

Sortiermaschine

Erzeugt von Doxygen 1.9.5

1 Sortiermaschine	1
1.1 Zur Dokumentation	1
1.1.1 Klassen	1
2 Hierarchie-Verzeichnis	1
2.1 Klassenhierarchie	1
3 Klassen-Verzeichnis	2
3.1 Auflistung der Klassen	2
4 Datei-Verzeichnis	3
4.1 Auflistung der Dateien	3
5 Klassen-Dokumentation	3
5.1 AnimatableLcd Klassenreferenz	3
5.1.1 Ausführliche Beschreibung	3
5.1.2 Dokumentation der Elementfunktionen	5
5.1.3 Dokumentation der Datenelemente	9
5.2 AnimString Klassenreferenz	10
5.2.1 Ausführliche Beschreibung	10
5.2.2 Beschreibung der Konstruktoren und Destruktoren	11
5.2.3 Dokumentation der Elementfunktionen	11
5.2.4 Dokumentation der Datenelemente	12
5.3 ButtonHandler Klassenreferenz	13
5.3.1 Ausführliche Beschreibung	13
5.3.2 Beschreibung der Konstruktoren und Destruktoren	13
5.3.3 Dokumentation der Elementfunktionen	14
5.3.4 Dokumentation der Datenelemente	14
5.4 Callable Klassenreferenz	15
5.4.1 Ausführliche Beschreibung	15
5.4.2 Beschreibung der Konstruktoren und Destruktoren	15
5.4.3 Dokumentation der Elementfunktionen	15
5.5 CallHandler Klassenreferenz	16
5.5.1 Ausführliche Beschreibung	16
5.5.2 Dokumentation der Elementfunktionen	17
5.5.3 Dokumentation der Datenelemente	20
5.6 CustomServo Klassenreferenz	21
5.6.1 Ausführliche Beschreibung	21
5.6.2 Dokumentation der Elementfunktionen	22
5.6.3 Dokumentation der Datenelemente	27
5.7 FuncCall Strukturreferenz	29
5.7.1 Ausführliche Beschreibung	29
5.7.2 Beschreibung der Konstruktoren und Destruktoren	29
5.7.3 Dokumentation der Elementfunktionen	30

5.7.4 Dokumentation der Datenelemente	31
5.8 LcdDotAnim Klassenreferenz	31
5.8.1 Ausführliche Beschreibung	31
5.8.2 Beschreibung der Konstruktoren und Destruktoren	32
5.8.3 Dokumentation der Elementfunktionen	32
5.9 LcdLoadingAnim Klassenreferenz	33
5.9.1 Ausführliche Beschreibung	33
5.9.2 Dokumentation der Elementfunktionen	34
5.10 LcdString Strukturreferenz	36
5.10.1 Ausführliche Beschreibung	36
5.10.2 Beschreibung der Konstruktoren und Destruktoren	37
5.10.3 Dokumentation der Elementfunktionen	37
5.10.4 Dokumentation der Datenelemente	38
6 Datei-Dokumentation	39
6.1 animLcd.h-Dateireferenz	39
6.1.1 Ausführliche Beschreibung	39
6.1.2 Variablen-Dokumentation	40
6.2 animLcd.h	40
6.3 animLcd.ino-Dateireferenz	40
6.3.1 Ausführliche Beschreibung	40
6.3.2 Variablen-Dokumentation	41
6.4 animLcd.ino	42
6.5 animString.h-Dateireferenz	43
6.5.1 Ausführliche Beschreibung	43
6.6 animString.h	44
6.7 animString.ino-Dateireferenz	45
6.7.1 Ausführliche Beschreibung	45
6.8 animString.ino	46
6.9 callHandler.h-Dateireferenz	47
6.9.1 Ausführliche Beschreibung	47
6.10 callHandler.h	48
6.11 callHandler.ino-Dateireferenz	49
6.11.1 Ausführliche Beschreibung	49
6.12 callHandler.ino	50
6.13 customServo.h-Dateireferenz	50
6.13.1 Ausführliche Beschreibung	50
6.14 customServo.h	51
6.15 customServo.ino-Dateireferenz	51
6.15.1 Ausführliche Beschreibung	51
6.16 customServo.ino	52
6.17 header.h-Dateireferenz	53

6.17.1 Ausführliche Beschreibung	53
6.17.2 Dokumentation der benutzerdefinierten Typen	53
6.18 header.h	54
6.19 index.md-Dateireferenz	54
6.20 sketch.ino-Dateireferenz	54
6.20.1 Ausführliche Beschreibung	54
6.20.2 Makro-Dokumentation	56
6.20.3 Dokumentation der Aufzählungstypen	56
6.20.4 Dokumentation der Funktionen	57
6.20.5 Variablen-Dokumentation	64
6.21 sketch.ino	68
Index	71

1 Sortiermaschine

Hallo, dies ist die Dokumentation für den Code der Sortiermaschine von Johannes und Arne
 Online Dokumentation: <https://arnecode.github.io/Sortiermaschine/index.html>

1.1 Zur Dokumentation

Ein guter Ort um zu starten ist die [sketch.ino](#) Datei. Dort wird die Logik des gesamten Programms zusammengeführt. Für eine Liste aller Dateien bitte im Menü unter Dateien nachschauen.

1.1.1 Klassen

Für Liste aller Klassen bitte im Menü unter Klassen nachschauen. Ein paar wichtige Klassen in diesem Projekt sind:

- [AnimatableLcd](#) - ermöglicht es Animationen auf dem Lcd-Display anzuzeigen
- [CallHandler](#) - lässt Calls nacheinander laufen
- [CustomServo](#) - Servo, bei dem die Geschwindigkeit gesteuert werden kann

2 Hierarchie-Verzeichnis

2.1 Klassenhierarchie

Die Liste der Ableitungen ist -mit Einschränkungen- alphabetisch sortiert:

ButtonHandler	13
Callable	15

FuncCall	29
LcdString	36
AnimString	10
LcdDotAnim	31
LcdLoadingAnim	33
CallHandler	16
LiquidCrystal_I2C	
AnimatableLcd	3
Servo	
CustomServo	21

3 Klassen-Verzeichnis

3.1 Auflistung der Klassen

Hier folgt die Aufzählung aller Klassen, Strukturen, Varianten und Schnittstellen mit einer Kurzbeschreibung:

AnimatableLcd	
Eigener Lcd, ermöglicht es Animationen auf dem Lcd Display anzuzeigen	3
AnimString	
Die Klasse für animierbare LcdStrings	10
ButtonHandler	
Kleine Klasse die Knopfdrucke verarbeitet	13
Callable	
Ein Call der vom CallHandler aufgerufen werden kann	15
CallHandler	
Klasse, die Calls nacheinander aufruft	16
CustomServo	
Eine Eigene Servo-Klasse, die es ermöglicht den Servo mit verschiedenen Geschwindigkeiten zu bewegen	21
FuncCall	
Ein Call der eine Funktion ausführt	29
LcdDotAnim	
Die Klasse der Lcd Punktanimationen	31
LcdLoadingAnim	
Die Klasse der Lcd Ladeanimationen	33
LcdString	
Ein String der auf dem AnimatableLcd angezeigt werden kann	36

4 Datei-Verzeichnis

4.1 Auflistung der Dateien

Hier folgt die Aufzählung aller Dateien mit einer Kurzbeschreibung:

animLcd.h	Header-Datei für den animierbaren lcd (AnimatableLcd)	39
animLcd.ino	Implementation für die AnimatableLcd Klasse	40
animString.h	Header datei für eine Mehrzahl von animierbaren Strings und der Callable Klasse	43
animString.ino	Implementationen der Callable und LcdString Klassen	45
callHandler.h	Header datei für den CallHandler	47
callHandler.ino	Umsetzung der CallHandler Klasse	49
customServo.h	Header Datei der CustomServo Klasse	50
customServo.ino	Umsetzung der CustomServo Klasse	51
header.h	Definiert variablen-types die überall im Programm benutzt werden	53
sketch.ino	Hauptdatei, wichtigste Funktionen sind setup() und loop()	54

5 Klassen-Dokumentation

5.1 AnimatableLcd Klassenreferenz

5.1.1 Ausführliche Beschreibung

Eigener Lcd, ermöglicht es Animationen auf dem Lcd Display anzuzeigen.

Definiert in Zeile 17 der Datei [animLcd.h](#).

```
#include <animLcd.h>
```

Zusammengehörigkeiten von AnimatableLcd:



Öffentliche Methoden

- void **setAnimation** (**AnimString** * _animString)
Setzt die aktuelle Animation.
- void **printCentered** (String text, int length=-1, int row=0)
Gibt einen String zentriert auf dem Lcd-Display aus.
- void **printPretty** (String text)
gibt den Text "schön" aus, das heißt zentriert und mit automatischen Zeilenumbrüchen
- void **update** ()
*wird immer wieder von **loop()** aufgerufen um die Animationen zu updaten*
- void **init** ()
Überschreibt die normale lcd init function.
- void **print** (const String &text)
Eigene Lcd-print funktion, die die Möglichkeit bietet eigene Characters in den Text einzufügen.

Öffentliche Attribute

- bool **doAnimation** = false
Gibt an, ob der Monitor animiert werden soll.

Private Attribute

- **AnimString** * **animString**
Die zurzeit laufende Animation.

5.1.2 Dokumentation der Elementfunktionen

5.1.2.1 `init()` `void AnimatableLcd::init ()`

Überschreibt die normale lcd init function.

Definiert in Zeile 43 der Datei `animLcd.ino`.

```
00044 {
00045     LiquidCrystal_I2C::init();
00046     backlight();
00047     noCursor();
00048     lcd.createChar(0, loading_empty_c);
00049     lcd.createChar(1, loading_full_c);
00050 }
```

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



5.1.2.2 `print()` `void AnimatableLcd::print (` `const String & text)`

Eigene Lcd-print funktion, die die Möglichkeit bietet eigene Characters in den Text einzufügen.

Für eigene Character einfach die nummer des Characters in den Text einfügen (**\1n für den nten Character**), \1 für Leerzeichen, das nicht in Zeilenumbruch resultiert

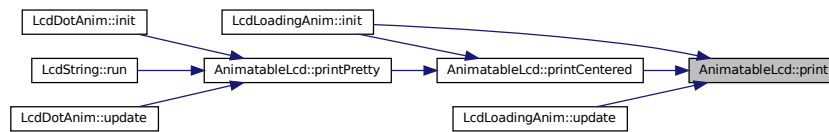
Parameter

<code>text</code>	
-------------------	--

Definiert in Zeile 66 der Datei `animLcd.ino`.

```
00067 {
00068     //custom print with ability to use custom characters, just insert the number of the custom
    character in the string (\1n for the nth character)
00069     //and it will be converted to the custom character (\1n so that \0 doesn't appear in the string,
    because it means end of string)
00070     for(char c:text){
00071         if(c>=8&&c<=15){//if it is a custom character
00072             write(c-8);
00073         }else if(c==1){//defining a non-newline space
00074             LiquidCrystal_I2C::print(" ");
00075         }
00076         else{
00077             LiquidCrystal_I2C::print(c);
00078         }
00079     }
00080 }
```


Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



5.1.2.3 printCentered() `void AnimatableLcd::printCentered (`
`String text,`
`int length = -1,`
`int row = 0)`

Gibt einen String zentriert auf dem Lcd-Display aus.

Parameter

<i>text</i>	
<i>length</i>	Länge des Textes, wird neu berechnet wenn Nichts angegeben
<i>row</i>	Zeile in der der Text ausgegeben werden soll

Definiert in Zeile 88 der Datei [animLcd.ino](#).

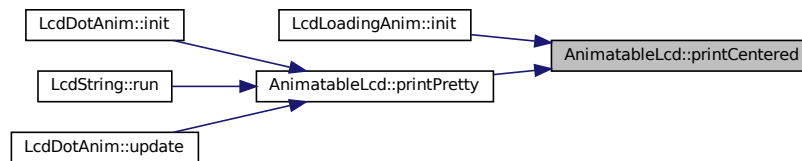
```

00089 {
00090   if (length == -1) {
00091     length = text.length();
00092   }
00093   int offset = (16 - length) / 2; //rundet immer ab, da int
00094   setCursor(offset, row);
00095   print(text);
00096 }
  
```

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



5.1.2.4 printPretty() void AnimatableLcd::printPretty (String text)

gibt den Text "schön" aus, das heißt zentriert und mit automatischen Zeilenumbrüchen

Parameter

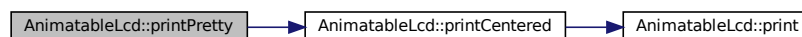
<i>text</i>	
-------------	--

Definiert in Zeile 102 der Datei [animLcd.ino](#).

```

00103 {
00104   clear();
00105   int length = text.length();
00106   if (length <= 16) {
00107     printCentered(text, length);
00108     return 0;
00109   }
00110   int spacePos = -1;
00111   for (int i = 15; i >= 0; i--) {
00112     if (text[i] == ' ') {
00113       spacePos = i;
00114       break;
00115     }
00116   }
00117   String row1, row2;
00118   if (spacePos != -1) {
00119     row1 = text.substring(0, spacePos);
00120     row2 = text.substring(spacePos + 1);
00121   } else {
00122     row1 = text.substring(0, 16);
00123     row2 = text.substring(16);
00124   }
00125   printCentered(row1, row1.length(), 0);
00126   printCentered(row2, row2.length(), 1);
00127 }
  
```

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



5.1.2.5 setAnimation() `void AnimatableLcd::setAnimation (AnimString * _animString)`

Setzt die aktuelle Animation.

Parameter

<code>_animString</code>	
--------------------------	--

Definiert in Zeile 56 der Datei `animLcd.ino`.

```

00057 {
00058     doAnimation = true;
00059     animString = _animString;
00060 }
  
```

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



5.1.2.6 update() `void AnimatableLcd::update ()`

wird immer wieder von `loop()` aufgerufen um die Animationen zu updaten

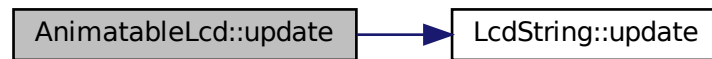
Definiert in Zeile 132 der Datei `animLcd.ino`.

```

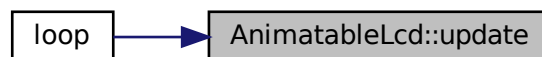
00133 {
  
```

```
00134     if (!doAnimation) {  
00135         return;  
00136     }  
00137     animString->update();  
00138 }
```

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



5.1.3 Dokumentation der Datenelemente

5.1.3.1 animString `AnimString* AnimatableLcd::animString [private]`

Die zurzeit laufende Animation.

Definiert in Zeile 22 der Datei [animLcd.h](#).

5.1.3.2 doAnimation `bool AnimatableLcd::doAnimation = false`

Gibt an, ob der Monitor animiert werden soll.

Definiert in Zeile 28 der Datei [animLcd.h](#).

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- [animLcd.h](#)
- [animLcd.ino](#)

5.2 AnimString Klassenreferenz

5.2.1 Ausführliche Beschreibung

Die Klasse für animierbare LcdStrings.

Wird nie selbst instanziiert aber Lcd Animationen erben von dieser Klasse

Definiert in Zeile 76 der Datei [animString.h](#).

```
#include <animString.h>
```

Zusammengehörigkeiten von AnimString:



Öffentliche Methoden

- virtual `~AnimString()`
- virtual void `init()`
- void `run()`
 - setzt Variablen die für alle Animationen notwendig sind und ruft dann ihre eigenen `init` Funktionen auf
- `LcdString` (`String text`, `AnimatableLcd *lcd`, `time_t duration=0`)

Geschützte Attribute

- `time_t stepDuration`
 - wie lange ein Schritt der Animation dauert
- `time_t animStart`
 - wann die animation begann
- `time_t lastRefresh`
 - wann das letzte mal die Anzeige erneuert wurde

Weitere Geerbte Elemente

5.2.2 Beschreibung der Konstruktoren und Destruktoren

5.2.2.1 ~AnimString() `virtual AnimString::~~AnimString () [inline], [virtual]`

Definiert in Zeile 95 der Datei [animString.h](#).

```
00095 {}
```

5.2.3 Dokumentation der Elementfunktionen

5.2.3.1 init() `virtual void AnimString::init () [inline], [virtual]`

Erneute Implementation in [LcdLoadingAnim](#) und [LcdDotAnim](#).

Definiert in Zeile 96 der Datei [animString.h](#).

```
00096 {}
```

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



5.2.3.2 LcdString() `LcdString::LcdString (String text, AnimatableLcd * lcd, time_t duration = 0) [inline]`

Definiert in Zeile 65 der Datei [animString.h](#).

```
00066 : text(text), duration(duration), lcd(lcd) { }
```

5.2.3.3 run() void AnimString::run () [virtual]

setzt Variablen die für alle Animationen notwendig sind und ruft dann ihre eigenen init Funktionen auf

Erneute Implementation von [LcdString](#).

Definiert in Zeile 52 der Datei [animString.ino](#).

```
00053 {
00054     callStart = millis();
00055     lcd->clear();
00056     lcd->setAnimation(this);
00057     animStart = millis();
00058     lastRefresh = millis();
00059     init();
00060 }
```

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



5.2.4 Dokumentation der Datenelemente

5.2.4.1 animStart time_t AnimString::animStart [protected]

wann die animation begann

Definiert in Zeile 87 der Datei [animString.h](#).

5.2.4.2 lastRefresh time_t AnimString::lastRefresh [protected]

wann das letzte mal die Anzeige erneuert wurde

Definiert in Zeile 92 der Datei [animString.h](#).

5.2.4.3 stepDuration time_t AnimString::stepDuration [protected]

wie lange ein Schritt der Animation dauert

Definiert in Zeile 82 der Datei [animString.h](#).

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- [animString.h](#)
- [animString.ino](#)

5.3 ButtonHandler Klassenreferenz

5.3.1 Ausführliche Beschreibung

Kleine Klasse die Knopfdrücke verarbeitet.

Definiert in Zeile 109 der Datei [sketch.ino](#).

Öffentliche Methoden

- [ButtonHandler](#) ()
Erstellt ein neues [ButtonHandler](#) Objekt.
- [ButtonHandler](#) (int [pin](#), void(*[onclick](#)())())
void [update](#) ()
Prüft, ob der Knopf gedrückt/losgelassen wurde.

Öffentliche Attribute

- void(* [onclick](#))()
Die Funktion die bei einem Klick, d.h. einem Drücken und loslassen des Knopfes ausgeführt wird.

Private Attribute

- int [pin](#)
Der Pin an dem der Knopf angeschlossen ist.
- bool [isPressed](#) = false
Gibt an, ob der Knopf momentan Gedrückt ist.

5.3.2 Beschreibung der Konstruktoren und Destruktoren

5.3.2.1 ButtonHandler() [1/2] `ButtonHandler::ButtonHandler () [inline]`

Definiert in Zeile 126 der Datei [sketch.ino](#).
00126 {}

5.3.2.2 ButtonHandler() [2/2] `ButtonHandler::ButtonHandler (int pin, void(*)() onclick) [inline]`

Erstellt ein neues [ButtonHandler](#) Objekt.

Parameter

<i>pin</i>	Der Pin an dem der Knopf angeschlossen ist
<i>onclick</i>	Die Funktion die bei einem Klick, d.h. einem Drücken und loslassen des Knopfes ausgeführt wird

Definiert in Zeile 133 der Datei [sketch.ino](#).

```
00133                                     : pin(pin), onclick onclick) {
00134     pinMode(pin, INPUT_PULLUP);
00135 }
```

5.3.3 Dokumentation der Elementfunktionen**5.3.3.1 update()** `void ButtonHandler::update () [inline]`

Prüft, ob der Knopf gedrückt/losgelassen wurde.

Wird von [loop\(\)](#) aufgerufen und ruft die [ButtonHandler::onclick](#) Funktion auf, wenn ein Klick festgestellt wurde

Definiert in Zeile 140 der Datei [sketch.ino](#).

```
00141 {
00142     bool isPressedNew = digitalRead(pin) == HIGH;
00143     if (isPressedNew != isPressed) { //is not being pressed now, but was being pressed
00144         if (isPressed) {
00145             Serial.println("click");
00146             onclick();
00147         }
00148     }
00149     isPressed = isPressedNew;
00150 }
```

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:

**5.3.4 Dokumentation der Datenelemente****5.3.4.1 isPressed** `bool ButtonHandler::isPressed = false [private]`

Gibt an, ob der Knopf momentan Gedrückt ist.

Definiert in Zeile 119 der Datei [sketch.ino](#).

5.3.4.2 onclick `void(* ButtonHandler::onclick) ()`

Die Funktion die bei einem Klick, d.h. einem Drücken und loslassen des Knopfes ausgeführt wird.

Definiert in Zeile 125 der Datei [sketch.ino](#).

5.3.4.3 pin `int ButtonHandler::pin [private]`

Der Pin an dem der Knopf angeschlossen ist.

Definiert in Zeile 114 der Datei [sketch.ino](#).

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

- [sketch.ino](#)

5.4 Callable Klassenreferenz

5.4.1 Ausführliche Beschreibung

Ein Call der vom [CallHandler](#) aufgerufen werden kann.

Wird nie selbst instanziiert sondern nur Abgeleitete Klassen

Definiert in Zeile 24 der Datei [animString.h](#).

```
#include <animString.h>
```

Private Methoden

- virtual void [run](#) ()
- virtual bool [isDone](#) ()
- virtual [~Callable](#) ()

5.4.2 Beschreibung der Konstruktoren und Destruktoren

5.4.2.1 [~Callable\(\)](#) `virtual Callable::~~Callable () [inline], [private], [virtual]`

Definiert in Zeile 27 der Datei [animString.h](#).

```
00027 {} //let's derived classes free their own memory. ~functions are called when the object is deleted
```

5.4.3 Dokumentation der Elementfunktionen

5.4.3.1 isDone() `virtual bool Callable::isDone () [inline], [private], [virtual]`

Erneute Implementation in [FuncCall](#) und [LcdString](#).

Definiert in Zeile 26 der Datei [animString.h](#).

```
00026 {}
```

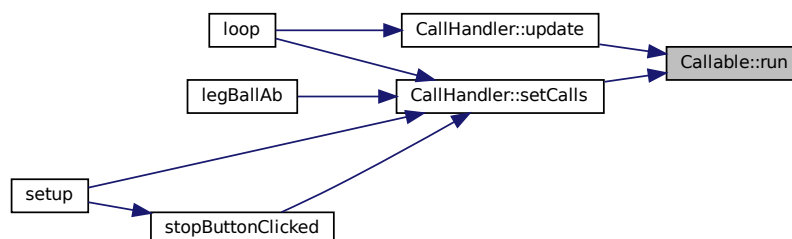
5.4.3.2 run() `virtual void Callable::run () [inline], [private], [virtual]`

Erneute Implementation in [FuncCall](#), [LcdString](#) und [AnimString](#).

Definiert in Zeile 25 der Datei [animString.h](#).

```
00025 {} //virtual->must be implemented by derived classes
```

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

- [animString.h](#)

5.5 CallHandler Klassenreferenz

5.5.1 Ausführliche Beschreibung

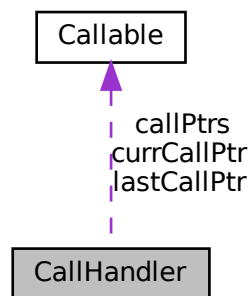
Klasse, die Calls nacheinander aufruft.

Ermöglicht es Calls wie z.B. Funktionen nacheinander aufzurufen, ohne die `delay()` Funktion zu verwenden

Definiert in Zeile 17 der Datei [callHandler.h](#).

```
#include <callHandler.h>
```

Zusammengehörigkeiten von CallHandler:



Öffentliche Methoden

- void `deleteCalls` ()
setzt den Speicherplatz der von den Calls besetzt wurde frei
- void `setCalls` (`Callable` *newCallPtrs[], size_t nCalls)
Setzt die neuen Calls, die ausgeführt werden sollen.
- void `update` ()
Wechselt zum nächsten Call, wenn der Aktuelle vorbei ist und aktualisiert den jetzigen (z.B. animationen)

Öffentliche Attribute

- bool `running` = false
Gibt an, ob der `CallHandler` fertig ist.

Private Attribute

- `Callable` ** `callPtrs`
Die liste der aktuellen Calls.
- `Callable` ** `currCallPtr`
Der Call der zurzeit ausgeführt wird.
- `Callable` ** `lastCallPtr`
Der letzte Call.
- `time_t` `lastCallT`
Der Zeitpunkt an dem der letzte Call ausgeführt wurde.
- bool `callsSet` = false
Sagt aus, ob `CallHandler::callPtrs` zu einer gültigen Speicheradresse zeigt.

5.5.2 Dokumentation der Elementfunktionen

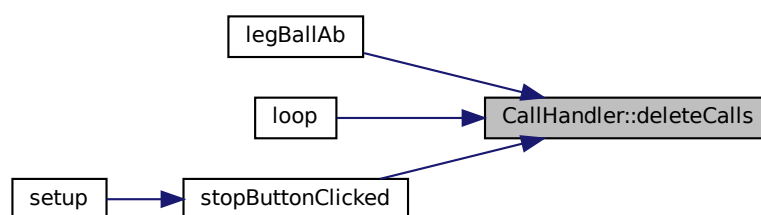
5.5.2.1 deleteCalls() `void CallHandler::deleteCalls ()`

setzt den Speicherplatz der von den Calls besetzt wurde frei

Definiert in Zeile 12 der Datei `callHandler.ino`.

```
00013 {
00014     if (!callsSet) {
00015         return;
00016     }
00017     callsSet = false;
00018     for (Callable** callPtr = callPtrs; callPtr <= lastCallPtr; callPtr++) {
00019         delete *callPtr;
00020     }
00021     delete callPtrs;
00022 }
```

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



5.5.2.2 setCalls() `void CallHandler::setCalls (` `Callable * newCallPtrs[],` `size_t nCalls)`

Setzt die neuen Calls, die ausgeführt werden sollen.

Calls werden im **Heap** gespeichert um sie zwischen Funktionen hin- und hergeben zu können und sie benutzen nachdem die Exekution abgeschlossen ist (bzw an das `CallHandler` Objekt)

Warnung

`nCalls` darf auf keinen Fall größer als die tatsächliche Anzahl an Calls sein, sonst stürzt das Programm ab weil es versucht nicht vorhandene Calls auszuführen

Parameter

<code>newCallPtrs</code>	
<code>nCalls</code>	

Definiert in Zeile 30 der Datei `callHandler.ino`.

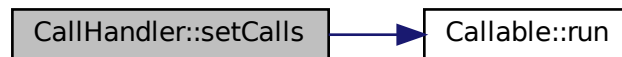
```
00031 {
00032     /*if(callsSet){ //doing this would result in two sets of calls being in heap at once
00033         deleteCalls(); //solution is to delete previous calls before initializing a new one
00034     }*/
```

```

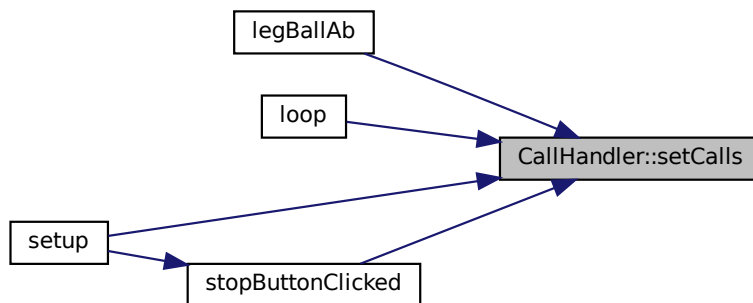
00035  callPtrs = newCallPtrs;
00036  callsSet = true;
00037  currCallPtr = callPtrs;
00038  (*currCallPtr)->run();
00039  lastCallPtr = callPtrs + nCalls - 1;
00040  lastCallT = millis();
00041  running = true;
00042  }

```

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



5.5.2.3 update() void CallHandler::update ()

Wechselt zum nächsten Call, wenn der Aktuelle vorbei ist und aktualisiert den jetzigen (z.B. animationen)

Wird von `loop()` aufgerufen

Definiert in Zeile 47 der Datei `callHandler.ino`.

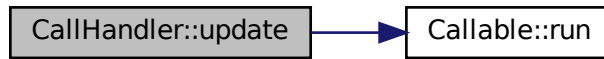
```

00048 {
00049   if (!running) {
00050     return;
00051   }
00052   time_t timePassed = millis() - lastCallT;
00053   if ((*currCallPtr)->isDone()) { /*->currCall->isDone()
00054     if (currCallPtr == lastCallPtr) {
00055       running = false;
00056       return;
00057     }
00058     currCallPtr++;
00059     (*currCallPtr)->run(); /*->currCall->run();
00060   }

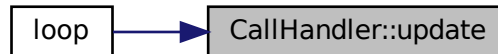
```

```
00061 }
```

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



5.5.3 Dokumentation der Datenelemente

5.5.3.1 `callPtrs` `Callable** CallHandler::callPtrs [private]`

Die liste der aktuellen Calls.

Wird mithilfe von `CallHandler::setCalls` gesetzt. Die Calls werden im Heap gespeichert, das heißt zum einen, dass sie zwischen Funktionen hin- und hergegeben werden können, zum anderen aber auch, dass sie mithilfe von `CallHandler::deleteCalls` manuell wieder gelöscht werden müssen

Definiert in Zeile 23 der Datei `callHandler.h`.

5.5.3.2 `callsSet` `bool CallHandler::callsSet = false [private]`

Sagt aus, ob `CallHandler::callPtrs` zu einer gültigen Speicheradresse zeigt.

Definiert in Zeile 44 der Datei `callHandler.h`.

5.5.3.3 currCallPtr `Callable** CallHandler::currCallPtr [private]`

Der Call der zurzeit ausgeführt wird.

Es handelt sich hierbei um einen Pointer-Pointer. Der Pointer zeigt zu einer Stelle in der `CallHandler::callPtrs` Liste, die wiederum zum tatsächlichen Call zeigt

Definiert in Zeile 29 der Datei `callHandler.h`.

5.5.3.4 lastCallPtr `Callable** CallHandler::lastCallPtr [private]`

Der letzte Call.

Wird benutzt um zu wissen, wann der letzte Call ausgeführt wurde

Definiert in Zeile 34 der Datei `callHandler.h`.

5.5.3.5 lastCallT `time_t CallHandler::lastCallT [private]`

Der Zeitpunkt an dem der letzte Call ausgeführt wurde.

Definiert in Zeile 39 der Datei `callHandler.h`.

5.5.3.6 running `bool CallHandler::running = false`

Gibt an, ob der `CallHandler` fertig ist.

Definiert in Zeile 50 der Datei `callHandler.h`.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- `callHandler.h`
- `callHandler.ino`

5.6 CustomServo Klassenreferenz

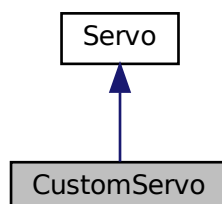
5.6.1 Ausführliche Beschreibung

Eine Eigene Servo-Klasse, die es ermöglicht den Servo mit verschiedenen Geschwindigkeiten zu bewegen.

Definiert in Zeile 13 der Datei `customServo.h`.

```
#include <customServo.h>
```

Zusammengehörigkeiten von CustomServo:



Öffentliche Methoden

- void `write` (short newAngle)
Bewegt den Servo mit einer vorher spezifizierten Geschwindigkeit.
- void `write` (short newAngle, `time_t` duration)
Bewegt den Servo in duration Millisekunden an den angegebenen Winkel.
- void `writeDirect` (short angle)
Steuert den Servo direkt an, entspricht dem normalen Servo::write.
- void `setSpeed` (float newSpeed)
Setzt eine neue Geschwindigkeit des Servos.
- void `updatePos` ()
Aktualisiert die Position des Servomotors.
- void `stop` ()
Stoppt den Servo.
- void `start` ()
Lässt den Servo weiterlaufen.
- bool `isDone` ()
Gibt an, ob der Servo angekommen ist.

Öffentliche Attribute

- bool `done` =true

Private Methoden

- void `startMove` ()
Setzt Variablen, die benötigt werden um den Servo zu bewegen.

Private Attribute

- short `startAngle`
Der Winkel an dem sich der Servo bei Start der Animation befand.
- short `targetAngle`
Der Zielwinkel.
- float `speed`
Die Geschwindigkeit des Servos in Grad pro Millisekunde.
- `time_t` `startTime`
Zeitpunkt an dem der Servo anfang sich zu bewegen (in Millisekunden)

5.6.2 Dokumentation der Elementfunktionen

5.6.2.1 isDone() `bool CustomServo::isDone ()`

Gibt an, ob der Servo angekommen ist.

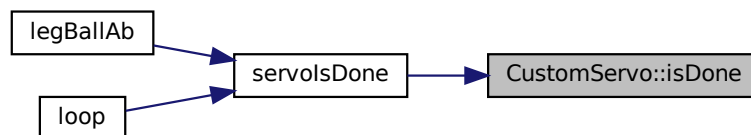
Rückgabe

true
false

Definiert in Zeile 91 der Datei `customServo.ino`.

```
00092 {  
00093     return read() == targetAngle;  
00094 }
```

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



5.6.2.2 setSpeed() `void CustomServo::setSpeed (float newSpeed)`

Setzt eine neue Geschwindigkeit des Servos.

Parameter

<code>newSpeed</code>	Die neue Geschwindigkeit in Grad pro Millisekunde
-----------------------	---

Definiert in Zeile 52 der Datei `customServo.ino`.

```
00053 {  
00054     startMove();  
00055     speed = newSpeed;  
00056 }
```

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



5.6.2.3 `start()` `void CustomServo::start ()`

Lässt den Servo weiterlaufen.

Definiert in Zeile 107 der Datei `customServo.ino`.

```
00108 {  
00109     done = false;  
00110 }
```

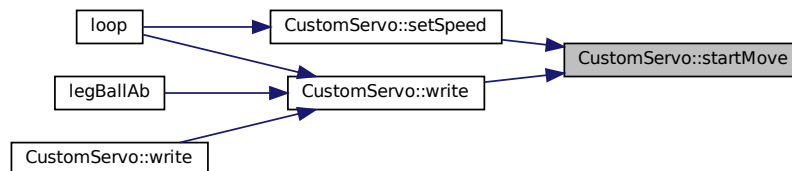
5.6.2.4 `startMove()` `void CustomServo::startMove () [private]`

Setzt Variablen, die benötigt werden um den Servo zu bewegen.

Definiert in Zeile 11 der Datei `customServo.ino`.

```
00012 {  
00013     startAngle = read();  
00014     startTime = millis();  
00015     done = false;  
00016 }
```

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



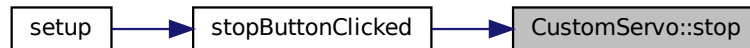
5.6.2.5 stop() void CustomServo::stop ()

Stoppt den Servo.

Definiert in Zeile 99 der Datei `customServo.ino`.

```
00100 {  
00101     done = true;  
00102 }
```

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



5.6.2.6 updatePos() void CustomServo::updatePos ()

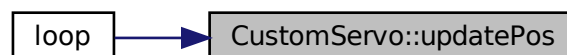
Aktualisiert die Position des Servomotors.

Wird von `loop()` aufgerufen

Definiert in Zeile 61 der Datei `customServo.ino`.

```
00062 {  
00063     if (done) {  
00064         return;  
00065     }  
00066     long timePassed = millis() - startTime;  
00067     short newAngle;  
00068     if (targetAngle > startAngle) {  
00069         newAngle = startAngle + timePassed * speed;  
00070         if (newAngle >= targetAngle) {  
00071             Servo::write(targetAngle);  
00072             done = true;  
00073             return;  
00074         }  
00075     } else {  
00076         newAngle = startAngle - timePassed * speed;  
00077         if (newAngle <= targetAngle) {  
00078             Servo::write(targetAngle);  
00079             done = true;  
00080             return;  
00081         }  
00082     }  
00083     Servo::write(newAngle);  
00084 }
```

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



5.6.2.7 write() [1/2] `void CustomServo::write (`
`short newAngle)`

Bewegt den Servo mit einer vorher spezifizierten Geschwindigkeit.

Parameter

<code>newAngle</code>	
-----------------------	--

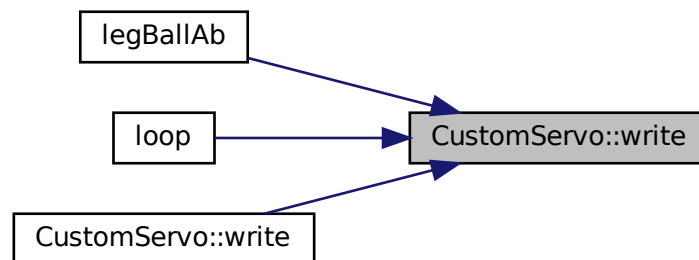
Definiert in Zeile 22 der Datei `customServo.ino`.

```
00023 {  
00024     startMove();  
00025     targetAngle = newAngle;  
00026 }
```

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



5.6.2.8 write() [2/2] `void CustomServo::write (`
`short newAngle,`
`time_t duration)`

Bewegt den Servo in duration Millisekunden an den angegebenen Winkel.

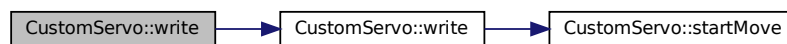
Parameter

<i>newAngle</i>	
<i>duration</i>	

Definiert in Zeile 33 der Datei [customServo.ino](#).

```
00034 {  
00035   write(newAngle);  
00036   speed = (float)(targetAngle - startAngle) / (float)duration;  
00037 }
```

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



5.6.2.9 writeDirect() void CustomServo::writeDirect (short angle)

Steuert den Servo direkt an, entspricht dem normalen Servo::write.

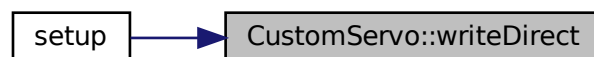
Parameter

<i>angle</i>	
--------------	--

Definiert in Zeile 43 der Datei [customServo.ino](#).

```
00044 {  
00045   Servo::write(angle);  
00046 }
```

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



5.6.3 Dokumentation der Datenelemente

5.6.3.1 done `bool CustomServo::done =true`

Definiert in Zeile [36](#) der Datei [customServo.h](#).

5.6.3.2 speed `float CustomServo::speed [private]`

Die Geschwindigkeit des Servos in Grad pro Millisekunde.

Definiert in Zeile [28](#) der Datei [customServo.h](#).

5.6.3.3 startAngle `short CustomServo::startAngle [private]`

Der Winkel an dem sich der Servo bei Start der Animation befand.

Definiert in Zeile [18](#) der Datei [customServo.h](#).

5.6.3.4 startTime `time_t CustomServo::startTime [private]`

Zeitpunkt an dem der Servo anfang sich zu bewegen (in Millisekunden)

Definiert in Zeile [33](#) der Datei [customServo.h](#).

5.6.3.5 targetAngle `short CustomServo::targetAngle [private]`

Der Zielwinkel.

Definiert in Zeile [23](#) der Datei [customServo.h](#).

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- [customServo.h](#)
- [customServo.ino](#)

5.7 FuncCall Strukturreferenz

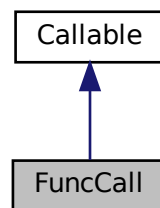
5.7.1 Ausführliche Beschreibung

Ein Call der eine Funktion ausführt.

Definiert in Zeile 33 der Datei `animString.h`.

```
#include <animString.h>
```

Zusammengehörigkeiten von FuncCall:



Öffentliche Methoden

- `FuncCall (func_t< void > call, func_t< bool > _isDone)`
- `FuncCall (func_t< void > call)`
- `virtual ~FuncCall ()`
- `void run ()`

ruft die angegebene Funktion auf

- `bool isDone ()`
Gibt zurück, ob der nächste Call ausgeführt werden sollte.

Öffentliche Attribute

- `func_t< void > call`
die Funktion die aufgerufen wird, wenn der Call an der Reihe ist
- `func_t< bool > _isDone`
bestimmt, ob dieser Call vorbei ist

5.7.2 Beschreibung der Konstruktoren und Destruktoren

5.7.2.1 FuncCall() [1/2] `FuncCall::FuncCall (`
 `func_t< void > call,`
 `func_t< bool > _isDone) [inline]`

Definiert in Zeile 44 der Datei [animString.h](#).

```
00044                                     :  
00045     call(call), _isDone(_isDone) { }
```

5.7.2.2 FuncCall() [2/2] `FuncCall::FuncCall (`
 `func_t< void > call) [inline]`

Definiert in Zeile 46 der Datei [animString.h](#).

```
00046                                     : //when no isDone function is provided, isDone defaults to true  
00047     call(call), _isDone([]() {return true;}) {}
```

5.7.2.3 ~FuncCall() `virtual FuncCall::~~FuncCall () [inline], [virtual]`

Definiert in Zeile 48 der Datei [animString.h](#).

```
00048 {}
```

5.7.3 Dokumentation der Elementfunktionen

5.7.3.1 isDone() `bool FuncCall::isDone () [virtual]`

Gibt zurück, ob der nächste Call ausgeführt werden sollte.

Rückgabe

 true

 false

Erneute Implementation von [Callable](#).

Definiert in Zeile 24 der Datei [animString.ino](#).

```
00025 {  
00026     return _isDone();  
00027 }
```

5.7.3.2 run() `void FuncCall::run () [virtual]`

ruft die angegebene Funktion auf

Erneute Implementation von [Callable](#).

Definiert in Zeile 14 der Datei [animString.ino](#).

```
00015 {  
00016     call();  
00017 }
```

5.7.4 Dokumentation der Datenelemente

5.7.4.1 `_isDone` `func_t<bool> FuncCall::_isDone`

bestimmt, ob dieser Call vorbei ist

Definiert in Zeile 43 der Datei [animString.h](#).

5.7.4.2 `call` `func_t<void> FuncCall::call`

die Funktion die aufgerufen wird, wenn der Call an der Reihe ist

Definiert in Zeile 38 der Datei [animString.h](#).

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Dateien:

- [animString.h](#)
- [animString.ino](#)

5.8 LcdDotAnim Klassenreferenz

5.8.1 Ausführliche Beschreibung

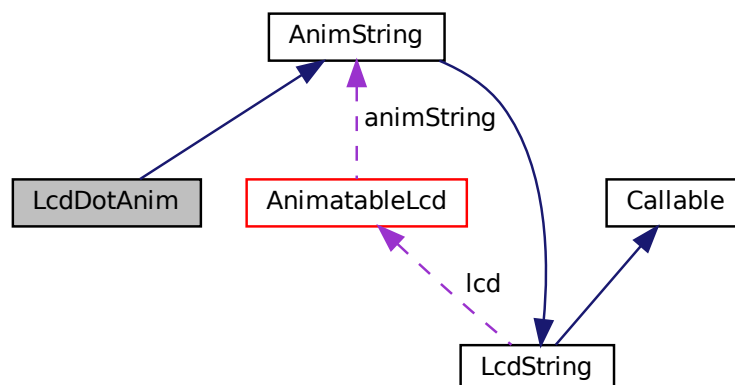
Die Klasse der Lcd Punktanimationen.

zeigt immer wieder keinen, dann einen, dann zwei, dann drei und letztendlich wieder keinen Punkt nach dem Text an

Definiert in Zeile 113 der Datei [animString.h](#).

```
#include <animString.h>
```

Zusammengehörigkeiten von LcdDotAnim:



Öffentliche Methoden

- `LcdDotAnim` (`String text`, `AnimatableLcd *lcd`, `time_t duration=0`, `time_t _stepDuration=500`)
- `void init ()`
initialisiert die Punktanimation
- `void update ()`
aktualisiert die Punktanimation, wird von `loop()` aufgerufen

Weitere Geerbte Elemente

5.8.2 Beschreibung der Konstruktoren und Destruktoren

5.8.2.1 LcdDotAnim() `LcdDotAnim::LcdDotAnim (`
`String text,`
`AnimatableLcd * lcd,`
`time_t duration = 0,`
`time_t _stepDuration = 500) [inline]`

Definiert in Zeile 115 der Datei `animString.h`.

```
00116 : AnimString(text, lcd, duration) {
00117     stepDuration = _stepDuration;
00118 }
```

5.8.3 Dokumentation der Elementfunktionen

5.8.3.1 init() `void LcdDotAnim::init () [virtual]`

initialisiert die Punktanimation

Erneute Implementation von `AnimString`.

Definiert in Zeile 103 der Datei `animString.ino`.

```
00104 {
00105     lcd->printPretty(text + "\1\1\1");//spaces that can't be broken up to newlines
00106 }
```

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



5.8.3.2 update() void LcdDotAnim::update () [virtual]

aktualisiert die Punktanimation, wird von [loop\(\)](#) aufgerufen

Erneute Implementation von [LcdString](#).

Definiert in Zeile 111 der Datei [animString.ino](#).

```
00112 {
00113     time_t time = millis();
00114     if ((time - lastRefresh) < stepDuration) {
00115         return;
00116     }
00117     lastRefresh = time;
00118     int numDots = ((time - animStart) / stepDuration) % 4;
00119     char dots[4];
00120     for (int i = 0; i < 3; i++) {
00121         if (i < numDots) {
00122             dots[i] = '.';
00123         } else {
00124             dots[i] = '\1';
00125         }
00126     }
00127     dots[3] = '\0';
00128     lcd->printPretty(text + dots);
00129 }
```

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- [animString.h](#)
- [animString.ino](#)

5.9 LcdLoadingAnim Klassenreferenz**5.9.1 Ausführliche Beschreibung**

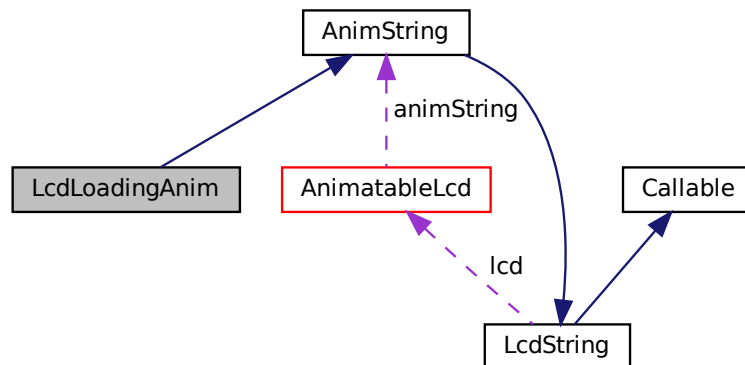
Die Klasse der Lcd Ladeanimationen.

Zeigt Acht Ladebalken und den Fortschritt in Prozent an

Definiert in Zeile 103 der Datei [animString.h](#).

```
#include <animString.h>
```

Zusammengehörigkeiten von LcdLoadingAnim:



Öffentliche Methoden

- void `init` ()
initialisierung der Ladeanimation
- void `update` ()
aktualisiert die Ladeanimation, wird von `loop()` aufgerufen

Weitere Geerbte Elemente

5.9.2 Dokumentation der Elementfunktionen

5.9.2.1 `init()` void LcdLoadingAnim::init () [virtual]

initialisierung der Ladeanimation

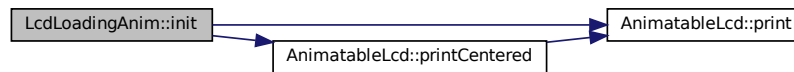
Erneute Implementation von [AnimString](#).

Definiert in Zeile 64 der Datei `animString.ino`.

```

00065 {
00066     stepDuration = duration / 9;
00067     if (text.length() > 16) {
00068         Serial.print("warning: text given for loading animation is too long, text: ");
00069         Serial.println(text);
00070     }
00071     lcd->printCentered(text);
00072     lcd->setCursor(LOADING_BAR_OFFSET, 1);
00073     for (int i = 0; i < 8; i++) {
00074         lcd->write(0);
00075     }
00076     lcd->print("0% ");
00077 }
  
```

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



5.9.2.2 update() void LcdLoadingAnim::update () [virtual]

aktualisiert die Ladeanimation, wird von `loop()` aufgerufen

Erneute Implementation von `LcdString`.

Definiert in Zeile 82 der Datei `animString.ino`.

```

00083 {
00084     time_t time = millis();
00085     short percent = (time - animStart) * 100 / duration;
00086     if (time - lastRefresh > stepDuration) {
00087         short nToFill = percent * 9 / 100;
00088         if (nToFill == 0) {
00089             return;
00090         }
00091         lcd->setCursor(nToFill + LOADING_BAR_OFFSET - 1, 1);
00092         lcd->write(1);
00093         lastRefresh = time;
00094     }
00095     lcd->setCursor(8 + LOADING_BAR_OFFSET, 1);
00096     lcd->print(percent);
00097     lcd->print("%");
00098 }
  
```

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- [animString.h](#)
- [animString.ino](#)

5.10 LcdString Strukturreferenz

5.10.1 Ausführliche Beschreibung

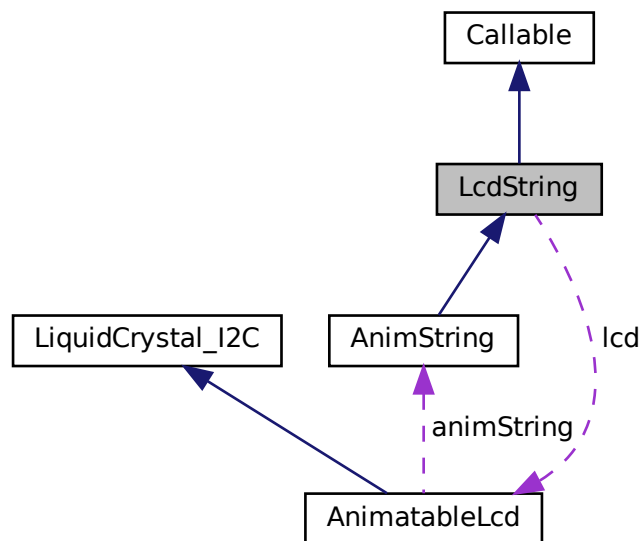
Ein String der auf dem [AnimatableLcd](#) angezeigt werden kann.

Animationen werden von dieser Klasse abgeleitet

Definiert in Zeile 56 der Datei [animString.h](#).

```
#include <animString.h>
```

Zusammengehörigkeiten von LcdString:



Öffentliche Methoden

- [LcdString](#) (String [text](#), [AnimatableLcd](#) *[lcd](#), time_t [duration](#)=0)
- virtual [~LcdString](#) ()
- bool [isDone](#) ()
Gibt zurück, ob die *duration* überschritten ist.
- virtual void [run](#) ()
gibt den String auf dem Lcd-Display aus
- virtual void [update](#) ()

Öffentliche Attribute

- String [text](#)
der Text der angezeigt wird
- [AnimatableLcd](#) * [lcd](#)
- time_t [duration](#)
- time_t [callStart](#)

5.10.2 Beschreibung der Konstruktoren und Destruktoren

5.10.2.1 LcdString() `LcdString::LcdString (`
 `String text,`
 `AnimatableLcd * lcd,`
 `time_t duration = 0) [inline]`

Definiert in Zeile 65 der Datei [animString.h](#).

```
00066 : text(text), duration(duration), lcd(lcd) { }
```

5.10.2.2 ~LcdString() `virtual LcdString::~~LcdString () [inline], [virtual]`

Definiert in Zeile 67 der Datei [animString.h](#).

```
00067 {}
```

5.10.3 Dokumentation der Elementfunktionen

5.10.3.1 isDone() `bool LcdString::isDone () [virtual]`

Gibt zurück, ob die duration überschritten ist.

Rückgabe

true

false

Erneute Implementation von [Callable](#).

Definiert in Zeile 44 der Datei [animString.ino](#).

```
00045 {  
00046     return millis() - callStart > duration;  
00047 }
```

5.10.3.2 run() `void LcdString::run () [virtual]`

gibt den String auf dem Lcd-Display aus

Erneute Implementation von [Callable](#).

Erneute Implementation in [AnimString](#).

Definiert in Zeile 32 der Datei [animString.ino](#).

```
00033 {  
00034     callStart = millis();  
00035     lcd->doAnimation = false;  
00036     lcd->printPretty(this->text);  
00037 }
```

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



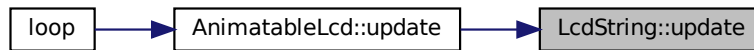
5.10.3.3 update() `virtual void LcdString::update () [inline], [virtual]`

Erneute Implementation in [LcdLoadingAnim](#) und [LcdDotAnim](#).

Definiert in Zeile 70 der Datei [animString.h](#).

00070 {}

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



5.10.4 Dokumentation der Datenelemente

5.10.4.1 callStart `time_t LcdString::callStart`

Definiert in Zeile 64 der Datei [animString.h](#).

5.10.4.2 duration `time_t LcdString::duration`

Definiert in Zeile 63 der Datei [animString.h](#).

5.10.4.3 lcd `AnimatableLcd* LcdString::lcd`

Definiert in Zeile 62 der Datei [animString.h](#).

5.10.4.4 text `String LcdString::text`

der Text der angezeigt wird

Definiert in Zeile 61 der Datei [animString.h](#).

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Dateien:

- [animString.h](#)
- [animString.ino](#)

6 Datei-Dokumentation

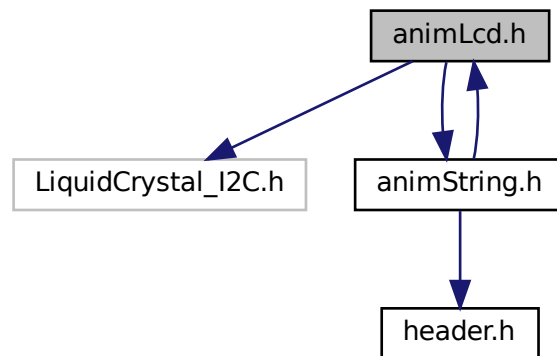
6.1 animLcd.h-Dateireferenz

6.1.1 Ausführliche Beschreibung

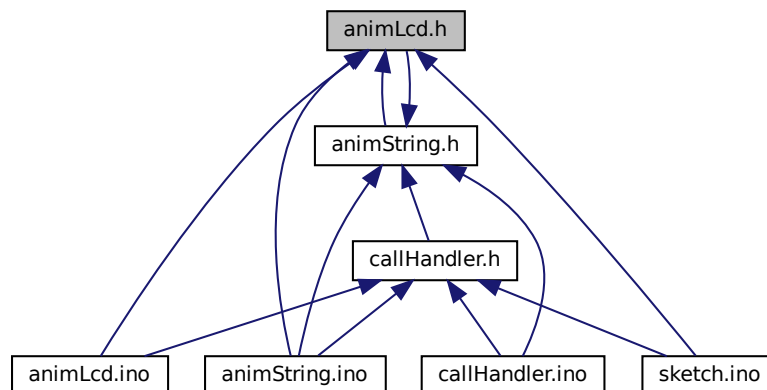
Header-Datei für den animierbaren lcd ([AnimatableLcd](#))

Definiert in Datei [animLcd.h](#).

```
#include <LiquidCrystal_I2C.h>
#include "animString.h"
Include-Abhängigkeitsdiagramm für animLcd.h:
```



Dieser Graph zeigt, welche Datei direkt oder indirekt diese Datei enthält:



Klassen

- class [AnimatableLcd](#)

Eigener Lcd, ermöglicht es Animationen auf dem Lcd Display anzuzeigen.

Variablen

- const int [LOADING_BAR_OFFSET](#) = 2

6.1.2 Variablen-Dokumentation

6.1.2.1 [LOADING_BAR_OFFSET](#) const int [LOADING_BAR_OFFSET](#) = 2

Definiert in Zeile [12](#) der Datei [animLcd.h](#).

6.2 animLcd.h

[gehe zur Dokumentation dieser Datei](#)

```
00001
00006 #ifndef ANIMLCD_H
00007 #define ANIMLCD_H
00008 #include <LiquidCrystal_I2C.h>
00009 class AnimatableLcd;
00010 #include "animString.h"
00011
00012 const int LOADING_BAR_OFFSET = 2;
00017 class AnimatableLcd: public LiquidCrystal_I2C {
00022     AnimString* animString;
00023 public:
00028     bool doAnimation = false;
00029     using LiquidCrystal_I2C::LiquidCrystal_I2C; //using the LiquidCrystal constructor
00030     void setAnimation(AnimString* _animString);
00031     void printCentered(String text, int length = -1, int row = 0);
00032     void printPretty(String text);
00033     void update();
00034     void init();
00035     using LiquidCrystal_I2C::print;
00036     void print(const String& text);
00037 };
00038 #endif
```

6.3 animLcd.ino-Dateireferenz

6.3.1 Ausführliche Beschreibung

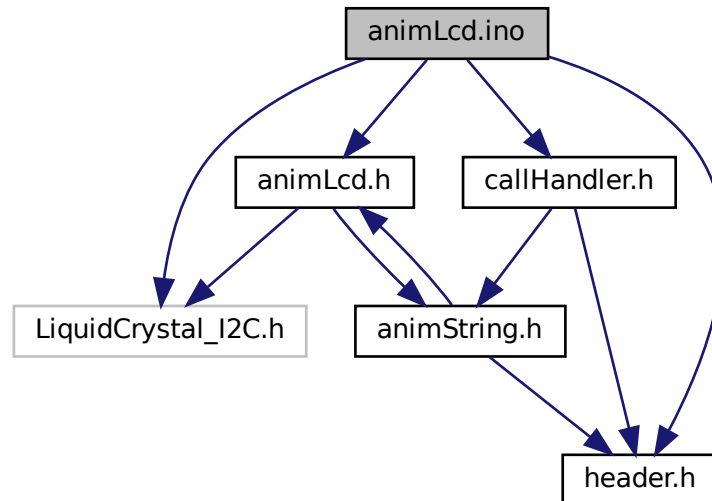
Implementation für die [AnimatableLcd](#) Klasse.

Definiert in Datei [animLcd.ino](#).

```
#include <LiquidCrystal_I2C.h>
#include "header.h"
#include "animLcd.h"
```

```
#include "callHandler.h"
```

Include-Abhängigkeitsdiagramm für animLcd.ino:



Variablen

- const byte `loading_empty_c` [8]
Werte für einen eigenen Character der ein leeres Viereck darstellt (für die [LcdLoadingAnim](#))
- const byte `loading_full_c` [8]
Werte für einen eigenen Character der ein volles Viereck darstellt (für die [LcdLoadingAnim](#))

6.3.2 Variablen-Dokumentation

6.3.2.1 `loading_empty_c` const byte loading_empty_c[8]

Initialisierung:

```
= {
  B11111,
  B10001,
  B10001,
  B10001,
  B10001,
  B10001,
  B10001,
  B10001,
  B11111
}
```

Werte für einen eigenen Character der ein leeres Viereck darstellt (für die [LcdLoadingAnim](#))

Definiert in Zeile 15 der Datei [animLcd.ino](#).

6.3.2.2 loading_full_c `const byte loading_full_c[8]`

Initialisierung:

```
= {
  B11111,
  B11111,
  B11111,
  B11111,
  B11111,
  B11111,
  B11111,
  B11111
}
```

Werte für einen eigenen Character der ein volles Viereck darstellt (für die [LcdLoadingAnim](#))

Definiert in Zeile 29 der Datei [animLcd.ino](#).

6.4 animLcd.ino

[gehe zur Dokumentation dieser Datei](#)

```
00001
00006 #include <LiquidCrystal_I2C.h>
00007 #include "header.h"
00008 #include "animLcd.h"
00009 #include "callHandler.h"
00010
00015 const byte loading_empty_c[8] = { //is used to define a custom character representing a square
00016   B11111,
00017   B10001,
00018   B10001,
00019   B10001,
00020   B10001,
00021   B10001,
00022   B10001,
00023   B11111
00024 };
00029 const byte loading_full_c[8] = { //is used to define a custom character representing a filled square
00030   B11111,
00031   B11111,
00032   B11111,
00033   B11111,
00034   B11111,
00035   B11111,
00036   B11111,
00037   B11111
00038 };
00043 void AnimatableLcd::init()
00044 {
00045   LiquidCrystal_I2C::init();
00046   backlight();
00047   noCursor();
00048   lcd.createChar(0, loading_empty_c);
00049   lcd.createChar(1, loading_full_c);
00050 }
00056 void AnimatableLcd::setAnimation(AnimString* _animString)
00057 {
00058   doAnimation = true;
00059   animString = _animString;
00060 }
00066 void AnimatableLcd::print(const String& text)
00067 {
00068   //custom print with ability to use custom characters, just insert the number of the custom
00069   character in the string (\ln for the nth character)
00070   //and it will be converted to the custom character (\ln so that \0 doesn't appear in the string,
00071   because it means end of string)
00070   for(char c:text){
00071     if(c>=8&&c<=15){//if it is a custom character
00072       write(c-8);
00073     }else if(c==1){//defining a non-newline space
00074       LiquidCrystal_I2C::print(" ");
00075     }
00076     else{
00077       LiquidCrystal_I2C::print(c);
00078     }
00079   }
00080 }
00088 void AnimatableLcd::printCentered(String text, int length = -1, int row = 0) //length<=16
00089 {
```

```

00090     if (length == -1) {
00091         length = text.length();
00092     }
00093     int offset = (16 - length) / 2; //rundet immer ab, da int
00094     setCursor(offset, row);
00095     print(text);
00096 }
00102 void AnimatableLcd::printPretty(String text) //handelt zeilenumbrüche und schreibt zentriert
00103 {
00104     clear();
00105     int length = text.length();
00106     if (length <= 16) {
00107         printCentered(text, length);
00108         return 0;
00109     }
00110     int spacePos = -1;
00111     for (int i = 15; i >= 0; i--) {
00112         if (text[i] == ' ') {
00113             spacePos = i;
00114             break;
00115         }
00116     }
00117     String row1, row2;
00118     if (spacePos != -1) {
00119         row1 = text.substring(0, spacePos);
00120         row2 = text.substring(spacePos + 1);
00121     } else {
00122         row1 = text.substring(0, 16);
00123         row2 = text.substring(16);
00124     }
00125     printCentered(row1, row1.length(), 0);
00126     printCentered(row2, row2.length(), 1);
00127 }
00132 void AnimatableLcd::update()
00133 {
00134     if (!doAnimation) {
00135         return;
00136     }
00137     animString->update();
00138 }

```

6.5 animString.h-Dateireferenz

6.5.1 Ausführliche Beschreibung

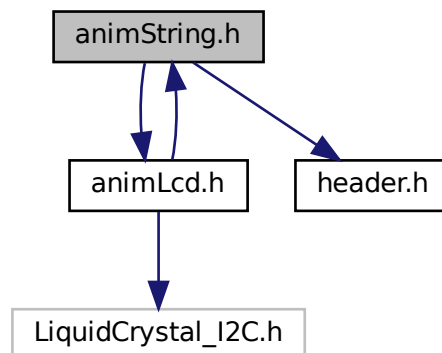
Header datei für eine Mehrzahl von animierbaren Strings und der [Callable](#) Klasse.

Definiert in Datei [animString.h](#).

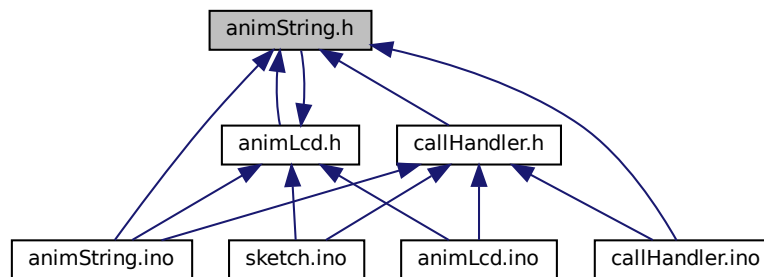
```
#include "animLcd.h"
```

```
#include "header.h"
```

Include-Abhängigkeitsdiagramm für animString.h:



Dieser Graph zeigt, welche Datei direkt oder indirekt diese Datei enthält:



Klassen

- class [Callable](#)
Ein Call der vom [CallHandler](#) aufgerufen werden kann.
- struct [FuncCall](#)
Ein Call der eine Funktion ausführt.
- struct [LcdString](#)
Ein String der auf dem [AnimatableLcd](#) angezeigt werden kann.
- class [AnimString](#)
Die Klasse für animierbare LcdStrings.
- class [LcdLoadingAnim](#)
Die Klasse der Lcd Ladeanimationen.
- class [LcdDotAnim](#)
Die Klasse der Lcd Punktanimationen.

6.6 animString.h

[gehe zur Dokumentation dieser Datei](#)

```

00001
00006 //the implementations for the animatable string class
00007 //animatable strings are strings that can be passed to the animatable lcd
00008 #ifndef ANIMSTRING_H
00009 #define ANIMSTRING_H
00010 struct Callable;
00011 struct FuncCallable;
00012 struct LcdString;
00013 class LcdLoadingAnim;
00014 class LcdDotAnim;
00015
00016 #include "animLcd.h"
00017 #include "header.h"
00018
00024 class Callable {
00025     virtual void run() {} //virtual->must be implemented by derived classes
00026     virtual bool isDone() {}
00027     virtual ~Callable() {} //let's derived classes free their own memory. ~functions are called when the
        object is deleted
00028 };
00033 struct FuncCall: public Callable {
00038     func_t<void> call;
00043     func_t<bool> _isDone;
00044     FuncCall(func_t<void> call, func_t<bool> _isDone):
00045         call(call), _isDone(_isDone) {}
00046     FuncCall(func_t<void> call): //when no isDone function is provided, isDone defaults to true
00047         call(call), _isDone([]() {return true;}) {}
  
```

```

00048     virtual ~FuncCall() {}
00049     void run();
00050     bool isDone();
00051 };
00052 struct LcdString: public Callable {
00053     String text;
00054     AnimatableLcd* lcd;
00055     time_t duration;
00056     time_t callStart; //time at which the string was written to the LCD
00057     LcdString(String text, AnimatableLcd* lcd, time_t duration = 0)
00058         : text(text), duration(duration), lcd(lcd) {}
00059     virtual ~LcdString() {}
00060     bool isDone();
00061     virtual void run();
00062     virtual void update() {}
00063 };
00064 class AnimString: public LcdString {
00065     protected:
00066         time_t stepDuration;
00067         time_t animStart;
00068         time_t lastRefresh;
00069     public:
00070         using LcdString::LcdString;
00071         virtual ~AnimString() {}
00072         virtual void init() {}
00073         void run();
00074 };
00075 class LcdLoadingAnim: public AnimString {
00076     public:
00077         using AnimString::AnimString;
00078         void init();
00079         void update();
00080 };
00081 class LcdDotAnim: public AnimString {
00082     public:
00083         LcdDotAnim(String text, AnimatableLcd* lcd, time_t duration = 0, time_t _stepDuration = 500)
00084             : AnimString(text, lcd, duration) {
00085             stepDuration = _stepDuration;
00086         }
00087         void init();
00088         void update();
00089 };
00090 #endif

```

6.7 animString.ino-Dateireferenz

6.7.1 Ausführliche Beschreibung

Implementationen der [Callable](#) und [LcdString](#) Klassen.

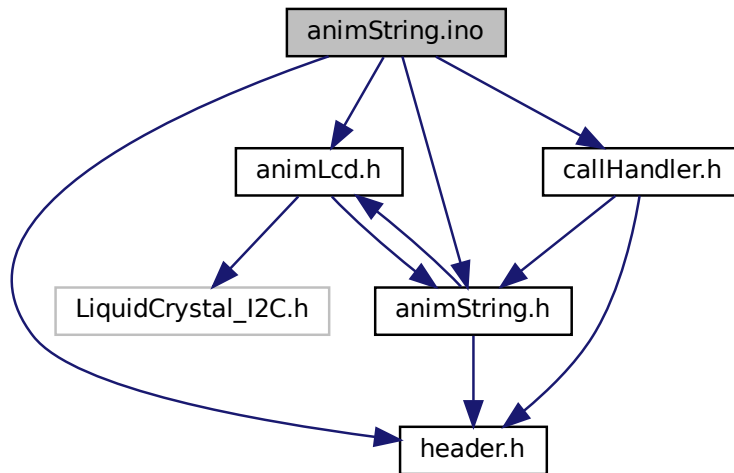
Definiert in Datei [animString.ino](#).

```

#include "animLcd.h"
#include "callHandler.h"
#include "header.h"
#include "animString.h"

```


Include-Abhängigkeitsdiagramm für animString.ino:



6.8 animString.ino

[gehe zur Dokumentation dieser Datei](#)

```

00001
00006 #include "animLcd.h"
00007 #include "callHandler.h"
00008 #include "header.h"
00009 #include "animString.h"
00014 void FuncCall::run()
00015 {
00016     call();
00017 }
00024 bool FuncCall::isDone()
00025 {
00026     return _isDone();
00027 }
00032 void LcdString::run()
00033 {
00034     callStart = millis();
00035     lcd->doAnimation = false;
00036     lcd->printPretty(this->text);
00037 }
00044 bool LcdString::isDone()
00045 {
00046     return millis() - callStart > duration;
00047 }
00052 void AnimString::run()
00053 {
00054     callStart = millis();
00055     lcd->clear();
00056     lcd->setAnimation(this);
00057     animStart = millis();
00058     lastRefresh = millis();
00059     init();
00060 }
00064 void LcdLoadingAnim::init()
00065 {
00066     stepDuration = duration / 9;
00067     if (text.length() > 16) {
00068         Serial.print("warning: text given for loading animation is too long, text: ");
00069         Serial.println(text);
00070     }
00071     lcd->printCentered(text);
00072     lcd->setCursor(LOADING_BAR_OFFSET, 1);
00073     for (int i = 0; i < 8; i++) {
00074         lcd->write(0);
  
```

```

00075     }
00076     lcd->print("0% ");
00077 }
00082 void LcdLoadingAnim::update()
00083 {
00084     time_t time = millis();
00085     short percent = (time - animStart) * 100 / duration;
00086     if (time - lastRefresh > stepDuration) {
00087         short nToFill = percent * 9 / 100;
00088         if (nToFill == 0) {
00089             return;
00090         }
00091         lcd->setCursor(nToFill + LOADING_BAR_OFFSET - 1, 1);
00092         lcd->write(1);
00093         lastRefresh = time;
00094     }
00095     lcd->setCursor(8 + LOADING_BAR_OFFSET, 1);
00096     lcd->print(percent);
00097     lcd->print("%");
00098 }
00103 void LcdDotAnim::init()
00104 {
00105     lcd->printPretty(text + "\1\1\1"); //spaces that can't be broken up to newlines
00106 }
00111 void LcdDotAnim::update()
00112 {
00113     time_t time = millis();
00114     if ((time - lastRefresh) < stepDuration) {
00115         return;
00116     }
00117     lastRefresh = time;
00118     int numDots = ((time - animStart) / stepDuration) % 4;
00119     char dots[4];
00120     for (int i = 0; i < 3; i++) {
00121         if (i < numDots) {
00122             dots[i] = '.';
00123         } else {
00124             dots[i] = '\1';
00125         }
00126     }
00127     dots[3] = '\0';
00128     lcd->printPretty(text + dots);
00129 }

```

6.9 callHandler.h-Dateireferenz

6.9.1 Ausführliche Beschreibung

header datei für den [CallHandler](#)

Definiert in Datei [callHandler.h](#).

```

#include "header.h"
#include "animString.h"

```

Include-Abhängigkeitsdiagramm für callHandler.h:



Dieser Graph zeigt, welche Datei direkt oder indirekt diese Datei enthält:



Klassen

- class `CallHandler`

Klasse, die Calls nacheinander aufruft.

6.10 callHandler.h

[gehe zur Dokumentation dieser Datei](#)

```

00001
00005 #ifndef CALLHANDLER_H
00006 #define CALLHANDLER_H
00007 class CallHandler;
00008
00009 #include "header.h"
00010 #include "animString.h"
00017 class CallHandler { //calls functions after a certain delay
  
```

```

00023     Callable** callPtrs;
00029     Callable** currCallPtr;
00034     Callable** lastCallPtr;
00039     time_t lastCallT;
00044     bool callsSet = false;
00045 public:
00050     bool running = false;
00051     void deleteCalls();
00052     void setCalls(Callable* newCallPtrs[], size_t nCalls);
00053     void update();
00054 };
00055 #endif

```

6.11 callHandler.ino-Dateireferenz

6.11.1 Ausführliche Beschreibung

Umsetzung der [CallHandler](#) Klasse.

Definiert in Datei [callHandler.ino](#).

```

#include "header.h"
#include "animString.h"
#include "callHandler.h"

```

Include-Abhängigkeitsdiagramm für callHandler.ino:



6.12 callHandler.ino

gehe zur Dokumentation dieser Datei

```

00001
00005 #include "header.h"
00006 #include "animString.h"
00007 #include "callHandler.h"
00012 void CallHandler::deleteCalls()
00013 {
00014     if (!callsSet) {
00015         return;
00016     }
00017     callsSet = false;
00018     for (Callable** callPtr = callPtrs; callPtr <= lastCallPtr; callPtr++) {
00019         delete *callPtr;
00020     }
00021     delete callPtrs;
00022 }
00030 void CallHandler::setCalls(Callable* newCallPtrs[], size_t nCalls)
00031 {
00032     /*if(callsSet){ //doing this would result in two sets of calls being in heap at once
00033         deleteCalls(); //solution is to delete previous calls before initializing a new one
00034     }*/
00035     callPtrs = newCallPtrs;
00036     callsSet = true;
00037     currCallPtr = callPtrs;
00038     (*currCallPtr)->run();
00039     lastCallPtr = callPtrs + nCalls - 1;
00040     lastCallT = millis();
00041     running = true;
00042 }
00047 void CallHandler::update()
00048 {
00049     if (!running) {
00050         return;
00051     }
00052     time_t timePassed = millis() - lastCallT;
00053     if ((*currCallPtr)->isDone()) { //->currCall->isDone()
00054         if (currCallPtr == lastCallPtr) {
00055             running = false;
00056             return;
00057         }
00058         currCallPtr++;
00059         (*currCallPtr)->run(); //->currCall->run();
00060     }
00061 }

```

6.13 customServo.h-Dateireferenz

6.13.1 Ausführliche Beschreibung

Header Datei der [CustomServo](#) Klasse.

Definiert in Datei [customServo.h](#).

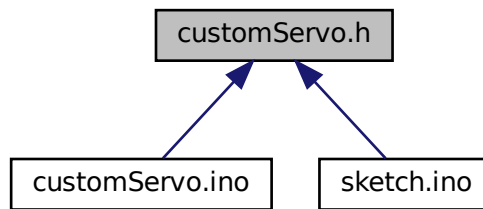
```
#include <Servo.h>
```

```
#include "header.h"
```

Include-Abhängigkeitsdiagramm für customServo.h:



Dieser Graph zeigt, welche Datei direkt oder indirekt diese Datei enthält:



Klassen

- class [CustomServo](#)

Eine Eigene Servo-Klasse, die es ermöglicht den Servo mit verschiedenen Geschwindigkeiten zu bewegen.

6.14 customServo.h

[gehe zur Dokumentation dieser Datei](#)

```

00001
00005 #ifndef CUSTOMSERVO_H
00006 #define CUSTOMSERVO_H
00007 #include <Servo.h>
00008 #include "header.h"
00013 class CustomServo: public Servo {
00018     short startAngle;
00023     short targetAngle;
00028     float speed;
00033     time_t startTime; //time at which servo started moving
00034     void startMove();
00035 public:
00036     bool done=true;
00037     void write(short newAngle);
00038     void write(short newAngle, time_t duration);
00039     void writeDirect(short angle);
00040     void setSpeed(float newSpeed);
00041     void updatePos();
00042     void stop();
00043     void start();
00044     bool isDone();
00045 };
00046 #endif
  
```

6.15 customServo.ino-Dateireferenz

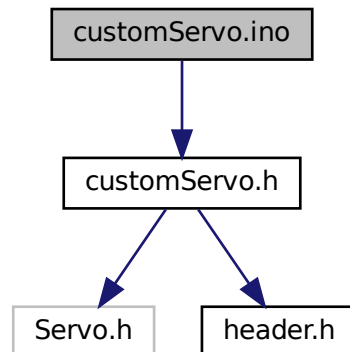
6.15.1 Ausführliche Beschreibung

Umsetzung der [CustomServo](#) Klasse.

Definiert in Datei [customServo.ino](#).

```
#include "customServo.h"
```

Include-Abhängigkeitsdiagramm für customServo.ino:



6.16 customServo.ino

[gehe zur Dokumentation dieser Datei](#)

```

00001
00006 #include "customServo.h"
00011 void CustomServo::startMove()
00012 {
00013     startAngle = read();
00014     startTime = millis();
00015     done = false;
00016 }
00022 void CustomServo::write(short newAngle)
00023 {
00024     startMove();
00025     targetAngle = newAngle;
00026 }
00033 void CustomServo::write(short newAngle, time_t duration)
00034 {
00035     write(newAngle);
00036     speed = (float)(targetAngle - startAngle) / (float)duration;
00037 }
00043 void CustomServo::writeDirect(short angle)
00044 {
00045     Servo::write(angle);
00046 }
00052 void CustomServo::setSpeed(float newSpeed)
00053 {
00054     startMove();
00055     speed = newSpeed;
00056 }
00061 void CustomServo::updatePos()
00062 {
00063     if (done) {
00064         return;
00065     }
00066     long timePassed = millis() - startTime;
00067     short newAngle;
00068     if (targetAngle > startAngle) {
00069         newAngle = startAngle + timePassed * speed;
00070         if (newAngle >= targetAngle) {
00071             Servo::write(targetAngle);
00072             done = true;
00073             return;
00074         }
00075     } else {
00076         newAngle = startAngle - timePassed * speed;
00077         if (newAngle <= targetAngle) {
00078             Servo::write(targetAngle);
00079             done = true;

```

```

00080         return;
00081     }
00082 }
00083 Servo::write(newAngle);
00084 }
00091 bool CustomServo::isDone()
00092 {
00093     return read() == targetAngle;
00094 }
00099 void CustomServo::stop()
00100 {
00101     done = true;
00102 }
00107 void CustomServo::start()
00108 {
00109     done = false;
00110 }

```

6.17 header.h-Dateireferenz

6.17.1 Ausführliche Beschreibung

Definiert variablen-types die überall im Programm benutzt werden.

Definiert in Datei [header.h](#).

Dieser Graph zeigt, welche Datei direkt oder indirekt diese Datei enthält:



Typdefinitionen

- using [time_t](#) = unsigned long
Ein Zeit Typ.
- template<typename ReturnT = void>
using [func_t](#) = ReturnT(*)()
Ein Funktions Typ.

6.17.2 Dokumentation der benutzerdefinierten Typen

6.17.2.1 func_t template<typename ReturnT = void>
using [func_t](#) = ReturnT(*)()

Ein Funktions Typ.

Template-Parameter

<i>ReturnT</i>	der Rückgabewert der Funktion die referenziert wird
----------------	---

Definiert in Zeile 17 der Datei [header.h](#).

6.17.2.2 `time_t` `using time_t = unsigned long`

Ein Zeit Typ.

Definiert in Zeile 10 der Datei [header.h](#).

6.18 header.h

[gehe zur Dokumentation dieser Datei](#)

```
00001
00005 #ifndef HEADER_H
00006 #define HEADER_H
00010 using time_t = unsigned long;
00016 template <typename ReturnT = void>
00017 using func_t=ReturnT(*)(); //defining a function type
00018 #endif
```

6.19 index.md-Dateireferenz

6.20 sketch.ino-Dateireferenz

6.20.1 Ausführliche Beschreibung

Hauptdatei, wichtigste Funktionen sind [setup\(\)](#) und [loop\(\)](#)

Definiert in Datei [sketch.ino](#).

```
#include "customServo.h"
#include "animLcd.h"
#include "callHandler.h"
#include "header.h"
```

Include-Abhängigkeitsdiagramm für sketch.ino:



Funktionen

- Farbe `measureColor ()`
Misst mithilfe des Reflexoptokopplers die Farbe des Balls.
- void `setLedColor` (unsigned char r, unsigned char g, unsigned char b)
Setzt die Farbe der RGB-Led.
- void `stopButtonClicked ()`
wird ausgeführt wenn der Stop-Knopf gedrückt wird
- bool `servolsDone ()`
Hilfs Funktion, Methoden können nicht als Funktionsparameter benutzt werden.
- template<short angle>
void `legBallAb` (String name)
Bewegt einen Ball zum Loch, legt ihn Ab und geht zurück.
- void `setup ()`
Wird am Anfang des Programms aufgerufen.
- void `loop ()`
Wird immer wieder ausgeführt.

Klassen

- class `ButtonHandler`
Kleine Klasse die Knopfdrücke verarbeitet.

Makrodefinitionen

- #define `GEH_ZURUECK`
2 Befehle, gibt "gehe zurück" auf dem Bildschirm aus und geht zurück

Aufzählungen

- enum `Farbe` { `WHITE` , `BLACK` , `ORANGE` , `NOTHING` }
Farben, werden für die Messungen des Reflexoptokopplers benutzt.

Variablen

- const int `LOADING_DURATION` = 3000
Gibt an, wie lange die Ladeanimation beim "Hochfahren" dauert.
- const int `ANGLE_LEFT_HOLE` = 180
Winkel des linken Lochs in Grad.
- const int `ANGLE_RIGHT_HOLE` = 90
Winkel des rechten Lochs in Grad.
- const int `ANGLE_CENTER` = 130
Winkel der Ablagefläche für neue Bälle, die sortiert werden sollen.
- const int `ANGLE_MIN` = 45
Der kleinste sichere Winkel.
- const int `PIN_SERVO` = 6
Der Pin an dem der Servomotor angeschlossen ist.
- const int `PIN_STOPBUTTON` = 13
Der Pin an dem der Start/Stop Knopf angeschlossen ist.

- `const int PIN_RED = 11`
Der Pin um die Rotfärbung der RGB-Led zu steuern.
- `const int PIN_GREEN = 10`
Der Pin um die Grünfärbung der RGB-Led zu steuern.
- `const int PIN_BLUE = 9`
Der Pin um die Blaufärbung der RGB-Led zu steuern.
- `const float SERVO_SPEED_DEFAULT = 0.01f`
Die normale Geschwindigkeit des Servos.
- `const float SERVO_SPEED_FAST = 0.5f`
Die "schnelle" Geschwindigkeit des Servos.
- `AnimatableLcd lcd (0x27, 16, 2)`
Der animierbare Lcd.
- `CallHandler callHandler`
Die CallHandler Instanz.
- `CustomServo servo`
Der Servo, eine CustomServo Instanz.
- `int nWhite = 0`
Die Anzahl weißer Bälle, die schon sortiert wurden.
- `int nBlack = 0`
Die Anzahl schwarzer Bälle, die schon sortiert wurden.
- `int nOrange = 0`
Die Anzahl orangener Bälle, die schon sortiert/entfernt wurden.
- `bool doFlicker = false`
Sagt aus, ob das Display flackern und die Led blinken soll.
- `ButtonHandler stopButton`
der Stop Knopf

6.20.2 Makro-Dokumentation

6.20.2.1 GEH_ZURUECK `#define GEH_ZURUECK`

Wert:

```
new LcdDotAnim("Gehe zur\365ck", &lcd), \
new FuncCall([]() {\
    servo.setSpeed(SERVO_SPEED_FAST); \
    servo.write(ANGLE_CENTER); \
}, &servoIsDone)
```

2 Befehle, gibt "gehe zurück" auf dem Bildschirm aus und geht zurück

Definiert in Zeile 238 der Datei `sketch.ino`.

6.20.3 Dokumentation der Aufzählungstypen

6.20.3.1 Farbe `enum Farbe`

Farben, werden für die Messungen des Reflexoptokopplers benutzt.

Aufzählungswerte

WHITE	
BLACK	
ORANGE	
NOTHING	

Definiert in Zeile 156 der Datei `sketch.ino`.

```
00157 {
00158     WHITE,
00159     BLACK,
00160     ORANGE,
00161     NOTHING
00162 };
```

6.20.4 Dokumentation der Funktionen

6.20.4.1 `legBallAb()` `template<short angle>`

```
void legBallAb (
    String name )
```

Bewegt einen Ball zum Loch, legt ihn Ab und geht zurück.

Template-Parameter

<i>angle</i>	Der Winkel als Template, da Lambdas (Funktionen die als Parameter weitergegeben werden) keine Variablen von Außen beinhalten dürfen
--------------	---

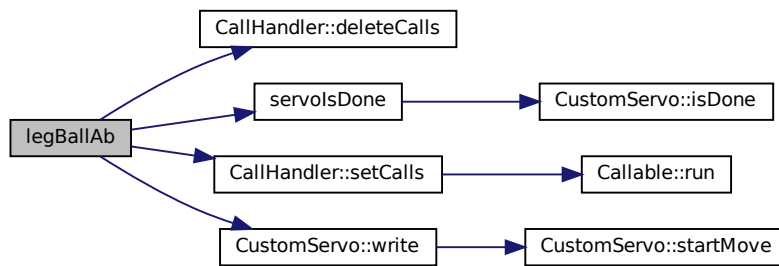
Parameter

<i>name</i>	Die Farbe des Balls
-------------	---------------------

Definiert in Zeile 251 der Datei `sketch.ino`.

```
00252 {
00253     callHandler.deleteCalls();
00254     /*static*/ auto calls = new Callable*[6] {
00255         //static so that the space for the calls is only allocated once
00256         //didn't end up being necessary because at there is only one object in heap at one point in time
00257         (callHandler.deleteCalls())
00258         // new so that it is allocated on the heap
00259         //auto automatically sets the type, in this case Callable*[] (Callable**)
00260         new LcdString("Ball erkannt, vorsicht", &lcd, 1000), //objects get upcasted to Callable*
00261         new LcdDotAnim(name + "er Ball, drehe Links", &lcd, 0),
00262         new FuncCall([]() {
00263             servo.write(angle);
00264             }, &servoIsDone),
00265         new LcdString("Angekommen", &lcd, 1000),
00266         GEH_ZURUECK //2 elemente
00267     };
00268     callHandler.setCalls(calls, 6); //if the number is too large the program crashes
00269 }
```

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



6.20.4.2 loop() void loop ()

Wird immer wieder ausgeführt.

Hier werden alle möglichen Objekte wie der Servo, der Lcd usw. aktualisiert und die Aktionen (Farbe des Balls messen, Bewegung des Servos Starten etc.) koordiniert

Definiert in Zeile 292 der Datei [sketch.ino](#).

```

00293 {
00294   callHandler.update();
00295   lcd.update();
00296   servo.updatePos();
00297   stopButton.update();
00298   if (doFlicker) {
00299     unsigned short b = (millis() / 2) % 513; //helligkeit: 0-512
00300     if (b > 255) {
00301       b = 511 - b; //wenn b größer als 255, wird die helligkeit kleiner, Werte über 255 werden also
"gespiegelt"
00302     }
00303     setLedColor(b, b / 2, 0); //orange
00304     if (random(3) == 0) {
00305       lcd.noBacklight();
00306     } else {
00307       lcd.backlight();
00308     }
00309   }
00310   if (callHandler.running) {
00311     return;
00312   }
00313   Farbe farbe = mesureColor();
00314   servo.setSpeed(SERVO_SPEED_DEFAULT);
00315   switch (farbe) {
00316     case WHITE:
00317     {
00318       nWhite++;
00319       Serial.println("white");
00320       legBallAb<ANGLE_LEFT_HOLE>("Wei\342");
00321       setLedColor(255, 255, 255);
00322       break;
00323     }
00324     case BLACK:
00325     {
00326       nBlack++;
00327       Serial.println("black");
00328       legBallAb<ANGLE_RIGHT_HOLE>("Schwarz");
00329       setLedColor(0, 0, 255);
00330       break;
00331     }
00332     case NOTHING:
00333     {
00334       Serial.println("nothing");
00335       callHandler.deleteCalls();
  
```

```

00336         /*static*/ auto callsNothing = new Callable*[1] {
00337             new LcdString(String("Ball einlegen W:") + nWhite + String(" S:") + nBlack + String(" O:") +
nOrange, &lcd, 1000)
00338         };
00339         callHandler.setCalls(callsNothing, 1);
00340         setLedColor(0, 255, 0);
00341         break;
00342     }
00343     case ORANGE:
00344     {
00345         doFlicker = true;
00346         nOrange++;
00347         Serial.println("orange");
00348         callHandler.deleteCalls();
00349         /*static*/ auto callsOrange = new Callable*[9] {
00350             new LcdDotAnim("\xC0ra\10g\xD9ner Ba\xED1", &lcd, 0),
//https://arduino.stackexchange.com/a/46833
00351             new FuncCall([]() {
00352                 servo.write(ANGLE_RIGHT_HOLE);
00353             }, &servoIsDone),
00354             new LcdString("Fehler \1rkannt", &lcd, 1000),
00355             new FuncCall([]() {
00356                 servo.setSpeed(SERVO_SPEED_FAST);
00357                 servo.write(ANGLE_MIN);
00358             }, &servoIsDone),
00359             new FuncCall([]() {
00360                 servo.setSpeed(SERVO_SPEED_DEFAULT);
00361             }, &servoIsDone),
00362             new FuncCall([]() {
00363                 doFlicker = false;
00364                 setLedColor(0, 255, 0);
00365                 lcd.backlight();
00366             },
00367             new LcdString("Fehler beseitigt", &lcd, 2000),
GEH_ZURUECK
00368         );
00369     };
00370     callHandler.setCalls(callsOrange, 9);
00371     break;
00372 }
00373 }
00374 }

```

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



6.20.4.3 `measureColor()` `Farbe` `measureColor ()`

Misst mithilfe des Reflexoptokopplers die Farbe des Balls.

Rückgabe

Farbe Die Farbe des Balls

Definiert in Zeile 169 der Datei `sketch.ino`.

```

00170 {
00171     //return WHITE;//inputs hardcoden, für Testzwecke
00172     int hue = random(0, 1000); //inputs simulieren
00173     //int hue=analogRead(A0);//tatsächlich Farbe messen
00174     if (hue <= 100) {
00175         return ORANGE;
00176     }
00177     if (hue < 500) {
00178         return WHITE;
00179     }
00180     if (hue < 830) {
00181         return NOTHING;
00182     }
00183     return BLACK;
00184 }
  
```

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



6.20.4.4 servolsDone() `bool servoIsDone ()`

Hilfs Funktion, Methoden können nicht als Funktionsparameter benutzt werden.

Rückgabe

true
false

Definiert in Zeile 230 der Datei `sketch.ino`.

```
00231 {  
00232     return servo.isDone();  
00233 }
```

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



6.20.4.5 setLedColor() void setLedColor (

```

    unsigned char r,
    unsigned char g,
    unsigned char b )

```

Setzt die Farbe der RGB-Led.

Parameter

<i>r</i>	Rot (0-255)
<i>g</i>	Grün (0-255)
<i>b</i>	Blau (0-255)

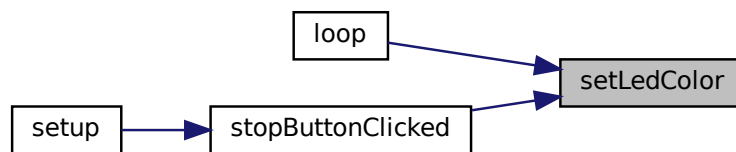
Definiert in Zeile 192 der Datei [sketch.ino](#).

```

00193 {
00194   analogWrite(PIN_RED, r);
00195   analogWrite(PIN_GREEN, g);
00196   analogWrite(PIN_BLUE, b);
00197 }

```

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



6.20.4.6 setup() void setup ()

Wird am Anfang des Programms aufgerufen.

Definiert in Zeile 273 der Datei [sketch.ino](#).

```

00274 {
00275   Serial.begin(9600);
00276   Serial.println("setup");
00277   servo.attach(PIN_SERVO);
00278   lcd.init();
00279   servo.writeDirect(ANGLE_CENTER);
00280   callHandler.setCalls(new Callable*[1] {
00281     new LcdLoadingAnim("Lade", &lcd, LOADING_DURATION),
00282   }, 1);
00283   randomSeed(analogRead(A1));
00284   stopButton = ButtonHandler(PIN_STOPBUTTON, &stopButtonClicked);
00285 }

```

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



6.20.4.7 stopButtonClicked() void stopButtonClicked ()

wird ausgeführt wenn der Stop-Knopf gedrückt wird

Definiert in Zeile 202 der Datei sketch.ino.

```

00203 {
00204   static bool isStopped = false; //wird nur einmal initialisiert
00205   isStopped = !isStopped;
00206   if (isStopped) {
00207     Serial.println("stopping servo");
00208     servo.stop();
00209     callHandler.deleteCalls();
00210     /*static*/ auto call = new Callable*[1] {
00211       new LcdDotAnim("gestoppt, warte auf start", &lcd, 1000000000000000) //ja, sollte ich vermutlich
    besser implementieren
00212     };
00213     callHandler.setCalls(call, 1);
00214     setLedColor(255, 0, 0);
00215     doFlicker = false;
00216   } else {
00217     callHandler.running = false;
00218   }
00219 }
  
```

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



6.20.5 Variablen-Dokumentation

6.20.5.1 **ANGLE_CENTER** `const int ANGLE_CENTER = 130`

Winkel der Ablagefläche für neue Bälle, die sortiert werden sollen.

Definiert in Zeile [29](#) der Datei [sketch.ino](#).

6.20.5.2 **ANGLE_LEFT_HOLE** `const int ANGLE_LEFT_HOLE = 180`

Winkel des linken Lochs in Grad.

Definiert in Zeile [19](#) der Datei [sketch.ino](#).

6.20.5.3 **ANGLE_MIN** `const int ANGLE_MIN = 45`

Der kleinste sichere Winkel.

Warnung

Wenn dieser Winkel nicht eingehalten wird schlägt der Arm gegen den Stopper

Definiert in Zeile [34](#) der Datei [sketch.ino](#).

6.20.5.4 **ANGLE_RIGHT_HOLE** `const int ANGLE_RIGHT_HOLE = 90`

Winkel des rechten Lochs in Grad.

Definiert in Zeile [24](#) der Datei [sketch.ino](#).

6.20.5.5 callHandler `CallHandler callHandler`

Die `CallHandler` Instanz.

Definiert in Zeile 80 der Datei `sketch.ino`.

6.20.5.6 doFlicker `bool doFlicker = false`

Sagt aus, ob das Display flackern und die Led blinken soll.

Wird auf true gesetzt, wenn ein Orangener Ball entdeckt wird

Definiert in Zeile 103 der Datei `sketch.ino`.

6.20.5.7 lcd `AnimatableLcd lcd(0x27, 16, 2) (`
`0x27 ,`
`16 ,`
`2)`

Der animierbare Lcd.

6.20.5.8 LOADING_DURATION `const int LOADING_DURATION = 3000`

Gibt an, wie lange die Ladeanimation beim "Hochfahren" dauert.

Definiert in Zeile 14 der Datei `sketch.ino`.

6.20.5.9 nBlack `int nBlack = 0`

Die Anzahl schwarzer Bälle, die schon sortiert wurden.

Definiert in Zeile 94 der Datei `sketch.ino`.

6.20.5.10 nOrange `int nOrange = 0`

Die Anzahl orangener Bälle, die schon sortiert/entfernt wurden.

Definiert in Zeile 98 der Datei `sketch.ino`.

6.20.5.11 nWhite `int nWhite = 0`

Die Anzahl weißer Bälle, die schon sortiert wurden.

Definiert in Zeile [90](#) der Datei [sketch.ino](#).

6.20.5.12 PIN_BLUE `const int PIN_BLUE = 9`

Der Pin um die Blaufärbung der RGB-Led zu steuern.

Definiert in Zeile [59](#) der Datei [sketch.ino](#).

6.20.5.13 PIN_GREEN `const int PIN_GREEN = 10`

Der Pin um die Grünfärbung der RGB-Led zu steuern.

Definiert in Zeile [54](#) der Datei [sketch.ino](#).

6.20.5.14 PIN_RED `const int PIN_RED = 11`

Der Pin um die Rotfärbung der RGB-Led zu steuern.

Definiert in Zeile [49](#) der Datei [sketch.ino](#).

6.20.5.15 PIN_SERVO `const int PIN_SERVO = 6`

Der Pin an dem der Servomotor angeschlossen ist.

Definiert in Zeile [39](#) der Datei [sketch.ino](#).

6.20.5.16 PIN_STOPBUTTON `const int PIN_STOPBUTTON = 13`

Der Pin an dem der Start/Stop Knopf angeschlossen ist.

Definiert in Zeile [44](#) der Datei [sketch.ino](#).

6.20.5.17 servo `CustomServo servo`

Der Servo, eine `CustomServo` Instanz.

Definiert in Zeile 85 der Datei `sketch.ino`.

6.20.5.18 SERVO_SPEED_DEFAULT `const float SERVO_SPEED_DEFAULT = 0.01f`

Die normale Geschwindigkeit des Servos.

Definiert in Zeile 64 der Datei `sketch.ino`.

6.20.5.19 SERVO_SPEED_FAST `const float SERVO_SPEED_FAST = 0.5f`

Die "schnelle" Geschwindigkeit des Servos.

Warnung

Servo bewegt sich hier nicht mit Maximalgeschwindigkeit. Wenn diese Geschwindigkeit schneller eingestellt wird als der Servo tatsächlich ist, hört er möglicherweise auf sich zu bewegen, bevor er an seinem Ziel angekommen ist Alternativ könnte `CustomServo::writeDirect` benutzt werden

Definiert in Zeile 70 der Datei `sketch.ino`.

6.20.5.20 stopButton `ButtonHandler stopButton`

der Stop Knopf

Definiert in Zeile 224 der Datei `sketch.ino`.

6.21 sketch.ino

gehe zur Dokumentation dieser Datei

```

00001
00006 #include "customServo.h"
00007 #include "animLcd.h"
00008 #include "callHandler.h"
00009 #include "header.h"
00014 const int LOADING_DURATION = 3000;
00019 const int ANGLE_LEFT_HOLE = 180;
00024 const int ANGLE_RIGHT_HOLE = 90;
00029 const int ANGLE_CENTER = 130;
00034 const int ANGLE_MIN = 45;
00039 const int PIN_SERVO = 6;
00044 const int PIN_STOPBUTTON = 13;
00049 const int PIN_RED = 11;
00054 const int PIN_GREEN = 10;
00059 const int PIN_BLUE = 9;
00064 const float SERVO_SPEED_DEFAULT = 0.01f;
00070 const float SERVO_SPEED_FAST = 0.5f;
00075 AnimatableLcd lcd(0x27, 16, 2);
00080 CallHandler callHandler;
00085 CustomServo servo;
00086
00090 int nWhite = 0;
00094 int nBlack = 0;
00098 int nOrange = 0;
00103 bool doFlicker = false;
00104
00109 class ButtonHandler { //handels button clicks
00114     int pin;
00119     bool isPressed = false;
00120 public:
00125     void (*onclick)();
00126     ButtonHandler() {}
00133     ButtonHandler(int pin, void (*onclick)()): pin(pin), onclick(onclick) {
00134         pinMode(pin, INPUT_PULLUP);
00135     }
00140     void update()
00141     {
00142         bool isPressedNew = digitalRead(pin) == HIGH;
00143         if (isPressedNew != isPressed) { //is not being pressed now, but was being pressed
00144             if (isPressed) {
00145                 Serial.println("click");
00146                 onclick();
00147             }
00148         }
00149         isPressed = isPressedNew;
00150     }
00151 };
00156 enum Farbe
00157 {
00158     WHITE,
00159     BLACK,
00160     ORANGE,
00161     NOTHING
00162 };
00163 Farbe measureColor(); //sonst erkennt Arduino Farbe nicht als typ an
//https://forum.arduino.cc/t/syntax-for-a-function-returning-an-enumerated-type/107241
00169 Farbe measureColor()
00170 {
00171     //return WHITE; //inputs hardcoden, für Testzwecke
00172     int hue = random(0, 1000); //inputs simulieren
00173     //int hue=analogRead(A0); //tatsächlich Farbe messen
00174     if (hue <= 100) {
00175         return ORANGE;
00176     }
00177     if (hue < 500) {
00178         return WHITE;
00179     }
00180     if (hue < 830) {
00181         return NOTHING;
00182     }
00183     return BLACK;
00184 }
00192 void setLedColor(unsigned char r, unsigned char g, unsigned char b) //unsigned char: 0-255
00193 {
00194     analogWrite(PIN_RED, r);
00195     analogWrite(PIN_GREEN, g);
00196     analogWrite(PIN_BLUE, b);
00197 }
00202 void stopButtonClicked()
00203 {
00204     static bool isStopped = false; //wird nur einmal initialisiert
00205     isStopped = !isStopped;
00206     if (isStopped) {

```

```

00207     Serial.println("stopping servo");
00208     servo.stop();
00209     callHandler.deleteCalls();
00210     /*static*/ auto call = new Callable*[1] {
00211         new LcdDotAnim("gestoppt, warte auf start", &lcd, 1000000000000000) //ja, sollte ich vermutlich
        besser implementieren
00212     };
00213     callHandler.setCalls(call, 1);
00214     setLedColor(255, 0, 0);
00215     doFlicker = false;
00216 } else {
00217     callHandler.running = false;
00218 }
00219 }
00224 ButtonHandler stopButton;
00230 bool servoIsDone()
00231 {
00232     return servo.isDone();
00233 }
00238 #define GEH_ZURUECK \
00239     new LcdDotAnim("Gehe zur\365ck",&lcd),\
00240     new FuncCall([]() {\
00241         servo.setSpeed(SERVO_SPEED_FAST);\
00242         servo.write(ANGLE_CENTER);\
00243     },&servoIsDone)
00250 template <short angle>
00251 void legBallAb(String name)
00252 {
00253     callHandler.deleteCalls();
00254     /*static*/ auto calls = new Callable*[6] {
00255         //static so that the space for the calls is only allocated once
        (https://cpp4arduino.com/2018/11/06/what-is-heap-fragmentation.html),
00256         //didn't end up being necessary because at there is only one object in heap at one point in time
        (callHandler.deleteCalls())
00257         // new so that it is allocated on the heap
        //auto automatically sets the type, in this case Callable*[] (Callable**)
00258         new LcdString("Ball erkannt, vorsicht", &lcd, 1000), //objects get upcasted to Callable*
00259         new LcdDotAnim(name + "er Ball, drehe Links", &lcd, 0),
00260         new FuncCall([]() {
00261             servo.write(angle);
00262         }, &servoIsDone),
00263         new LcdString("Angekommen", &lcd, 1000),
00264         GEH_ZURUECK //2 elemente
00265     };
00266     callHandler.setCalls(calls, 6); //if the number is too large the program crashes
00267 }
00273 void setup()
00274 {
00275     Serial.begin(9600);
00276     Serial.println("setup");
00277     servo.attach(PIN_SERVO);
00278     lcd.init();
00279     servo.writeDirect(ANGLE_CENTER);
00280     callHandler.setCalls(new Callable*[1] {
00281         new LcdLoadingAnim("Lade", &lcd, LOADING_DURATION),
00282     }, 1);
00283     randomSeed(analogRead(A1));
00284     stopButton = ButtonHandler(PIN_STOPBUTTON, &stopButtonClicked);
00285 }
00286
00287
00292 void loop()
00293 {
00294     callHandler.update();
00295     lcd.update();
00296     servo.updatePos();
00297     stopButton.update();
00298     if (doFlicker) {
00299         unsigned short b = (millis() / 2) % 513; //helligkeit: 0-512
00300         if (b > 255) {
00301             b = 511 - b; //wenn b größer als 255, wird die helligkeit kleiner, Werte über 255 werden also
            "gespiegelt"
00302         }
00303         setLedColor(b, b / 2, 0); //orange
00304         if (random(3) == 0) {
00305             lcd.noBacklight();
00306         } else {
00307             lcd.backlight();
00308         }
00309     }
00310     if (callHandler.running) {
00311         return;
00312     }
00313     Farbe farbe = measureColor();
00314     servo.setSpeed(SERVO_SPEED_DEFAULT);
00315     switch (farbe) {
00316         case WHITE:

```



```

00317     {
00318         nWhite++;
00319         Serial.println("white");
00320         legBallAb<ANGLE_LEFT_HOLE>("Wei\342");
00321         setLedColor(255, 255, 255);
00322         break;
00323     }
00324     case BLACK:
00325     {
00326         nBlack++;
00327         Serial.println("black");
00328         legBallAb<ANGLE_RIGHT_HOLE>("Schwarz");
00329         setLedColor(0, 0, 255);
00330         break;
00331     }
00332     case NOTHING:
00333     {
00334         Serial.println("nothing");
00335         callHandler.deleteCalls();
00336         /*static*/ auto callsNothing = new Callable*[1] {
00337             new LcdString(String("Ball einlegen W:") + nWhite + String(" S:") + nBlack + String(" O:") +
nOrange, &lcd, 1000)
00338         };
00339         callHandler.setCalls(callsNothing, 1);
00340         setLedColor(0, 255, 0);
00341         break;
00342     }
00343     case ORANGE:
00344     {
00345         doFlicker = true;
00346         nOrange++;
00347         Serial.println("orange");
00348         callHandler.deleteCalls();
00349         /*static*/ auto callsOrange = new Callable*[9] {
00350             new LcdDotAnim("\xC0ra\10g\xD9ner Ba\xED1", &lcd, 0),
//https://arduino.stackexchange.com/a/46833
00351             new FuncCall([]() {
00352                 servo.write(ANGLE_RIGHT_HOLE);
00353             }, &servoIsDone),
00354             new LcdString("Fehler \1rkannt", &lcd, 1000),
00355             new FuncCall([]() {
00356                 servo.setSpeed(SERVO_SPEED_FAST);
00357                 servo.write(ANGLE_MIN);
00358             }, &servoIsDone),
00359             new FuncCall([]() {
00360                 servo.setSpeed(SERVO_SPEED_DEFAULT);
00361             }, &servoIsDone),
00362             new FuncCall([]() {
00363                 doFlicker = false;
00364                 setLedColor(0, 255, 0);
00365                 lcd.backlight();
00366             },
00367             new LcdString("Fehler beseitigt", &lcd, 2000),
00368             GEH_ZURUECK
00369         );
00370         callHandler.setCalls(callsOrange, 9);
00371         break;
00372     }
00373 }
00374 }

```

Index

- _isDone
 - FuncCall, [31](#)
 - ~AnimString
 - AnimString, [11](#)
 - ~Callable
 - Callable, [15](#)
 - ~FuncCall
 - FuncCall, [30](#)
 - ~LcdString
 - LcdString, [37](#)
- ANGLE_CENTER
 - sketch.ino, [64](#)
- ANGLE_LEFT_HOLE
 - sketch.ino, [64](#)
- ANGLE_MIN
 - sketch.ino, [64](#)
- ANGLE_RIGHT_HOLE
 - sketch.ino, [64](#)
- AnimatableLcd, [3](#)
 - animString, [9](#)
 - doAnimation, [9](#)
 - init, [5](#)
 - print, [5](#)
 - printCentered, [6](#)
 - printPretty, [7](#)
 - setAnimation, [8](#)
 - update, [8](#)
- animLcd.h, [39](#)
 - LOADING_BAR_OFFSET, [40](#)
- animLcd.ino, [40](#)
 - loading_empty_c, [41](#)
 - loading_full_c, [41](#)
- animStart
 - AnimString, [12](#)
- AnimString, [10](#)
 - ~AnimString, [11](#)
 - animStart, [12](#)
 - init, [11](#)
 - lastRefresh, [12](#)
 - LcdString, [11](#)
 - run, [11](#)
 - stepDuration, [12](#)
- animString
 - AnimatableLcd, [9](#)
- animString.h, [43](#)
- animString.ino, [45](#)
- BLACK
 - sketch.ino, [57](#)
- ButtonHandler, [13](#)
 - ButtonHandler, [13](#)
 - isPressed, [14](#)
 - onclick, [14](#)
 - pin, [15](#)
 - update, [14](#)
- call
 - FuncCall, [31](#)
- Callable, [15](#)
 - ~Callable, [15](#)
 - isDone, [15](#)
 - run, [16](#)
- CallHandler, [16](#)
 - callPtrs, [20](#)
 - callsSet, [20](#)
 - currCallPtr, [20](#)
 - deleteCalls, [17](#)
 - lastCallPtr, [21](#)
 - lastCallT, [21](#)
 - running, [21](#)
 - setCalls, [18](#)
 - update, [19](#)
- callHandler
 - sketch.ino, [64](#)
- callHandler.h, [47](#)
- callHandler.ino, [49](#)
- callPtrs
 - CallHandler, [20](#)
- callsSet
 - CallHandler, [20](#)
- callStart
 - LcdString, [38](#)
- currCallPtr
 - CallHandler, [20](#)
- CustomServo, [21](#)
 - done, [27](#)
 - isDone, [22](#)
 - setSpeed, [23](#)
 - speed, [28](#)
 - start, [24](#)
 - startAngle, [28](#)
 - startMove, [24](#)
 - startTime, [28](#)
 - stop, [24](#)
 - targetAngle, [28](#)
 - updatePos, [25](#)
 - write, [25](#), [26](#)
 - writeDirect, [27](#)
- customServo.h, [50](#)
- customServo.ino, [51](#)
- deleteCalls
 - CallHandler, [17](#)
- doAnimation
 - AnimatableLcd, [9](#)
- doFlicker
 - sketch.ino, [65](#)
- done
 - CustomServo, [27](#)
- duration
 - LcdString, [38](#)

- Farbe
 - sketch.ino, 56
- func_t
 - header.h, 53
- FuncCall, 29
 - _isDone, 31
 - ~FuncCall, 30
 - call, 31
 - FuncCall, 29, 30
 - isDone, 30
 - run, 30
- GEH_ZURUECK
 - sketch.ino, 56
- header.h, 53
 - func_t, 53
 - time_t, 54
- index.md, 54
- init
 - AnimatableLcd, 5
 - AnimString, 11
 - LcdDotAnim, 32
 - LcdLoadingAnim, 34
- isDone
 - Callable, 15
 - CustomServo, 22
 - FuncCall, 30
 - LcdString, 37
- isPressed
 - ButtonHandler, 14
- lastCallPtr
 - CallHandler, 21
- lastCallT
 - CallHandler, 21
- lastRefresh
 - AnimString, 12
- lcd
 - LcdString, 38
 - sketch.ino, 65
- LcdDotAnim, 31
 - init, 32
 - LcdDotAnim, 32
 - update, 32
- LcdLoadingAnim, 33
 - init, 34
 - update, 35
- LcdString, 36
 - ~LcdString, 37
 - AnimString, 11
 - callStart, 38
 - duration, 38
 - isDone, 37
 - lcd, 38
 - LcdString, 37
 - run, 37
 - text, 38
 - update, 37
- legBallAb
 - sketch.ino, 57
- LOADING_BAR_OFFSET
 - animLcd.h, 40
- LOADING_DURATION
 - sketch.ino, 65
- loading_empty_c
 - animLcd.ino, 41
- loading_full_c
 - animLcd.ino, 41
- loop
 - sketch.ino, 58
- measureColor
 - sketch.ino, 60
- nBlack
 - sketch.ino, 65
- nOrange
 - sketch.ino, 65
- NOTHING
 - sketch.ino, 57
- nWhite
 - sketch.ino, 65
- onclick
 - ButtonHandler, 14
- ORANGE
 - sketch.ino, 57
- pin
 - ButtonHandler, 15
- PIN_BLUE
 - sketch.ino, 66
- PIN_GREEN
 - sketch.ino, 66
- PIN_RED
 - sketch.ino, 66
- PIN_SERVO
 - sketch.ino, 66
- PIN_STOPBUTTON
 - sketch.ino, 66
- print
 - AnimatableLcd, 5
- printCentered
 - AnimatableLcd, 6
- printPretty
 - AnimatableLcd, 7
- run
 - AnimString, 11
 - Callable, 16
 - FuncCall, 30
 - LcdString, 37
- running
 - CallHandler, 21
- servo
 - sketch.ino, 66

- SERVO_SPEED_DEFAULT
 - sketch.ino, [67](#)
- SERVO_SPEED_FAST
 - sketch.ino, [67](#)
- servosDone
 - sketch.ino, [61](#)
- setAnimation
 - AnimatableLcd, [8](#)
- setCalls
 - CallHandler, [18](#)
- setLedColor
 - sketch.ino, [61](#)
- setSpeed
 - CustomServo, [23](#)
- setup
 - sketch.ino, [62](#)
- sketch.ino, [54](#)
 - ANGLE_CENTER, [64](#)
 - ANGLE_LEFT_HOLE, [64](#)
 - ANGLE_MIN, [64](#)
 - ANGLE_RIGHT_HOLE, [64](#)
 - BLACK, [57](#)
 - callHandler, [64](#)
 - doFlicker, [65](#)
 - Farbe, [56](#)
 - GEH_ZURUECK, [56](#)
 - lcd, [65](#)
 - legBallAb, [57](#)
 - LOADING_DURATION, [65](#)
 - loop, [58](#)
 - measureColor, [60](#)
 - nBlack, [65](#)
 - nOrange, [65](#)
 - NOTHING, [57](#)
 - nWhite, [65](#)
 - ORANGE, [57](#)
 - PIN_BLUE, [66](#)
 - PIN_GREEN, [66](#)
 - PIN_RED, [66](#)
 - PIN_SERVO, [66](#)
 - PIN_STOPBUTTON, [66](#)
 - servo, [66](#)
 - SERVO_SPEED_DEFAULT, [67](#)
 - SERVO_SPEED_FAST, [67](#)
 - servosDone, [61](#)
 - setLedColor, [61](#)
 - setup, [62](#)
 - stopButton, [67](#)
 - stopButtonClicked, [63](#)
 - WHITE, [57](#)
- speed
 - CustomServo, [28](#)
- start
 - CustomServo, [24](#)
- startAngle
 - CustomServo, [28](#)
- startMove
 - CustomServo, [24](#)
- startTime
 - CustomServo, [28](#)
- stepDuration
 - AnimString, [12](#)
- stop
 - CustomServo, [24](#)
- stopButton
 - sketch.ino, [67](#)
- stopButtonClicked
 - sketch.ino, [63](#)
- targetAngle
 - CustomServo, [28](#)
- text
 - LcdString, [38](#)
- time_t
 - header.h, [54](#)
- update
 - AnimatableLcd, [8](#)
 - ButtonHandler, [14](#)
 - CallHandler, [19](#)
 - LcdDotAnim, [32](#)
 - LcdLoadingAnim, [35](#)
 - LcdString, [37](#)
- updatePos
 - CustomServo, [25](#)
- WHITE
 - sketch.ino, [57](#)
- write
 - CustomServo, [25](#), [26](#)
- writeDirect
 - CustomServo, [27](#)