauw) wordt de actuele intensiteit vergeleken met de eerdere intensiteit of met een referentie intensiteit. Als het verschil voor ieder kleurelement boven de *threshold* waarde uitkomt, is er sprake van kleurwijziging. De diverse manieren van analyseren gebruiken de kleurwijziging verschillend.

= start()

Hiermee wordt de camera op *capture mode* ingesteld. De routine geeft *true* terug als het is gelukt en anders *false*.

read()

Hiermee wordt met de camera een snapshot genomen. Vooraf moet de routine *start* zijn aangeroepen. Totdat de routine *stop* wordt aangeroepen, kunnen er met *read* meerdere snapshots achter elkaar worden genomen.

stop()

Hiermee de *capture mode* van de camera uitgeschakeld.

setReferenceColor(color)

Stel met deze routine de referentiekleur in, welke bij de kleuranalyse *ALT_REFERENCE* moet worden gebruikt. (Zie de routine *percentageChanged*).

= percentageChanged(alterationtype)

Wanneer bij *alterationtype* de waarde *ALT_INTENSITY* of *ALT_COLOR* wordt opgegeven, bepaalt *percentageChanged* welk percentage van een volledig beeld is gewijzigd ten opzichte van het voorafgaande beeld. Bij *ALT_INTENSITY* wordt de totale intensiteit van de kleurelementen (rood, groen en blauw) beoordeeld. Bij *ALT_COLOR* wordt de verhouding tussen de intensiteit van de kleurelementen beoordeeld. Wanneer bij *alterationtype* de waarde *ALT_REFERENCE* wordt opgegeven, bepaalt *percentageChanged* welk percentage van het beeld afwijkt van de referentiekleur. De referentiekleur wordt ingesteld met *setReferenceColor*.

Om *precentageChanged* te kunnen gebruiken, moet voorafgaand de routine *setAnalyse* zijn aangeroepen (bijvoorbeeld in de *setup*).

= image()

Geeft een snapshot terug. De snapshot heeft het standaard *Mat* format van *OpenCV*. Je kunt de snapshot gebruiken als parameter in de *OpenCV*-routine *imshow* om het beeld op het scherm te tonen. (Bedenk dat *OpenCV* een beeld pas werkelijk toont als *imshow* wordt gevolgd door een aanroep naar *waitKey* – zie de *OpenCV* documentatie op internet.)

= width()

Geeft de werkelijke breedte van de snapshots terug.

= height()

Geeft de werkelijke hoogte van de snapshots terug.

Nextion

De class *Nextion* maakt gebruik van seriële communicatie tussen de nextion en de arduino of de raspberry. Gebruik de *Nextion-editor* om pagina's op de nextion en de interactie met de gebruiker te ontwerpen. De nextion en de arduino of de raspberry communiceren als volgt:

Nextion	**>>**	**Arduino/Raspberry**
print "<tekst>~"		String = Nextion::receive();
print <veld>.txt print "~"		String = Nextion::receive();
print "#" print <getal>		String = Nextion::receive();
print "#" print <veld>.val		String = Nextion::receive();

Arduino/Raspberry	**>>**	**Nextion**
Nextion::sendPage ("<pagina>");		pagina
Nextion::sendText("<veld>", String)		tekst-veld
Nextion::sendNumber("<veld>", long)		getal-veld

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

Nextion nxt;

void setup()
{
  nxt.sendPage( "0");
  nxt.sendText( "b0", "Aan");
  nxt.sendText( "b1", "Uit");
}

void loop()
{
  String s = nxt.receive();
  if ( s != "" )
    nxt.sendText( "t0", s);
}
```

Wanneer in bovenstaand voorbeeld de *Touch Press Event* van de knoppen b0 en b1 geprogrammeerd is met respectievelijk *print "aan~"* en *print "uit~"*, dan zal het bovenstaande programma deze teksten echoën naar het tekstveld t0.

Vergeet bij het verzenden van de nextion naar de arduino of de raspberry niet het #-teken voorafgaand aan een getalwaarde of het ~-teken volgend op een tekstwaarde te verzenden. Deze tekens zijn alleen voor de communicatie van belang en worden door de instructie *receive* verder genegeerd. Verder blijft alle door de Nextion verzonden data in de communicatie-buffer van de arduino of raspberry aanwezig, totdat deze met de instructie *receive* wordt uitgelezen. Het ontvangen van data van de nextion is dus niet tijd-kritisch. Andersom worden de instructies *sendPage*, *sendText* en *sendNumber* meteen op de nextion uitgevoerd.

#include <nextion.h>

init(serial)

Setup. Stel met deze routine in via welke interface het display is verbonden. Op de raspberry is dit meestal *Serial1* (pin 14 en 15), op de arduino uno meestal *Serial* (pin 0 en 1), maar op de arduino mega heb je de keus tussen *Serial* (pin 0 en 1), *Serial1* (pin18 en 19), *Serial2* (pin 16 en 17) en *Serial3* (pin 14 en 15). Let erop dat de *Tx* van de nextion wordt aangesloten op de *Rx* van het board en de *Rx* van de nextion op de *Tx* van het board.

= receive()

Ontvangt de data die door de nextion wordt verzonden. De data wordt per waarde ontvangen. Wanneer de nextion drie waardes *print*, zijn er ook drie *receive*-instructies nodig om de drie waardes te ontvangen. Hoewel de nextion data als *tekst* en als *getal* kan verzenden, worden beide door de routine *receive* als *String* teruggegeven.

sendPage(page)

Met deze routine wordt op de nextion de opgegeven pagina getoond. De pagina mag als naam en als nummer tussen aanhalingstekens worden opgegeven.

sendText(field, text)

Stelt de tekst van één van de elementen (*field*) op een nextion-pagina in.

sendNumber(field, text)

Stelt het getal van één van de elementen (*field*) op een nextion-pagina in.

Scherm module *

* Alleen beschikbaar voor de Raspberry.

De scherm-module is geen class die moet worden geïnclude, maar komt beschikbaar wanneer de regel *#include <Arduino.h>* wordt vervangen door *#include <ArduinoGtk.h>*. De scherm-module zit dan in het window-event-systeem geïntegreerd. De standaard *loop*-functie wordt herhaald aangeroepen in zogenaamde *idle time* – de tijd dat het window-event-systeem niets staat te doen. Let op: hoe meer code er in de *loop*-functie staat, des te langer het duurt voordat het systeem op *events* zoals muisklikken en toetsindrukken reageert.

In plaats van de *loop*-functie zwaar te belasten, maakt het window-event-systeem het mogelijk om routines aan *events* te koppelen. Daarvoor gebruik je de functies *callOnMouseLClick*, *callOnMouseRClick*, *callOnMouseLRelease* en *callOnMouseRRelease*. De routines die je hier als parameter doorgeeft, worden bij de betreffende *events* aangeroepen.

In de *event*-routines vraag je met de functie *mouseWidget* op welk *widget* er tijdens de *event* onder de muisaanwijzer zit. *Widgets* zijn items op het scherm, zoals plaatjes, buttons en tekstvelden. Aan ieder *widget* dat je op het scherm plaatst geef je een naam mee en in de *event*-routine check je of het *widget* met gewenste naam onder de muisaanwijzer zit.

DeviceLib biedt twee handige oplossingen om met *widgets* te werken. Dat zijn de *drag-and-drop* en het samenbinden van *widgets* in tabellen.

Tijdens *drag-and-drop* sleep je een *widget* naar een andere plaats op het scherm. Met de *drag*-functie geef je door welk *widget* moet worden gesleept en met de *drop*-functie stop je het slepen. Met de functie *setDragField* kun je het gebied op het scherm afbakenen, waar een

```
#include <ArduinoGtk.h>

void* antw; // tLabel widget
void* afb;  // tImage widget


void muisklik()
{
  drag( mouseWidget( "AFB"));
}


void muislos()
{
  drop();

  if ( mouseWidget( "JA") )
    setText( antw, "Ja gekozen");
  if ( mouseWidget( "NEE") )
    setText( antw, "Nee gekozen");
  place( afb, 110, 10);

}


void setup()
{
  callOnMouseLClick( muisklik);
  callOnMouseLRelease( muislos);

  afb = create( tImage, "AFB");
  setSize( afb, 90, 20);
  place( afb, 110, 10);

  antw = create( tLabel, "ANTW");
  setSize( antw, 290, 20);
  place( antw, 10, 40);
  setText( antw, "<Antwoord>");

  void *w;

  addStickyArea( 10, 10, 90, 20);
  w = create( tImage, "JA");
  setSize( w, 90, 20);
  place( w, 10, 10);

  addStickyArea( 10, 210, 90, 20);
  w = create( tImage, "NEE");
  setSize( w, 90, 20);
  place( w, 210, 10);

}


void loop()
{
}
```

widget mag worden gesleept. Het is ook mogelijk om *sticky areas* op te geven – dat zijn gebieden waar een *widget* mag worden gedropt.

Het samenbinden van *widgets* in tabellen is bedoeld om formulieren te bedienen. Over het algemeen wordt de inhoud van een tabel gevuld met informatie uit een database. Een tabel-rij komt dan overeen met een record in de database. De uitwisseling tussen scherm en database vindt plaast via de class *StringList*:

SqlClient::nextRow	>>	*StringList*	>>	*setRowValues*
getRowValues	>>	*StringLIst*	>>	*SqlClient::filloutQuery*

Programmascherm

setTitle(title)

Setup. De titel van een programma vind je boven in het scherm terug.

setSize(width, height)

Setup. Stel de hoogte en breedte van het programmascherm in.

Widgets (schermelementen)

= create(type, name, parameter = “”)

Setup. Maakt een widget van een bepaald *type* aan. Geef ieder *widget* een unieke naam in *name* mee. Andere routines maken gebruik van de naam om het *widget* te herkennen. De volgende *type widgets* worden ondersteund: *tImage* (plaatje), *tLabel* (tekst), *tEdit* (invulveld), *tButton* (knop) en *tCheck* (vinkje). De *widgets* hebben extra informatie in de *parameter* nodig:

Type widget	*Parameter*
tImage	Pad naar het bestand met het plaatje.
tLabel	De tekst van het label, die op het scherm wordt getoond.
tEdit	De tekst die als eerste in het invulveld wordt getoond.
tButton	De tekst die op de knop wordt getoond.
tCheck	De tekst die naast het vinkje wordt getoond.

De routine *create* geeft het *widget* terug.

appendDefaultStyle (type, css_style)

Setup. Per *widget-type* kun je de standaardopmaak wijzigen. Geef het *type* door en welke stijl je wilt toepassen. De stijl volgt de regels van CSS, bijvoorbeeld “font-size: 10px”.

setSize(widget, width, height)

Setup. Geef het *widget* een bepaalde breedte en hoogte op het scherm.

place (widget, x, y)

Zet het *widget* op een bepaalde plaats op het scherm.

= name(widget)

Geeft de naam van een *widget* terug die verschillende routines nodig hebben om een *widget* aan te herkennen.

destroy(widget)

Verwijdert een *widget* dat met *create* werd aangemaakt.

show(widget, show = true)

Toont een *widget* op het scherm als bij parameter *show* de waarde *true* wordt doorgegeven – anders wordt *hide* aangeroepen.

hide(widget)

Maakt een *widget* onzichtbaar.

setText(widget, type, text)

Wijzigt de tekst van een *widget*. Geef ook het *type* van het *widget* door (*tLabel* of *tEdit*).

= text(widget, type)

Geeft de tekst van een *widget* terug. Geef ook het *type* van het *widget* door (*tLabel* of *tEdit*).

setCheck(widget, checked)

Plaatst of wist het vinkje van het *widget*. Het *widget* moet van het type *tCheck* zijn. Vul bij *checked* de waarde *true* (aangevinkt) dan wel *false* (vinkje gewist) in.

= isChecked(widget, type)

Geeft terug of het vinkje van het *widget* is geplaatst (*true*) dan wel gewist (*false*). Het *widget* moet van het type *tCheck* zijn.

Drag-and-drop

drag(widget)

Zorgt ervoor dat het *widget* met de muisaanwijzer mee wordt gesleept.

drop()

Stopt het slepen van een *widget*. Het *widget* blijft staan waar het naartoe is gesleept.

setDragField(xLeft, yTop, xRight, yBottom)

Deze routine bakent het gebied op het scherm af, waarbinnen een *widget* heen en weer kan worden gesleept.

Sticky areas

= addStickyArea(x, y, width, height)

Zorgt ervoor dat een *widget* dat wordt gesleept, blijft plakken op bepaalde schermposities. Geeft de index van het gebied terug.

removeStickyArea(ix)

Heft een *sticky area* op, zodat een *widget* dat wordt gesleept, er niet meer blijft plakken.

clearStickyAreas()

Heft alle *sticky areas* op.

Tabellen

= addRow(widget_list)

Voegt een lijst met *widgets* samen tot een rij in de tabel. De routine geeft de index van de rij terug.

setRowValues(row, value_list, type)

Vult de velden van een rij met de waardes uit de *value_list*. Geeft in *row* de index van de rij op. Alle velden moeten van hetzelfde *type* zijn (*tLabel* of *tEdit*).

getRowValues(row, value_list, type)

Vraagt de waardes van de velden in een rij op en plaatst deze in *value_list*. Geef met *row* de index van de rij op. Alle velden moeten van hetzelfde *type* zijn (*tLabel* of *tEdit*).

De muis

= mouseX()

Geeft de x-positie van de muis terug, waar er voor het laatst werd geklikt of voor het laatst werd losgelaten.

= mouseY()

Geeft de y-positie van de muis terug, waar er voor het laatst werd geklikt of voor het laatst werd losgelaten.

mouse(x, y)

Geeft in *x* en *y* de positie van de muis terug, waar er voor het laatst werd geklikt of voor het laatst werd losgelaten.

= mouseWidget(name = “”)

Geeft het *widget* terug waarop voor het laatst werd geklikt of dat werd losgelaten [5]. Geeft *null* terug als er niet op een *widget* werd geklikt. Geef bij *name* de naam van een *widget* door om te weten of er op het betreffende *widget* werd geklikt.

= callOnMouseLClick(callback)
= callOnMouseRClick(callback)

Zorgt ervoor dat de routine *callback* wordt opgeroepen als er respectievelijk op de linker dan wel rechter muisknop wordt geklikt.

= callOnMouseLRelease(callback)
= callOnMouseRRelease(callback)

Zorgt ervoor dat de routine *callback* wordt opgeroepen als respectievelijk de linker dan wel rechter muisknop wordt losgelaten.

[5] Als zich meerdere *widgets* op de plaats van klikken bevinden, wordt alleen de bovenste teruggegeven.

Basis classes

Device

```
Device();
void startTimerEvent( uint32_t msec, void (*event)());
void stopTimerEvent();
uint32_t millisTimer();
```

Actuator

```
Actuator();
bool isOn();
virtual void setOff();
virtual void setOn();
```

Sensor

```
Sensor();
bool dataReady();
virtual void read();
```

Devices

Beeper

```
Beeper();
uint8_t addTone( uint16_t herz, uint16_t msec);
void beep();
void clear();
void insertTone( uint8_t pos, uint16_t herz, uint16_t msec);
void play();
void remove( uint8_t pos, uint8_t count = 1);
void setBeeper( uint16_t herz, uint16_t msec);
void setPin( uint8_t pin);
void setOff();
void setOn();
```

Buzzer

```
Buzzer();
void setOn();
void setOff();
void setPin( uint8_t pin);
```

Camera

```
Camera();
struct PNT { uint16_t x; uint16_t y; };
struct CLR { uint8_t  b; uint8_t  g; uint8_t  r; };
struct SIZ { uint16_t w; uint16_t h; };
uint8_t height();
Mat image();
void init( uint16_t width, uint16_t height);
uint8_t percentageChanged( uint8_t alterationtype);
void read();
void setAnalyze( uint8_t threshold, SIZ blocksize = {0,0} );
void setReferenceColor( CLR color, bool background);
uint32_t spotCount( uint8_t minimalsize = 30);
bool start();
void stop();
uint16_t width();
```

DcMotor

```
DcMotor();
void setPin( uint8_t pin, bool pwm = true);
void setSpeed( uint8_t speed);
```

Distance

```
Distance();
uint16_t cm();
uint16_t inch();
void read();
void setPin( uint8_t trigger, uint8_t echo);
```

Led

```
Led();
void setBlink( uint32_t on = NOBLINK, uint32_t off = NOBLINK );
void setBrightness( uint8_t on = BRIGHT, uint8_t off = 0);
void setInversion( bool invert = true);
void setOn();
void setOff();
void setPin( uint8_t pin, bool pwm = false);
```

Motion

```
Motion();
bool detected();
void read();
```

```
void setOn();
void setOff();
void setPin( uint8_t pin);
```

Nextion

```
void init( SerialRPi& serial);
void init( HardwareSerial& serial);
String receive();
void sendPage( String page);
void sendNumber( String field, long data);
void sendText( String field, String data);
```

Rfid

```
Rfid();
void done();
bool init();
void read();
bool readValue( uint8_t index, String& value);
void reset();
void setAuthentication( const RFIDKEY key);
void setPin( uint8_t pinIrq, uint8_t pinRst = 0);
void setTimeout( uint32_t msec);
String tag( uint32_t expireAfter = 0);
bool writeValue( uint8_t index, String value);
```

RgbLed

```
RgbLed();
enum tColor { Red, Green, Blue, Magenta, Yellow, Cyan, White };
Led* blueLed();
Led* greenLed();
Led* redLed();
void setColor( tColor color);
void setColor( uint8_t red, uint8_t green, uint8_t blue);
void setInversion( bool invert = false);
void setPin( uint8_t pinRed, uint8_t pinGreen,
             uint8_t pinBlue, bool pwm = true);
void setOff();
void setOn();
```

Rotary

```
Rotary();
bool backward();
bool forward();
bool moving();
```

```
void read();
float rpm();
float rps();
void setPin( uint8_t pin1, uint8_t pin2);
void setTicksPerRotation( uint16_t ticks);
void setTimeout( uint32_t msec);
void swapDirection();
```

Sound

```
Sound();
enum tStreamer { Local, HDMI, Bluetooth };
uint8_t addFile( String path);
uint8_t addText( String text);
void clear();
void init( Streamer streamer, String device = "");
void insertFile( uint8_t pos, String path);
void insertText( uint8_t pos, String text);
void playFile( String path);
void playList();
void playText( String text);
void remove( uint8_t pos, uint8_t count = 1);
```

SqlClient

```
SqlClient();
void close();
StringList columnHeaders();
bool connect( const String host, const String database,
              const String user, const String password);
String error();
static String filloutQuery( const String querystr,
                            StringList values);
static String filloutQuery( const String querystr, ...);
StringList nextRow();
bool query( const String querystr);
```

Switch

```
Switch();
bool changed();
bool pressed();
void read();
bool released();
void setNormalOpen( bool normalopen);
void setPin( uint8_t pin, Connect connect);
void triggerEvent( void (*callback)());
```

TcpClient

```
TcpClient();
bool connect( String host, uint16_t port, String id);
bool connected();
bool close();
bool read( String& data);
bool send( String data);
String error();
```

TcpServer

```
TcpServer();
bool connect( uint16_t port);
bool connected();
void close();
void getClients( StringList& list);
bool hasClient( String client);
void stopClient( String client);
String client();  // acts on first client in the list
bool dataReady(); // acts on first client in the list
bool dataReady( String client);
bool read( String client, String& data);
bool send( String client, String data);
static String latestError();
static StringList* latestErrors();
```

Temperature

```
Temperature();
static StringList getIdList( uint8_t pin = 10);
void setSensor( String id, uint8_t pin = 10);
float kelvin();
float celcius();
float fahrenheit();
float humidity();
void read();
```

Extra

Average

```
Average();
void  add( float value);
float avg();
void  clear();
```

Json

```
Json();
Json( const Json& json);
Json add( const Json value);
Json add( const String value);
Json add( const char* value);
Json add( const double value, int decimals = 2);
Json add( const bool value);
Json add( const String key, const Json value);
Json add( const String key, const String value);
Json add( const String key, const char* value);
Json add( const String key, const double value, int decimals = 2);
Json add( const String key, const bool value);
Json array( bool keep = false);
void clear();
int count( const String& key);
void get( String& value, const String key,
          const uint8_t occurence = 1);
void get( double& value, const String key,
          const uint8_t occurence = 1);
void get( bool& value, const String key,
          const uint8_t occurence = 1);
Json object( bool keep = false);
Json operator= ( const Json json);
String toString();
```

LinkedList

```
LinkedList();
virtual bool add(T);
virtual T at( int index);
virtual void clear();
virtual int count()
virtual bool insert( int index, T);
virtual T operator[]( int index);
virtual void removeAll();
virtual T removeAt( int index);
virtual bool replace( int index, T);
virtual int size();
```

Scherm-module

Programmascherm

```
void setSize( uint16_t width, uint16_t height);
void setTitle( String title);
```

Widgets (schermelementen)

```
#define tLabel  1
#define tImage  2
#define tButton 3
#define tCheck  4
#define tRadio  5
#define tEdit   6
void appendDefaultStyle( uint8_t type, String style);
void* create( uint8_t type, String name);
void* create( uint8_t type, String name, String param);
void destroy( void* widget);
void hide( void* widget);
bool isChecked( void* widget);
String name( void* widget);
void place( void* widget, uint16_t x, uint16_t y);
void setCheck( void* widget, bool check);
void setSize( void* widget, uint16_t width, uint16_t height);
void setText( void* widget, uint8_t type, String text);
void show( void* widget);
void show( void* widget, bool show);
String text( void* widget, uint8_t type);
```

Drag-and-drop

```
void drag( void* widget);
void drop();
void setDragField( uint16_t left, uint16_t top,
                   uint16_t right, uint16_t bottom);
```

Sticky areas

```
uint8_t addStickyArea( uint16_t left, uint16_t top,
                       uint16_t width, uint16_t height);
void clearStickyArea();
void removeStickyArea( uint16_t ix);
```

Tabellen

```
void addRow( WidgetList* widgets);
void getRowValues( uint8_t row, StringList& values,
                   uint8_t type = tEdit);
void setRowValues( uint8_t row, StringList& values,
                   uint8_t type = tEdit);
```

De muis

```
void mouse( uint16_t &x, uint16_t &y);
void *mouseWidget( String name = "");
uint16_t mouseX();
uint16_t mouseY();
void callOnMouseLClick( CALLBACK routine);
void callOnMouseRClick( CALLBACK routine);
void callOnMouseLRelease( CALLBACK routine);
void callOnMouseRRelease( CALLBACK routine);
```

Devices aansluiten

Belangrijk om te weten

Veilig werken

- Raak de metalen aansluitpinnen van de ic's en boards nooit aan. De statisch elektriciteit op je lichaam kan ze stuk maken. *Tip*: Wrijf voor dat je begint met beide handen over een groot ongeverfd metalen voorwerp. Draag kleding van katoen.
- De dampen die bij het solderen vrijkomen, zijn schadelijk voor je gezondheid. *Tip*: Met een klein ventilatortje kun je ervoor zorgen dat ze bij je vandaan worden geblazen. Zorg voor voldoende ventilatie.
- Tot de 40 volt is het werken met elektriciteit veilig. Daarboven kan het dodelijk zijn (het stopcontact heeft 230 volt!). Over het algemeen werken de devices met voltages van 3, 5, 12 of 24 volt en is er niets aan de hand.
- Op een raspberry of arduino en op de boards van de devices staat aangegeven welke aansluitpin de plus (Vcc/Vdd/3V/5V) is en welke de ground (Gnd/0V).
    - Plus- en ground-pinnen mag je nooit op elkaar aansluiten. Dan krijg je kortsluiting.
    - GPIO-pinnen gaan stuk als je ze rechtstreeks op een plus- of ground-pin aansluit [6].
    - Als je een 3V-pin van een device op een 5V-pin van de arduino of raspberry aansluit, gaat het device stuk.
    - Als je een 5V-pin van een device op een 3V-pin van de arduino of raspberry aansluit, werkt het device niet goed.

Rekenen met elektriciteit

- De pinnen van een raspberry kunnen per stuk maximaal 16 mA (milli-ampère) aan.
  De banks (met meerdere pinnen) van een raspberry kunnen maximaal 50 mA aan.
  De 5 V pinnen van een raspberry kunnen samen zoveel aan als de adapter aankan. (Bedenk dat de raspberry zelf al 1.5 A neemt.)
  De 3.3 V pinnen van een raspberry kunnen samen maximaal 400 mA aan. (Bedenk dat deze 400 mA met de GPIO-pinnen moet worden gedeeld).
  De pinnen van een arduino kunnen per stuk maximaal 40 mA aan.
  De pinnen van een arduino kunnen samen maximaal 200 mA aan.
  De 5 V en 3.3 V output-pinnen van een arduino kunnen samen maximaal 500 mA aan.
- De GPIO-pinnen van een raspberry werken op 3.3 V (volt).
  De GPIO-pinnen van een arduino werken op 5.0 V [7].
- Je kunt de hoeveelheid milli-ampère van een schakeling verkleinen, door de weerstand (ohm's) te vergroten. Dit is een eenvoudige rekensom: *ohm = volt ÷ ampère*, of beter gezegd: *weerstand = spanning ÷ stroomsterkte*, of met natuurkundige letters: $R = U \div I$.
    - De minimale weerstand bij een raspberry pin: $R = 3.3 \div 0.016 = 210$ *ohm*.
    - De minimale weerstand bij een arduino pin: $R = 5.0 \div 0.040 = 125$ *ohm*.

  Als richtlijn gebruiken we meestal een weerstand van 330 Ω (ohm).

[6] Dit hangt overigens af van de modus (input/output) van de pinnen en hoe de interne elektronica van de raspberry en de arduino werkt. Wie precies weet hoe deze boards werken, kent de uitzonderingen waarbij een GPIO-pin niet stuk gaat.

[7] Dit geldt voor de meest gebruikte arduino-boards zoals de arduino uno en de arduino mega. Er bestaan kleinere arduino-boards waarvan de pinnen juist op 3.3 V werken.

- Kies in een schakeling liever de grootst mogelijke weerstand waarbij een schakeling nog werkt. Want hoe groter je de weerstand kiest, des te minder stroom er wordt getrokken en des te meer pinnen je kunt gebruiken.
- Let op: Een led-lampje (of een gewone diode) zorgt ervoor dat de spanning in een schakeling lager wordt. We noemen dit de spanningsval erover. Omdat dit maar heel weinig is, heeft dit meestal geen effect op de schakeling. In de praktijk rekenen we met een spanningsval van 0.3 V per led.
- Hetzelfde geldt voor een transisor. Hier is de spanningsval ongeveer 0.7 V.

Zelf schakelingen maken

Handige componenten

- Weerstand
  Dit is wel het meest verkochte elektronische component. Je gebruikt deze om de stroomsterkte in een schakeling te verlagen of om een GPIO-pin van de raspberry of de arduino te beschermen als je schakeling stuk gaat.
  Er bestaan verschillende soorten weerstanden, die voornamelijk in nauwkeurigheid verschillen. Over het algemeen volstaan koolstof-weerstanden met een nauwkeurigheid van 5%. Ze zijn beige van kleur. Het aantal ohm van de weerstand wordt weergegeven met vier gekleurde ringen, waarbij de achterste (goud- of zilverkleurige) ring niet meetelt.

- Transistor
  Een transistor gebruik je om grotere spanningen en stroomsterktes te schakelen.
  Er bestaan twee types, aangeduid met PNP en NPN. Over het algemeen gebruik je het type NPN. Bovendien kies je liever een darlington-transistor, omdat deze wat robuuster zijn. Zij werken bij een spanning van 3 tot 35 V en kunnen een stroomsterkte van 1 A aan. In de elektronica-winkel koop je er eentje met de code *BC517*.

  Een transistor heeft drie pootjes: de collector (C), de basis (B) en de emittor (E).
  - De collector sluit je via het device (bijv. motor of zoemer) op de plus aan. Eventueel is er nog een weerstand nodig om de stroomsterkte te beperken.
  - De basis sluit je via een 1 kΩ weerstand op een GPIO-pin van de arduino of raspberry aan.
  - De emittor sluit je rechtstreeks op de ground aan (belangrijk, anders gaat de transistor stuk).

  Is de transistor nog heel? Om daarachter te komen heb je een multimeter nodig die diodes kan doormeten. Met de volgende vijf metingen kun je het weten.
  - rode meetpen op de basis en zwarte meetpen op de emitter levert een waarde van ±800
  - rode meetpen op de basis en zwarte meetpen op de collector levert een waarde van ±1200
  - zwarte meetpen op de basis en rode meetpen op de emitter levert geen waarde
  - zwarte meetpen op de basis en rode meetpen op de collector levert geen waarde
  - rode meetpen op de collector en zwarte meetpen op de emitter levert geen waarde

- Diode
  Met een diode zorg je ervoor dat de stroom maar in één richting kan gaan.

Je gebruikt de diode meestal bij een zogenaamde H-brug, waarmee je een motor heen en weer kan laten bewegen.

Standaard interfaces

- W1 aansluiten
  W1 wordt ook wel *OneWire* genoemd. De pinnen van W1 heten DATA, VCC en GND. Wanneer je een device met *W1* aansluit, is een weerstand (4,7 kΩ) tussen de *VCC* en de *DATA* noodzakelijk. Zonder deze weerstand zal er geen communicatie zijn.

- I2C en SPI aansluiten
  I2C wordt ook wel *TWI* genoemd. De pinnen van *I2C* heten SDA, SCL, VCC en GND.
  De pinnen van *SPI* heten MOSI, MISO, CS (ook wel SS), CLK (ook wel SCK of SCL), VCC en GND.
  Bij *I2C* en *SPI* sluit je de pinnen van het board aan op de overeenkomstige pinnen van de arduino of raspberry. Let op dat je de *VCC* op het juist voltage aansluit (zie het item *Veilig werken*).

- UART aansluiten
  De pinnen van *UART* heten RX, TX, VCC en GND. De RX-pin van het board sluit je aan op de TX-pin van de arduino of raspberry. De TX-pin van het board sluit je aan op de RX-pin van de arduino of raspberry. Let op dat je de *VCC* op het juist voltage aansluit (zie het item *Veilig werken*).

Raspberry beheren

In dit deel staan verschillende acties in clusters gegroepeerd. Niet alle acties zijn nodig om met de *DeviceLib* library te kunnen werken. Voor een minimale installatie waarbij alle onderdelen van *DeviceLib* zullen werken, moeten de volgende kopjes worden uitgevoerd:

- Raspberry Pi Imager
- Het systeem up-to-date houden
- LAMP op de raspberry installeren
- De te gebruiken software
- Interfaces activeren
- *DeviceLib* installeren, bij de bcm-versie ook nog:
- Bcm2835 gpio library installeren
- De commando-geschiedenis wissen

Het laatste is echt nodig, omdat anders de mysql-commando's met passwords zichbaar blijven. Verder zijn er bij het installeren bestanden aangepast, waar onkundig gebruik veel ellende mee kan aanrichten.

Het besturingssysteem installeren

Raspberry Pi Imager

Het voorbereiden van een SD-kaart voor een raspberry is het gemakkelijkst met het programma *Raspberry Pi imager*. Je kiest het standaard besturingssysteem (*Raspberry Pi OS (32-bit)*) en de SD-kaart en klikt vervolgens op *Schrijf*.

Wanneer de raspberry met een nieuwe SD-kaart opstart, moet deze nog worden ingesteld. Kies de Nederlandse locale en een password voor het zogenaamde *root*-account. (Het is belangrijk om een password voor het *root*-account in te vullen, omdat er bij verschillende acties naar gevraagd wordt. Een blanko password wordt niet geaccepteerd.) Vervolgens: Skip de *Update Software* pagina, omdat deze wel eens wil haperen. De update komt bij het volgende item.

Het systeem up-to-date houden

Om de allernieuwste wijzigingen in het systeem te installeren heb je de commando-prompt nodig. Dit is met name nodig voordat je nieuwe software installeert. Klik op *LXTerminal* en type de volgende instructies:

```
sudo apt-get update
sudo apt-get upgrade -y
```

Als een programma blijft crashen kan het volgende helpen:

```
sudo apt full-upgrade
sudo apt-get autoremove
```

Bcm2538 gpio library installeren

De *bcm2538* library geeft vanuit programma's directe toegang tot de gpio-pinnen van de raspberry. Normaal gesproken worden de pinnen via het file-systeem benaderd. Om de directe toegang van *bcm2835* te kunnen gebruiken, moeten de programma's met superuser-rechten worden gestart: *sudo ./programma*. Je installeert *bcm2538* als volgt:

Download de laatste versie (xx in de opdrachtregel hieronder) van de library via *https://www.airspayce.com/mikem/bcm2835/* en pak hem terplekke uit.
Type daarna in *LXTerminal*:

```
cd  ~/Downloads/bcm2835-1.xx
./configure
make
sudo make check
sudo make install
```

De commando-geschiedenis wissen

Linux is een vrij onbekend besturingssysteem, waardoor het lastig is om verkeerde handelingen te herstellen. Om te voorkomen dat leerlingen via *LXTerminal* kritische commando's gebruiken zonder te weten wat ze doen, kan de commando-geschiedenis worden gewist. Type in *LXTerminal*:

```
history -c
history -w
```

Hetzelfde geldt voor de geschiedenis van mysql (zie het item over *LAMP* verderop). Deze wordt gewist door het geschiedenis-bestand te verwijderen.

```
rm -rf ~/.mysql_history
sudo rm -rf /root/.mysql_history
```

Algemene beheertaken

De klok aanpassen

Zodra de raspberry op internet is aangesloten, zet deze automatisch de klok gelijk. Maar over het algemeen zal een raspberry stand-alone werken en dan is een rtc-board (real-time-clock) een oplossing. De volgende instelling gaan uit va een *DS3231* rtc-board. Type in de *LXTerminal*:

```
sudo nano /boot/config.txt
```

Wis het #-teken voor de volgende tekst of voeg deze toe:
dtparam=i2c_arm=on
dtoverlay=i2c-rtc,ds3231
Start de raspberry opnieuw op.

```
sudo date MMDDhhmmYYYY
sudo hwclock -systohc
sudo apt-get purge fake-hwclock
sudo nano /etc/udev/rules.d/85-hwclock.rules
```

Voeg aan het einde toe:
KERNEL==”rtc0”,RUN+=”/sbin/hwclock --rtc=$root/$name --hctosys"
Start de raspberry opnieuw op.

Na verloop van tijd zal de rtc-tijd afwijken van de werkelijke tijd. Dan zij de volgende instructies opnieuw nodig:

```
sudo date MMDDhhmmYYYY
sudo hwclock -systohc
```

Een gebruiker toevoegen

Type in *LXTerminal* het volgende:

```
sudo adduser gebruiker --home /home/gebruiker
sudo adduser gebruiker audio
sudo adduser gebruiker video
sudo adduser gebruiker gpio
sudo adduser gebruiker spi
sudo adduser gebruiker i2c
sudo adduser gebruiker plugdev
sudo adduser gebruiker netdev
sudo adduser gebruiker cdrom

sudo chmod 777 /dev/gpiomem
```

Handige commando's in *LXTerminal*

De verkenner in sudo starten:

```
sudo pcmanfm
```

Programma's beheren

Programma's starten en stoppen

Een programma starten:

```
sudo ./programma
```

Een programma stoppen:

```
pidof programma
sudo kill pid
```

Je kunt een programma ook starten als de raspberry wordt aangezet. Type in *LXTerminal* het volgende commando:

```
sudo nano /etc/rc.local
```

Voeg het volgende toe vóór 'exit 0':

sudo /pad/programma &

Het &-teken achter de regel zorgt ervoor dat het programma na het starten door blijft gaan.

Een programma toevoegen aan het systeem-menu

In *LXTerminal* type het volgende commando:

```
sudo nano /usr/share/applications/programma.desktop
```

Maak een nieuw bestand met:

[Desktop Entry]
Name=programma-naam-en
Name[nl]=programma-naam-nl
Comment=programma-hint-en
Comment[nl]=programma-hint-nl
GenericName=programma-type-en
GenericName[nl]=programma-type-nl
Exec=/pad/programma-executable %F
Icon=/pad/programma-icon

Terminal=false
StartupNotify=true

Een webserver (met database) opzetten

LAMP op de raspberry installeren

LAMP staat voor *Linux + Apache + MySql + Php*. Met deze samenstelling kun je een webserver bouwen die een sql-database ondersteunt. Het besturingssysteem van de raspberry is afgeleid van Linux. Apache is de webserver en op de raspberry krijg je MariaDB als databaseserver, wat de volledig vrije versie van MySql is. Vervolgens programmeer je de webpagina's in PhP, omdat deze code op de server-kant wordt verwerkt en dus toegang heeft tot de database [8].

LAMP wordt als volgt via *LXTerminal* geïnstalleerd:

```
sudo apt-get install apache2 -y
sudo apt-get install php -y
sudo apt-get install php-gd -y
sudo apt-get install php-mysql -y
sudo apt-get install mariadb-server -y
sudo apt-get install default-libmysqlclient-dev -y
```

Na het installeren moet de database nog worden ingesteld. Dit gebeurt via *LXTerminal* als volgt:

```
sudo mysql -u root -p
```

Geef het passwordt op dat je tijdens het installeren voor het *root*-account hebt gebruikt.

```
MariaDB [(none)]> grant all privileges on *.* to root@localhost
                                          identified by "password";
MariaDB [(none)]> create user admin@localhost identified by
                                                        "devlib";
MariaDB [(none)]> grant all privileges on *.* to admin@localhost
                                            identified by "devlib";
MariaDB [(none)]> flush privileges;
MariaDB [(none)]> exit;
```

Vanaf nu log je voortaan met het *admin*-account in.
Het *root*-account gebruik je alleen nog voor noodgevallen.

Belangrijk: Voer nu de paragraaf *De commando-geschiedenis wissen* uit. Het admin-password blijft anders in de commando-geschiedenis staan.

PhpMyAdmin op de raspberry

PhpMyAdmin is een database-administratie programma. Via *LXTerminal* installeer je het programma als volgt:

```
sudo apt-get install phpmyadmin -y
```

Kies voor de *apache2* webserver.
Laat *dbconfig-common* niet een database aanmaken om *phpmyadmin* gebruiksklaar te maken.

[8] Dit in tegenstelling tot bijvoorbeeld JavaScript, waarvan de code op client-kant wordt verwerkt en dus geen toegang heeft tot de database.

Je start het programma in een browser met *localhost/phpmyadmin* en logt in met het *admin*-account van *MariaDb*. Het kan zijn dat dit niet lukt, omdat *phpmyadmin* zich niet goed in het systeem heeft genesteld. Dit kun je als volgt handmatig herstellen:

```
sudo nano /etc/apache2/apache2.conf
```
Voeg aan het eind de volgende regel toe:
Include /etc/phpmyadmin/apache.conf
Start de raspberry opnieuw op.

Verder zit er een bug in phpMyAdmin, die gemakkelijk kan worden hersteld.

```
sudo pcmanfm
```
- dubbelklik op /usr/share/phpmyadmin/libraries/sql.lib.php
- plaats in regel 613 een '(' voor *count* en een ')' achter *]*, waarna deze er als volgt uit moet zien:
*|| **(**(count($analyzed_sql_results['select_expr']**)** == 1)*

Tenslotte: Kom nooit aan de tabellen *infomation_schema*, *mysql* en *performance_schema*. Wijzigingen aan deze tabellen zullen de database onherstelbaar vernielen.

Een CGI-server opzetten

Met een *CGI-server* omzeil *apache* en *php*. Een CGI-server is een gewoon programma. In een gewone commando-shell print het programma een tekst op het scherm, maar in de CGI-omgeving print hetzelfde programma de tekst via de *CGI-server* naar het netwerk. Met de *LXTerminal* wordt dit als volgt bereikt.

Geef *www-data* toegang tot GPIO via *LXTerminal*:
```
sudo usermod -a -G gpio www-data
sudo usermod -a -G i2c www-data
sudo usermod -a -G spi www-data
```

Activeer *CGI* via de verkenner met sudo-rechten:
```
sudo pcmanfm
```
Ga naar de map */etc/apache2/mods-available*.
Selecteer de bestanden *cgid.conf* en *cgid.load*.
Kies in het menu *> Bewerken > Maak koppeling*.
Selecteer de map */etc/apache2/mods-enabled*.
Start de raspberry opnieuw op.

Maak een programma (coderen, compileren, bouwen), zoals bijvoorbeeld:
```
#include <stdio.h>
void main( int argc, char *argv[])
{
   printf( "Content-type:text/html\r\n\r\n");
   printf( "<h1>Test</h1>\n");
   return 0;
}
```
Plaats de executable in de map */usr/lib/cgi-bin*. En maak deze via *LXTerminal* beschikbaar:
```
sudo nano /etc/apache2/conf-available/serve-cgi-bin.conf
```
Wijzig de regel *ScriptAlias /~~cgi-bin~~/ /usr/lib/cgi-bin/* in *ScriptAlias // /usr/lib/cgi-bin/*.
Start de raspberry opnieuw op.

Nu kun je je programma in een webbrowser starten met:
`host-adres/programma` (bijv: *10.0.0.1/mijnapp* of *localhost/mijnapp*)

De raspberry als wifi-punt

Je kun de raspberry als wifi access point opzetten. Daarvoor moet je twee programma's installeren: *hostapd* en *dnsmasq*. Met behulp van de *LXTerminal* dat als volgt.

```
sudo apt-get remove –purge hostapd
sudo apt-get install hostapd dnsmasq
```

```
sudo nano /etc/dnsmasq.conf
```

Voeg aan het einde toe:

interface=wlan0
dhcp-range=10.0.0.2,10.0.0.5,255.255.255.0,12h

```
sudo nano /etc/hostapd/hostapd.conf
```

Maak een nieuw bestand met:

interface=wlan0
hw_mode=g
channel=10
auth_algs=1
wpa=2
wpa_key_mgmt=WPA-PSK
wpa_pairwise=CCMP
rsn_pairwise=CCMP
wpa_passphrase=wachtwoord (bijv. 123gast!@#)
ssid=hostid (bijv. onzehost)
ieee80211n=1
wmm_enabled=1
ht_capab=[HT40] [SHORT-GI-20] [DSSS_CCK-40]

```
sudo nano /etc/network/interfaces
```

Voeg aan het einde toe:

allow-hotplug wlan0
iface wlan0 inet static
address 10.0.0.1
netmask 255.255.255.0
network 10.0.0.0
broadcast 10.0.0.255

Maak eventueel van de volgende regel commentaar door er het #-teken voor te zetten:

#wpa-conf /etc/wpa_supplicant/wpa_supplicant.conf

```
sudo nano /etc/default/hostapd
```

Maak eventueel van de volgende regel commentaar door er het #-teken voor te zetten:

#DAEMON_CONF=""
#DAEMON_OPTS=""

```
sudo nano /etc/dhcpcd.conf
```

Voeg aan het einde toe:

denyinterfaces wlan0

```
sudo systemctl unmask hostapd
sudo systemctl enable hostapd
sudo systemctl enable dnsmasq
```

```
sudo service hostapd start
sudo service dnsmasq start
```
Start de raspberry opnieuw op.

Media-software installeren

De te gebruiken software

Om alle mogelijkheden van de *DeviceLib*-library te benutten, is het nodig om de volgende software te installeren: Gtk, Espeak en OpenCv. Dit doe je als volgt via *LXTerminal*:

```
sudo apt-get install libgtk-3-dev -y
sudo apt-get install espeak -y
sudo apt-get install libopencv-dev -y
```

Espeak is een spraaksynthesizer die geschreven tekst uitspreekt en met *OpenCv* kun je beeldbewerking programmeren.

Je kunt *Espeak* testen met de volgende instructie:

```
espeak "Hello world" --stdout | aplay
```

Bluetooth-koptelefoon/speaker

Je hebt *BlueAlsa* nodig om geluid over bluetooth te streamen. Je hebt *OmxPlater* nodig om mp3-bestanden af te spelen. Je installeert de programma's als volgt via *LXTerminal* [9] :

```
sudo apt-get install bluealsa -y
sudo apt-get install omxplayer -y
```

Het koppelen van een BT-koptelefoon/speaker gaat zo (*XX:XX:XX:XX:XX:XX* staat voor het mac-adres van het BT-apparaat):

```
sudo bluetoothctl
```
(Alle beschikbare BT-apparaten worden getoond.)
```
[bluetooth]# scan on
[bluetooth]# pair XX:XX:XX:XX:XX:XX
[bluetooth]# trust XX:XX:XX:XX:XX:XX
[bluetooth]# connect XX:XX:XX:XX:XX:XX
[bluetooth]# exit
```

Audio-commando's in *LXTerminal*

Een wav-bestand afspelen:
```
aplay bestand.wav
omxplayer bestand.wav
```

Een mp3-bestand afspelen:
```
omxplayer bestand.mp3
```

[9] Vanaf het besturingssysteem *Buster* zijn beide programma's al voorgeïnstalleerd.

Audio output-device kiezen:

Automatisch: `amixer cset numid=3 0`
Audio-jack: `amixer cset numid=3 1`
HDMI: `amixer cset numid=3 2`

Interfaces activeren

SPI, I2C en W1 (onewire) aanzetten

Kies in het systeemmenu:
Voorkeuren > Raspberry pi Configuratie programma
Vink op het tabblad *Interfaces* de volgende items aan:
SPI, *I2C* en *1-Wire*
Start de raspberry opnieuw op.

W1 gebruiksklaar maken

Gebruik *LXTerminal* om een w1 bij het systeem bekend te maken.

```
sudo nano /boot/config.txt
```

Wis het #-teken voor de volgende tekst of voeg deze toe:
dtparam=w1-gpio=on
dtoverlay=w1-gpio,gpiopin=17

Start de raspberry opnieuw op.

```
sudo modprobe w1-gpio
sudo modprobe w1-therm
```

Devices die gebruik maken van W1

Dallas DS18… thermometers.
Sluit de thermometer aan en type vervolgens in de *LXTerminal* de commando's:

```
cd /sys/bus/w1/devices
dir
cd 28-xxxxxxxxxxxx
cat w1_slave
```

I-buttons.

```
sudo nano /etc/rc.local
```

Voeg toe vóór *exit(0)*:
sh /etc/rc.sh

```
sudo nano /etc/rc.sh
```

Dit is een nieuw bestand met:
chmod a+w /sys/devices/w1_bus_master1/w1_master_remove