

Sortiermaschine

Erzeugt von Doxygen 1.9.5

1 Sortiermaschine	1
1.1 Zur Dokumentation	1
2 Hierarchie-Verzeichnis	3
2.1 Klassenhierarchie	3
3 Klassen-Verzeichnis	5
3.1 Auflistung der Klassen	5
4 Datei-Verzeichnis	7
4.1 Auflistung der Dateien	7
5 Klassen-Dokumentation	9
5.1 AnimatableLcd Klassenreferenz	9
5.1.1 Ausführliche Beschreibung	10
5.1.2 Dokumentation der Elementfunktionen	10
5.1.2.1 init()	10
5.1.2.2 print()	11
5.1.2.3 printCentered()	11
5.1.2.4 printPretty()	12
5.1.2.5 setAnimation()	13
5.1.2.6 update()	14
5.1.3 Dokumentation der Datenelemente	15
5.1.3.1 animString	15
5.1.3.2 doAnimation	15
5.2 AnimString Klassenreferenz	15
5.2.1 Ausführliche Beschreibung	16
5.2.2 Beschreibung der Konstruktoren und Destruktoren	16
5.2.2.1 ~AnimString()	16
5.2.3 Dokumentation der Elementfunktionen	16
5.2.3.1 init()	17
5.2.3.2 LcdString()	17
5.2.3.3 run()	17
5.2.4 Dokumentation der Datenelemente	18
5.2.4.1 animStart	18
5.2.4.2 lastRefresh	18
5.2.4.3 stepDuration	18
5.3 ButtonHandler Klassenreferenz	19
5.3.1 Ausführliche Beschreibung	19
5.3.2 Beschreibung der Konstruktoren und Destruktoren	19
5.3.2.1 ButtonHandler() [1/2]	19
5.3.2.2 ButtonHandler() [2/2]	19
5.3.3 Dokumentation der Elementfunktionen	20
5.3.3.1 update()	20

5.3.4 Dokumentation der Datenelemente	20
5.3.4.1 isPressed	21
5.3.4.2 onclick	21
5.3.4.3 pin	21
5.4 Callable Klassenreferenz	21
5.4.1 Ausführliche Beschreibung	22
5.4.2 Beschreibung der Konstruktoren und Destruktoren	22
5.4.2.1 ~Callable()	22
5.4.3 Dokumentation der Elementfunktionen	22
5.4.3.1 isDone()	22
5.4.3.2 run()	22
5.5 CallHandler Klassenreferenz	23
5.5.1 Ausführliche Beschreibung	24
5.5.2 Dokumentation der Elementfunktionen	24
5.5.2.1 deleteCalls()	24
5.5.2.2 setCalls()	24
5.5.2.3 update()	25
5.5.3 Dokumentation der Datenelemente	26
5.5.3.1 callPtrs	26
5.5.3.2 callsSet	27
5.5.3.3 currCallPtr	27
5.5.3.4 lastCallPtr	27
5.5.3.5 lastCallT	27
5.5.3.6 running	28
5.6 CustomServo Klassenreferenz	28
5.6.1 Ausführliche Beschreibung	29
5.6.2 Dokumentation der Elementfunktionen	29
5.6.2.1 isDone()	29
5.6.2.2 setSpeed()	30
5.6.2.3 start()	31
5.6.2.4 startMove()	31
5.6.2.5 stop()	32
5.6.2.6 updatePos()	32
5.6.2.7 write() [1/2]	33
5.6.2.8 write() [2/2]	33
5.6.2.9 writeDirect()	34
5.6.3 Dokumentation der Datenelemente	34
5.6.3.1 done	35
5.6.3.2 speed	35
5.6.3.3 startAngle	35
5.6.3.4 startTime	35
5.6.3.5 targetAngle	35

5.7 FuncCall Strukturreferenz	36
5.7.1 Ausführliche Beschreibung	36
5.7.2 Beschreibung der Konstruktoren und Destruktoren	36
5.7.2.1 FuncCall() [1/2]	37
5.7.2.2 FuncCall() [2/2]	37
5.7.2.3 ~FuncCall()	37
5.7.3 Dokumentation der Elementfunktionen	37
5.7.3.1 isDone()	37
5.7.3.2 run()	38
5.7.4 Dokumentation der Datenelemente	38
5.7.4.1 _isDone	38
5.7.4.2 call	38
5.8 LcdDotAnim Klassenreferenz	39
5.8.1 Ausführliche Beschreibung	39
5.8.2 Beschreibung der Konstruktoren und Destruktoren	39
5.8.2.1 LcdDotAnim()	40
5.8.3 Dokumentation der Elementfunktionen	40
5.8.3.1 init()	40
5.8.3.2 update()	40
5.9 LcdLoadingAnim Klassenreferenz	41
5.9.1 Ausführliche Beschreibung	42
5.9.2 Dokumentation der Elementfunktionen	42
5.9.2.1 init()	42
5.9.2.2 update()	43
5.10 LcdString Strukturreferenz	43
5.10.1 Ausführliche Beschreibung	44
5.10.2 Beschreibung der Konstruktoren und Destruktoren	45
5.10.2.1 LcdString()	45
5.10.2.2 ~LcdString()	45
5.10.3 Dokumentation der Elementfunktionen	45
5.10.3.1 isDone()	45
5.10.3.2 run()	46
5.10.3.3 update()	46
5.10.4 Dokumentation der Datenelemente	46
5.10.4.1 callStart	47
5.10.4.2 duration	47
5.10.4.3 lcd	47
5.10.4.4 text	47
6 Datei-Dokumentation	49
6.1 animLcd.h-Dateireferenz	49
6.1.1 Ausführliche Beschreibung	50

6.1.2 Variablen-Dokumentation	50
6.1.2.1 LOADING_BAR_OFFSET	50
6.2 animLcd.h	51
6.3 animLcd.ino-Dateireferenz	51
6.3.1 Ausführliche Beschreibung	52
6.3.2 Variablen-Dokumentation	52
6.3.2.1 loading_empty_c	52
6.3.2.2 loading_full_c	52
6.4 animLcd.ino	53
6.5 animString.h-Dateireferenz	54
6.5.1 Ausführliche Beschreibung	55
6.6 animString.h	55
6.7 animString.ino-Dateireferenz	56
6.7.1 Ausführliche Beschreibung	56
6.8 animString.ino	57
6.9 callHandler.h-Dateireferenz	58
6.9.1 Ausführliche Beschreibung	59
6.10 callHandler.h	59
6.11 callHandler.ino-Dateireferenz	59
6.11.1 Ausführliche Beschreibung	60
6.12 callHandler.ino	60
6.13 customServo.h-Dateireferenz	61
6.13.1 Ausführliche Beschreibung	62
6.14 customServo.h	62
6.15 customServo.ino-Dateireferenz	62
6.15.1 Ausführliche Beschreibung	63
6.16 customServo.ino	63
6.17 header.h-Dateireferenz	64
6.17.1 Ausführliche Beschreibung	64
6.17.2 Dokumentation der benutzerdefinierten Typen	64
6.17.2.1 func_t	64
6.17.2.2 time_t	65
6.18 header.h	65
6.19 index.md-Dateireferenz	65
6.20 sketch.ino-Dateireferenz	65
6.20.1 Ausführliche Beschreibung	67
6.20.2 Makro-Dokumentation	67
6.20.2.1 GEH_ZURUECK	67
6.20.3 Dokumentation der Aufzählungstypen	68
6.20.3.1 Farbe	68
6.20.4 Dokumentation der Funktionen	68
6.20.4.1 legBallAb()	68

6.20.4.2 loop()	69
6.20.4.3 mesureColor()	71
6.20.4.4 servosDone()	72
6.20.4.5 setLedColor()	73
6.20.4.6 setup()	73
6.20.4.7 stopButtonClicked()	74
6.20.5 Variablen-Dokumentation	75
6.20.5.1 ANGLE_CENTER	75
6.20.5.2 ANGLE_LEFT_HOLE	75
6.20.5.3 ANGLE_MIN	75
6.20.5.4 ANGLE_RIGHT_HOLE	76
6.20.5.5 callHandler	76
6.20.5.6 doFlicker	76
6.20.5.7 lcd	76
6.20.5.8 LOADING_DURATION	76
6.20.5.9 nBlack	77
6.20.5.10 nOrange	77
6.20.5.11 nWhite	77
6.20.5.12 PIN_BLUE	77
6.20.5.13 PIN_GREEN	77
6.20.5.14 PIN_RED	78
6.20.5.15 PIN_SERVO	78
6.20.5.16 PIN_STOPBUTTON	78
6.20.5.17 servo	78
6.20.5.18 SERVO_SPEED_DEFAULT	78
6.20.5.19 SERVO_SPEED_FAST	79
6.20.5.20 stopButton	79
6.21 sketch.ino	79
Index	83

Kapitel 1

Sortiermaschine

1.1 Zur Dokumentation

Diese Seite hat eine Suchfunktion. Dort kann nach Klassen/Variablen/Funktionen etc. gesucht werden

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. 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

Kapitel 2

Hierarchie-Verzeichnis

2.1 Klassenhierarchie

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

ButtonHandler	19
Callable	21
FuncCall	36
LcdString	43
AnimString	15
LcdDotAnim	39
LcdLoadingAnim	41
CallHandler	23
LiquidCrystal_I2C	
AnimatableLcd	9
Servo	
CustomServo	28

Kapitel 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	9
AnimString	Die Klasse für animierbare LcdStrings	15
ButtonHandler	Kleine Klasse die Knopfdrücke verarbeitet	19
Callable	Ein Call der vom CallHandler aufgerufen werden kann	21
CallHandler	Klasse, die Calls nacheinander aufruft	23
CustomServo	Eine Eigene Servo-Klasse, die es ermöglicht den Servo mit verschiedenen Geschwindigkeiten zu bewegen	28
FuncCall	Ein Call der eine Funktion ausführt	36
LcdDotAnim	Die Klasse der Lcd Punktanimationen	39
LcdLoadingAnim	Die Klasse der Lcd Ladeanimationen	41
LcdString	Ein String der auf dem AnimatableLcd angezeigt werden kann	43

Kapitel 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)	49
animLcd.ino	Implementation für die AnimatableLcd Klasse	51
animString.h	Header datei für eine Mehrzahl von animierbaren Strings und der Callable Klasse	54
animString.ino	Implementationen der Callable und LcdString Klassen	56
callHandler.h	Header datei für den CallHandler	58
callHandler.ino	Umsetzung der CallHandler Klasse	59
customServo.h	Header Datei der CustomServo Klasse	61
customServo.ino	Umsetzung der CustomServo Klasse	62
header.h	Definiert variablen-types die überall im Programm benutzt werden	64
sketch.ino	Hauptdatei, wichtigste Funktionen sind setup() und loop()	65

Kapitel 5

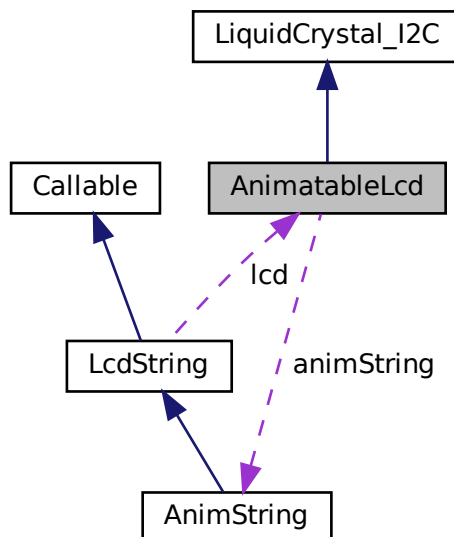
Klassen-Dokumentation

5.1 AnimatableLcd Klassenreferenz

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

```
#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.

5.1.1 Ausführliche Beschreibung

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

Definiert in Zeile 17 der Datei `animLcd.h`.

5.1.2 Dokumentation der Elementfunktionen

5.1.2.1 init()

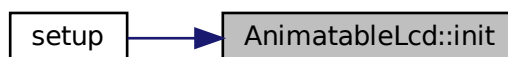
```
void AnimatableLcd::init ( )
```

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

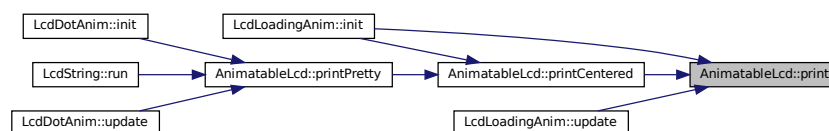
Parameter

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

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 (\ln for the nth character)
00069     //and it will be converted to the custom character (\ln 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.

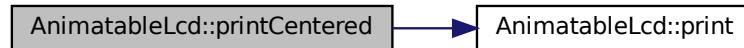
Parameter

<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

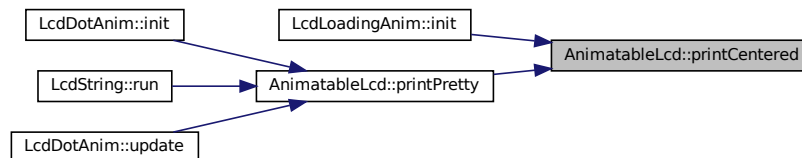
Definiert in Zeile 88 der Datei [animLcd.ino](#).

```
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](#).

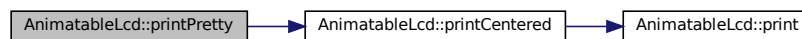
```
00103 {
00104     clear();
00105     int length = text.length();
00106     if (length <= 16) {
00107         printCentered(text, length);
00108         return 0;
00109     }
00110     int spacePos = -1;
00111     for (int i = 15; i >= 0; i--) {
00112         if (text[i] == ' ') {
00113             spacePos = i;
```

```

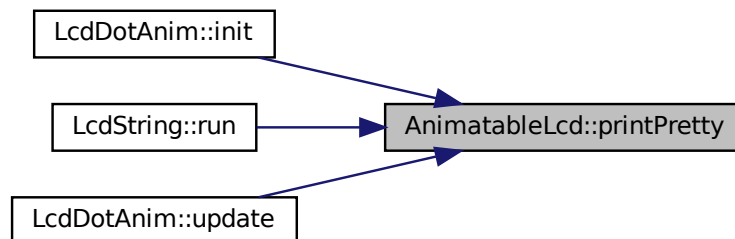
00114         break;
00115     }
00116 }
00117 String row1, row2;
00118 if (spacePos != -1) {
00119     row1 = text.substring(0, spacePos);
00120     row2 = text.substring(spacePos + 1);
00121 } else {
00122     row1 = text.substring(0, 16);
00123     row2 = text.substring(16);
00124 }
00125 printCentered(row1, 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](#).

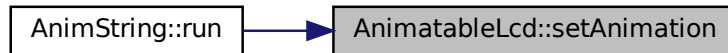
```

00057 {
00058     doAnimation = true;
00059     animString = _animString;

```

```
00060 }
```

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



5.1.2.6 update()

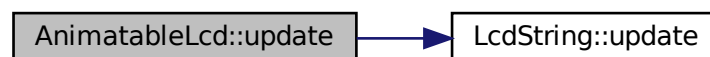
```
void AnimatableLcd::update ( )
```

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:

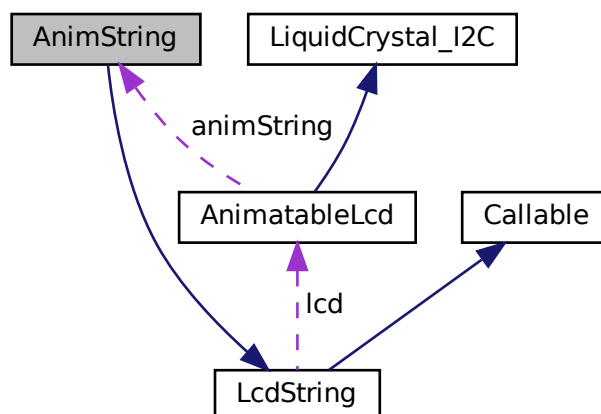
- [animLcd.h](#)
- [animLcd.ino](#)

5.2 AnimString Klassenreferenz

Die Klasse für animierbare LcdStrings.

```
#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

5.2.1 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](#).

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]
```

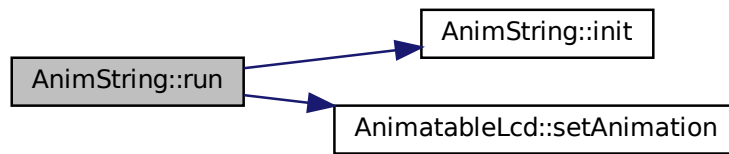
setzt Variablen die für alle Animationen notwendig sind und ruft dann ihre eigenen init Funktionen auf

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

Kleine Klasse die Knopfdrücke verarbeitet.

Öffentliche Methoden

- `ButtonHandler ()`
- `ButtonHandler (int pin, void(*onclick)())`
Erstellt ein neues `ButtonHandler` Objekt.
- `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.

5.3.1 Ausführliche Beschreibung

Kleine Klasse die Knopfdrücke verarbeitet.

Definiert in Zeile 109 der Datei `sketch.ino`.

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]
```

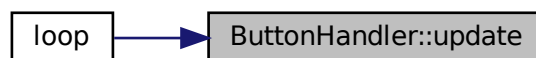
Prüft, ob der Knopf gedrückt/losgelassen wurde.

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

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

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

Private Methoden

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

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](#).

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()

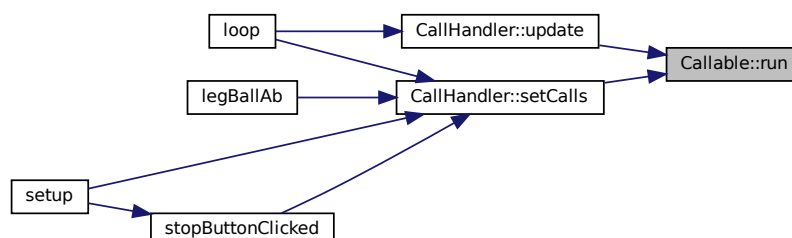
```
virtual void Callable::run ( ) [inline], [private], [virtual]
```

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:

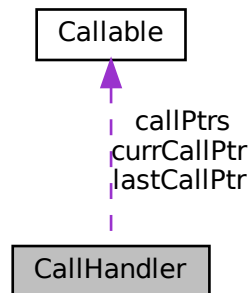
- [animString.h](#)

5.5 CallHandler Klassenreferenz

Klasse, die Calls nacheinander aufruft.

```
#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.

5.5.1 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`.

5.5.2 Dokumentation der Elementfunktionen

5.5.2.1 `deleteCalls()`

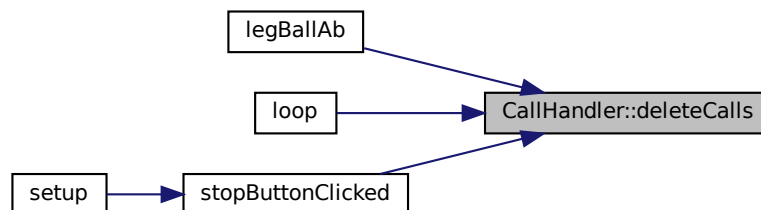
```
void CallHandler::deleteCalls ( )
```

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

<i>newCallPtrs</i>	
<i>nCalls</i>	

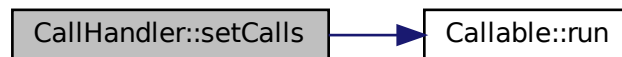
Definiert in Zeile 30 der Datei `callHandler.ino`.

```

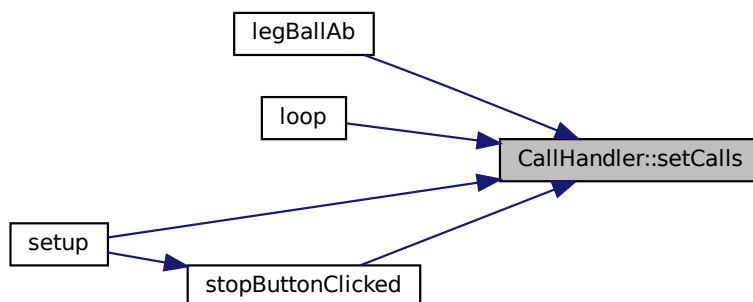
00031 {
00032     /*if(callsSet){ //doing this would result in two sets of calls being in heap at once
00033         deleteCalls(); //solution is to delete previous calls before initializing a new one
00034     }*/
00035     callPtrs = newCallPtrs;
00036     callsSet = true;
00037     currCallPtr = callPtrs;
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

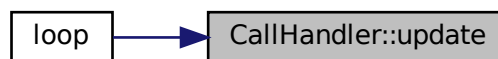
Definiert in Zeile 47 der Datei [callHandler.ino](#).

```
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:

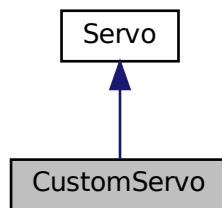
- [callHandler.h](#)
- [callHandler.ino](#)

5.6 CustomServo Klassenreferenz

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

```
#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)

5.6.1 Ausführliche Beschreibung

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

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

5.6.2 Dokumentation der Elementfunktionen

5.6.2.1 `isDone()`

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

Gibt an, ob der Servo angekommen ist.

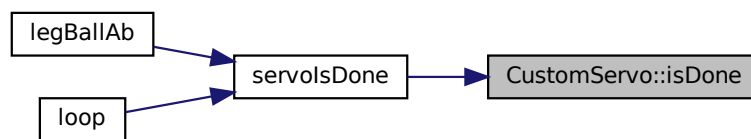
Rückgabe

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

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

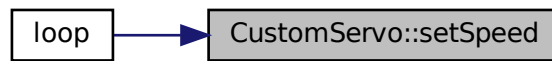
Definiert in Zeile 52 der Datei [customServo.ino](#).

```
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()

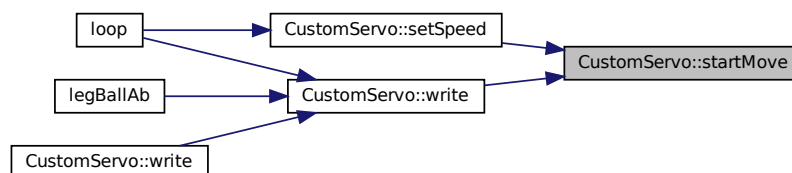
```
void CustomServo::startMove ( ) [private]
```

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()

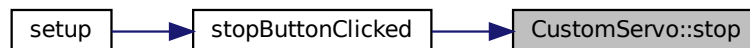
```
void CustomServo::stop ( )
```

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 ( )
```

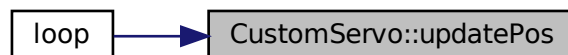
Aktualisiert die Position des Servomotors.

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

<i>newAngle</i>	
-----------------	--

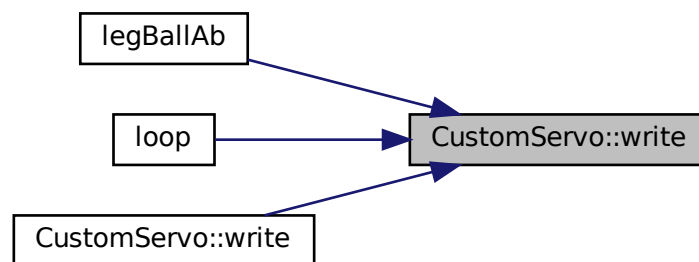
Definiert in Zeile 22 der Datei [customServo.ino](#).

```
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.

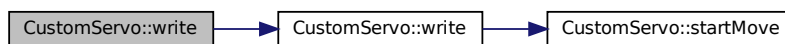
Parameter

<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:

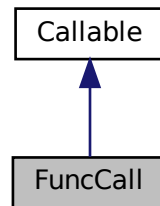
- [customServo.h](#)
- [customServo.ino](#)

5.7 FuncCall Strukturreferenz

Ein Call der eine Funktion ausführt.

```
#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

5.7.1 Ausführliche Beschreibung

Ein Call der eine Funktion ausführt.

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

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:

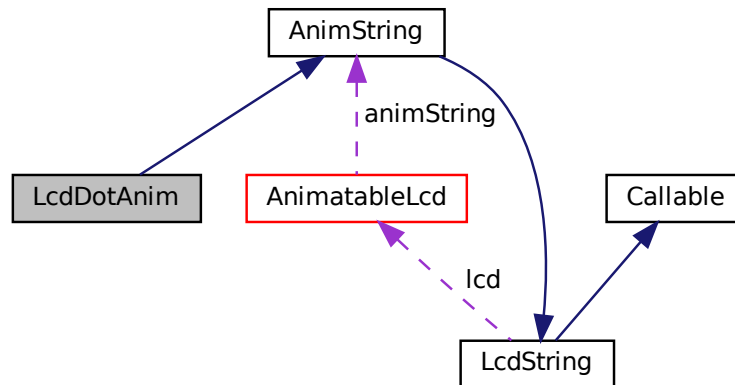
- [animString.h](#)
- [animString.ino](#)

5.8 LcdDotAnim Klassenreferenz

Die Klasse der Lcd Punktanimationen.

```
#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

5.8.1 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`.

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:

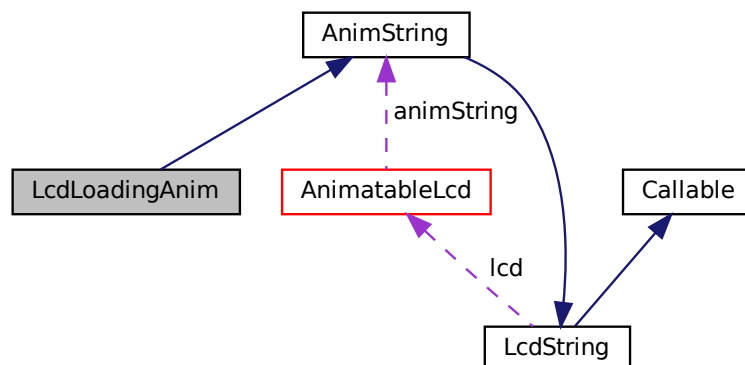
- [animString.h](#)
- [animString.ino](#)

5.9 LcdLoadingAnim Klassenreferenz

Die Klasse der Lcd Ladeanimationen.

```
#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

5.9.1 Ausführliche Beschreibung

Die Klasse der Lcd Ladeanimationen.

Zeigt acht Ladebalken und den Fortschritt in Prozent an

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

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 to 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:

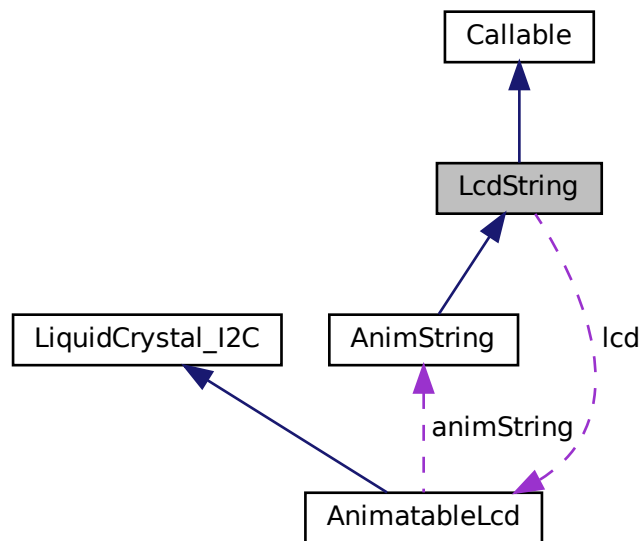
- [animString.h](#)
- [animString.ino](#)

5.10 LcdString Strukturreferenz

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

```
#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`

5.10.1 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`.

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()

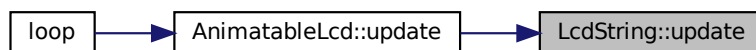
```
virtual void LcdString::update ( ) [inline], [virtual]
```

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](#)

Kapitel 6

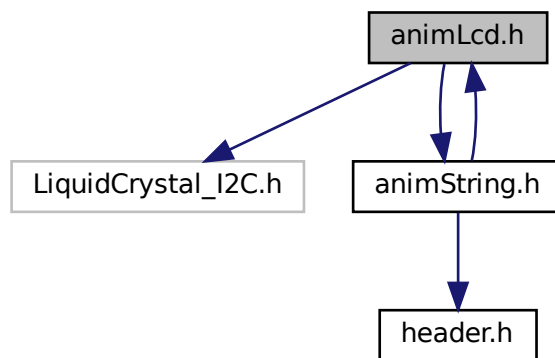
Datei-Dokumentation

6.1 animLcd.h-Dateireferenz

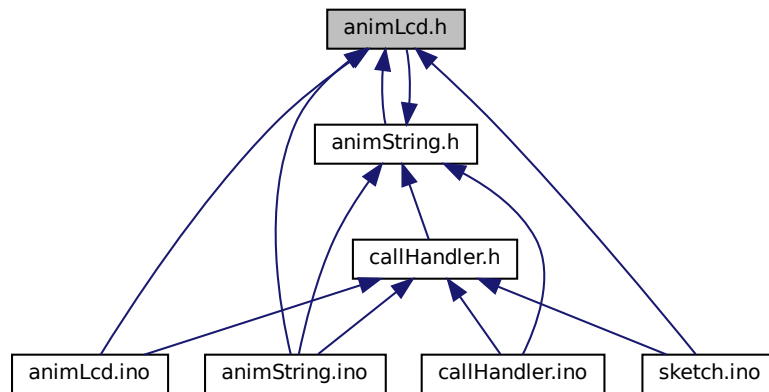
Header-Datei für den animierbaren lcd ([AnimatableLcd](#))

```
#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

6.1.1 Ausführliche Beschreibung

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

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

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

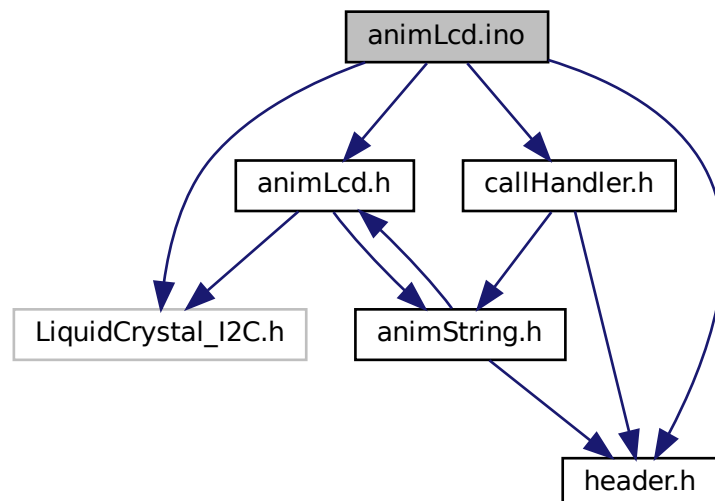
Implementation für die [AnimatableLcd](#) Klasse.

```

#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](#))

6.3.1 Ausführliche Beschreibung

Implementation für die [AnimatableLcd](#) Klasse.

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

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
    character in the string (\ln for the nth character)
00069     //and it will be converted to the custom character (\ln 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 }
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

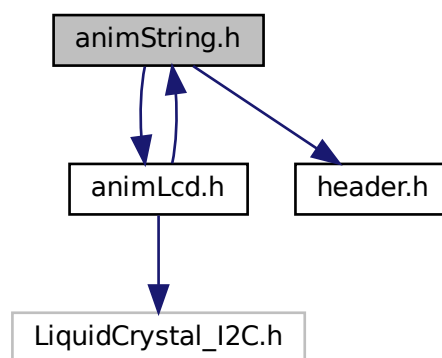
Header datei für eine Mehrzahl von animierbaren Strings und der [Callable](#) Klasse.

```

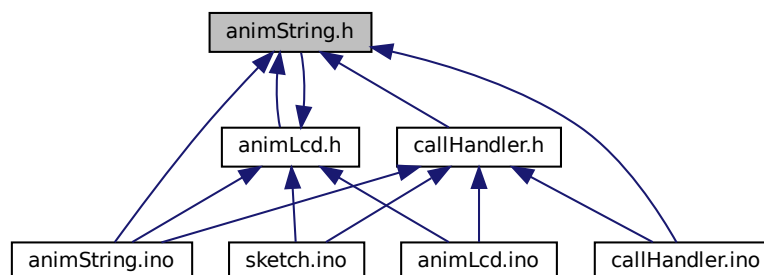
#include "animLcd.h"
#include "header.h"

```

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.

6.5.1 Ausführliche Beschreibung

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

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

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 };
00056 struct LcdString: public Callable {
00061     String text;
00062     AnimatableLcd* lcd;
00063     time_t duration;
00064     time_t callStart; //time at which the string was written to the LCD
00065     LcdString(String text, AnimatableLcd* lcd, time_t duration = 0)
00066         : text(text), duration(duration), lcd(lcd) {}
00067     virtual ~LcdString() {}
00068     bool isDone();
00069     virtual void run();
00070     virtual void update() {}

```

```

00071 };
00076 class AnimString: public LcdString {
00077     protected:
00082         time_t stepDuration;
00087         time_t animStart;
00092         time_t lastRefresh;
00093     public:
00094         using LcdString::LcdString;
00095         virtual ~AnimString() {}
00096         virtual void init() {}
00097         void run();
00098 };
00103 class LcdLoadingAnim: public AnimString {
00104     public:
00105         using AnimString::AnimString;
00106         void init();
00107         void update();
00108 };
00113 class LcdDotAnim: public AnimString {
00114     public:
00115         LcdDotAnim(String text, AnimatableLcd* lcd, time_t duration = 0, time_t _stepDuration = 500)
00116             : AnimString(text, lcd, duration) {
00117             stepDuration = _stepDuration;
00118         }
00119         void init();
00120         void update();
00121 };
00122 #endif

```

6.7 animString.ino-Dateireferenz

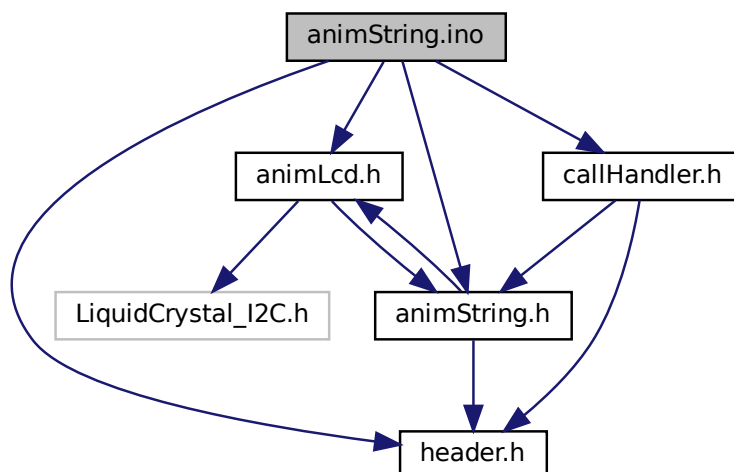
Implementationen der [Callable](#) und [LcdString](#) Klassen.

```

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

```

Include-Abhängigkeitsdiagramm für animString.ino:



6.7.1 Ausführliche Beschreibung

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

Definiert in Datei [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 to 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     }

```

```

00126     }
00127     dots[3] = '\0';
00128     lcd->printPretty(text + dots);
00129 }

```

6.9 callHandler.h-Dateireferenz

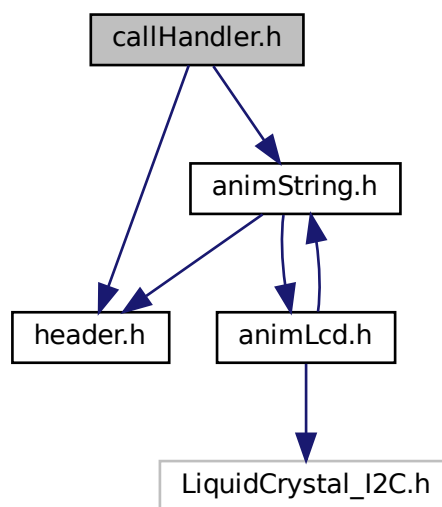
header datei für den [CallHandler](#)

```

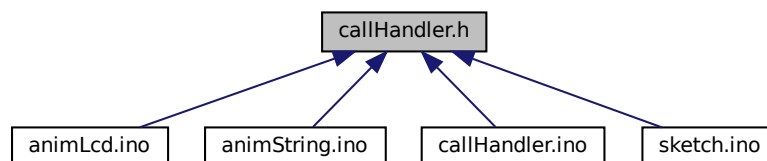
#include "header.h"
#include "animString.h"

```

Include-Abhängigkeitsdiagramm für callHandler.h:



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



Klassen

- class [CallHandler](#)

Klasse, die Calls nacheinander aufruft.

6.9.1 Ausführliche Beschreibung

header datei für den [CallHandler](#)

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

6.10 callHandler.h

[gehe zur Dokumentation dieser Datei](#)

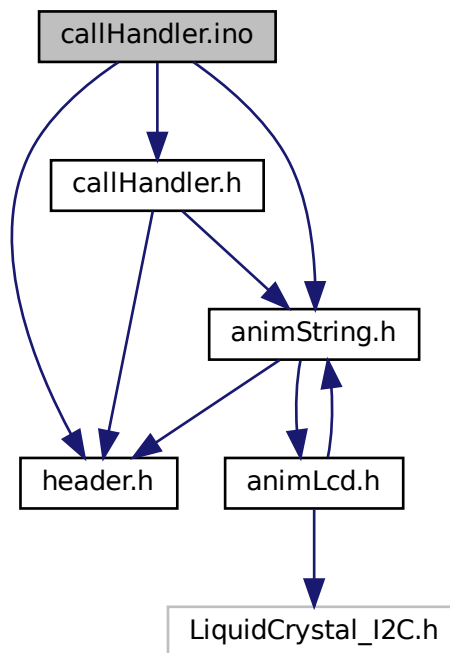
```
00001
00005 #ifndef CALLHANDLER_H
00006 #define CALLHANDLER_H
00007 class CallHandler;
00008
00009 #include "header.h"
00010 #include "animString.h"
00017 class CallHandler { //calls functions after a certain delay
00023     Callable** callPtrs;
00029     Callable** currCallPtr;
00034     Callable** 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

Umsetzung der [CallHandler](#) Klasse.

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

Include-Abhängigkeitsdiagramm für callHandler.ino:



6.11.1 Ausführliche Beschreibung

Umsetzung der [CallHandler](#) Klasse.

Definiert in Datei [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

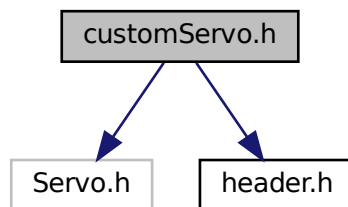
Header Datei der [CustomServo](#) Klasse.

```

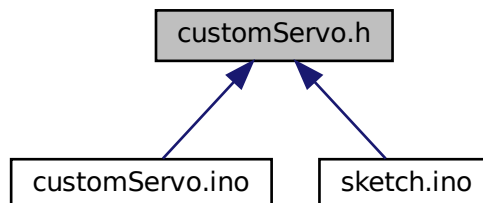
#include <Servo.h>
#include "header.h"

```

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.

6.13.1 Ausführliche Beschreibung

Header Datei der [CustomServo](#) Klasse.

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

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

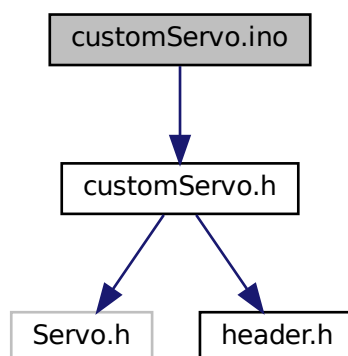
```

6.15 customServo.ino-Dateireferenz

Umsetzung der [CustomServo](#) Klasse.

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

Include-Abhängigkeitsdiagramm für customServo.ino:



6.15.1 Ausführliche Beschreibung

Umsetzung der [CustomServo](#) Klasse.

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

6.16 customServo.ino

[gehe zur Dokumentation dieser Datei](#)

```

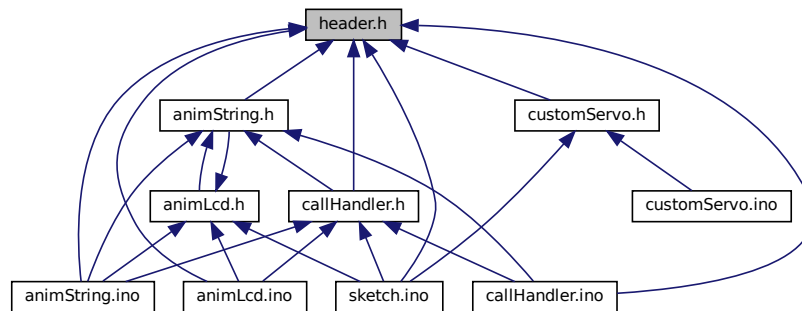
00001
00006 #include "customServo.h"
00011 void CustomServo::startMove()
00012 {
00013     startAngle = read();
00014     startTime = millis();
00015     done = false;
00016 }
00022 void CustomServo::write(short newAngle)
00023 {
00024     startMove();
00025     targetAngle = newAngle;
00026 }
00033 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

Definiert variablen-types die überall im Programm benutzt werden.

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.

6.17.1 Ausführliche Beschreibung

Definiert variablen-types die überall im Programm benutzt werden.

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

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
```

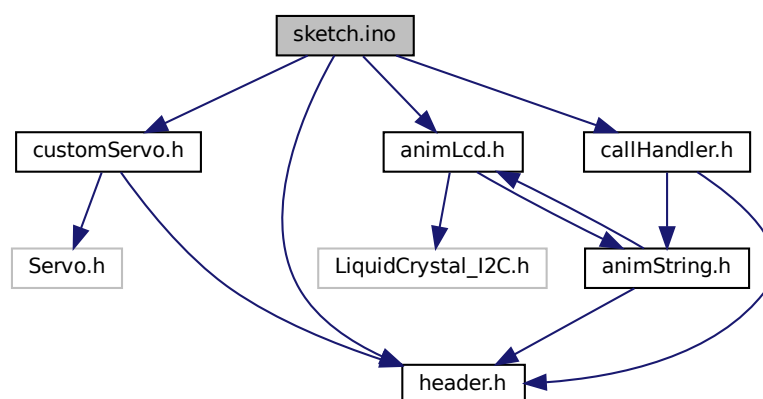
6.19 index.md-Dateireferenz

6.20 sketch.ino-Dateireferenz

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

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

Include-Abhängigkeitsdiagramm für sketch.ino:



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.

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.

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*

6.20.1 Ausführliche Beschreibung

Hauptdatei, wichtigste Funktionen sind `setup()` und `loop()`

Definiert in Datei `sketch.ino`.

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](#).

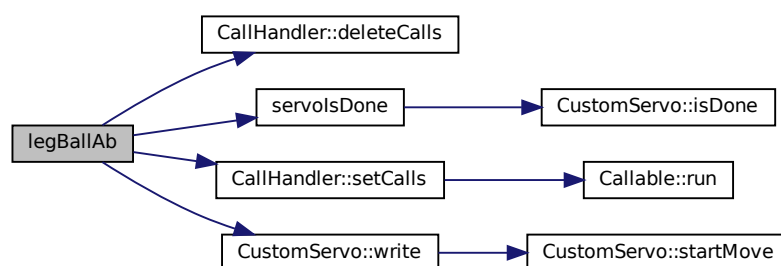
```
00252 {
00253     callHandler.deleteCalls();
00254     /*static*/ auto calls = new Callable*[6] {
```

```

00255 //static so that the space for the calls is only allocated once
00256 (https://cpp4arduino.com/2018/11/06/what-is-heap-fragmentation.html),
00257 //didn't end up being necessary because at there is only one object in heap at one point in time
00258 (callHandler.deleteCalls())
00259 // new so that it is allocated on the heap
00260 //auto automatically sets the type, in this case Callable*[] (Callable**)
00261 new LcdString("Ball erkannt, vorsicht", &lcd, 1000), //objects get upcasted to Callable*
00262 new LcdDotAnim(name + "er Ball, drehe Links", &lcd, 0),
00263 new FuncCall([]() {
00264     servo.write(angle);
00265 }, &servoIsDone),
00266 new LcdString("Angekommen", &lcd, 1000),
00267 GEH_ZURUECK //2 elemente
00268 };
00269 callHandler.setCalls(calls, 6); //if the number is too large the program crashes
00270 }

```

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
00302             "gespiegelt"
00303         }
00304         setLedColor(b, b / 2, 0); //orange
00305         if (random(3) == 0) {
00306             lcd.noBacklight();
00307         } else {
00308             lcd.backlight();
00309         }
00310     }
00311     if (callHandler.running) {
00312         return;
00313     }
00314     Farbe farbe = measureColor();
00315     servo.setSpeed(SERVO_SPEED_DEFAULT);

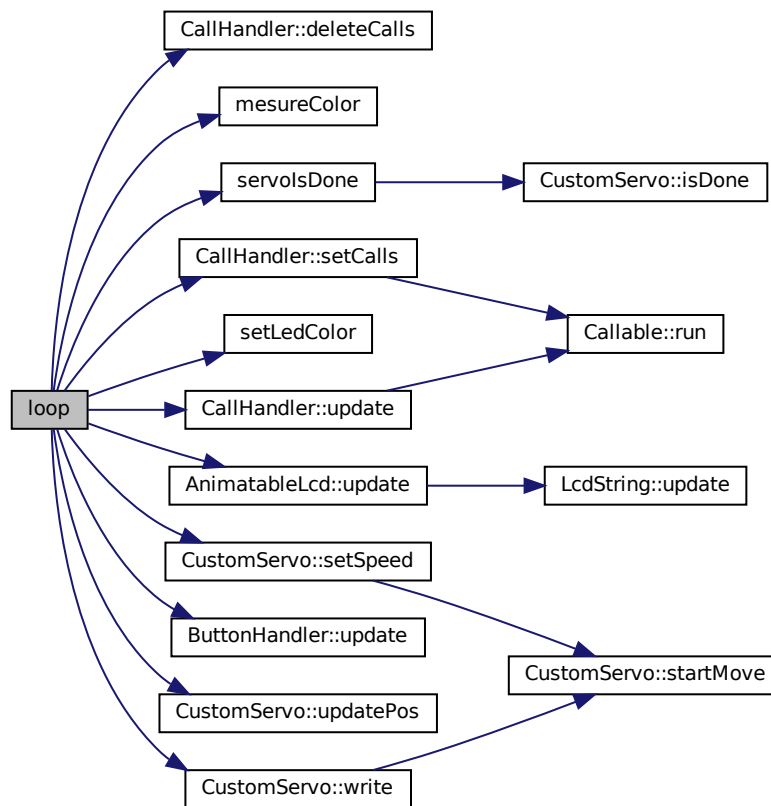
```

```

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             }, &servoIsDone),
00367             new LcdString("Fehler beseitigt", &lcd, 2000),
00368             GEH_ZURUECK
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 mesureColor()

```
Farbe mesureColor ( )
```

Misst mithilfe des Reflexoptokopplers die Farbe des Balls.

Rückgabe

Farbe Die Farbe des Balls

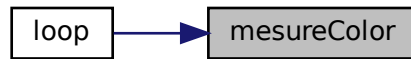
Definiert in Zeile 169 der Datei [sketch.ino](#).

```

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

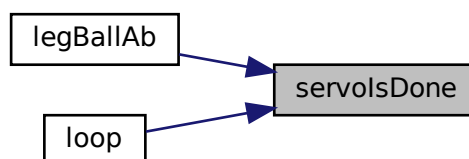
Definiert in Zeile 230 der Datei [sketch.ino](#).

```
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.

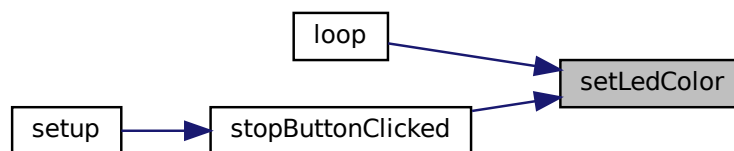
Parameter

<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](#).

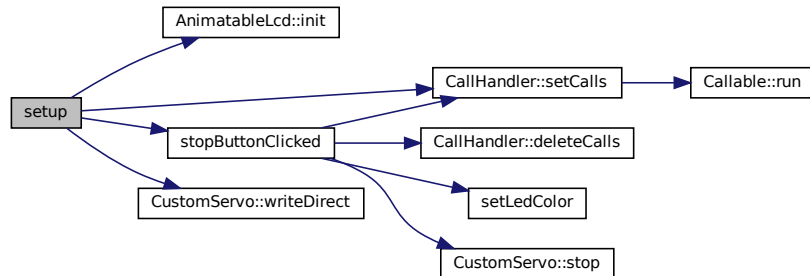
```
00274 {
00275     Serial.begin(9600);
00276     Serial.println("setup");
00277     servo.attach(PIN_SERVO);
00278     lcd.init();
00279     servo.writeDirect(ANGLE_CENTER);
00280     callHandler.setCalls(new Callable*[1] {
00281         new LcdLoadingAnim("Lade", &lcd, LOADING_DURATION),
00282     }, 1);
00283 }
```

```

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

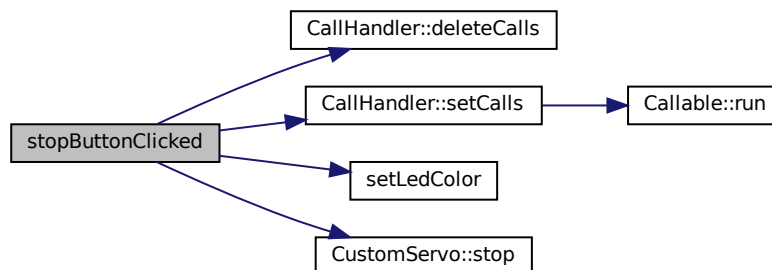
Definiert in Zeile 202 der Datei [sketch.ino](#).

```

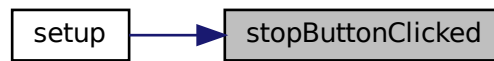
00203 {
00204     static bool isStopped = false; //wird nur einmal initialisiert
00205     isStopped = !isStopped;
00206     if (isStopped) {
00207         Serial.println("stopping servo");
00208         servo.stop();
00209         callHandler.deleteCalls();
00210         /*static*/ auto call = new Callable*[1] {
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 };
00152 enum Farbe
00153 {
00154     WHITE,
00155     BLACK,
00156     ORANGE,
00157     NOTHING
00158 };
00159 Farbe measureColor(); //sonst erkennt Arduino Farbe nicht als typ an
//https://forum.arduino.cc/t/syntax-for-a-function-returning-an-enumerated-type/107241
00160 Farbe measureColor()
00161 {
00162     //return WHITE;//inputs hardcoden, für Testzwecke
00163     int hue = random(0, 1000); //inputs simulieren
00164     //int hue=analogRead(A0);//tatsächlich Farbe messen
00165     if (hue <= 100) {
00166         return ORANGE;
00167     }
00168     if (hue < 500) {
00169         return WHITE;
00170     }
00171     if (hue < 830) {
00172         return NOTHING;
00173     }
00174     return BLACK;
00175 }
00176 void setLedColor(unsigned char r, unsigned char g, unsigned char b) //unsigned char: 0-255
00177 {
00178     analogWrite(PIN_RED, r);
00179     analogWrite(PIN_GREEN, g);
00180     analogWrite(PIN_BLUE, b);
00181 }
00182 void stopButtonClicked()
00183 {
00184     static bool isStopped = false; //wird nur einmal initialisiert
00185     isStopped = !isStopped;
00186     if (isStopped) {
00187         Serial.println("stopping servo");
00188         servo.stop();
00189         callHandler.deleteCalls();
00190         /*static*/ auto call = new Callable*[1] {
00191             new LcdDotAnim("gestoppt, warte auf start", &lcd, 1000000000000000) //ja, sollte ich vermutlich
00192             besser implementieren
00193         };
00194         callHandler.setCalls(call, 1);
00195         setLedColor(255, 0, 0);
00196         doFlicker = false;
00197     } else {
00198         callHandler.running = false;
00199     }
00200 }
00201 ButtonHandler stopButton;
00202 bool servoIsDone()
00203 {
00204     return servo.isDone();
00205 }
00206 #define GEH_ZURUECK \
00207     new LcdDotAnim("Gehe zur\365ck", &lcd), \
00208     new FuncCall([]() {\
00209         servo.setSpeed(SERVO_SPEED_FAST); \
00210         servo.write(ANGLE_CENTER); \
00211     }, &servoIsDone)
00212 template <short angle>
00213 void legBallAb(String name)
00214 {
00215     callHandler.deleteCalls();
00216     /*static*/ auto calls = new Callable*[6] {
00217         //static so that the space for the calls is only allocated once
00218         //https://cpp4arduino.com/2018/11/06/what-is-heap-fragmentation.html),
00219         //didn't end up being necessary because at there is only one object in heap at one point in time
00220         (callHandler.deleteCalls())
00221         // new so that it is allocated on the heap
00222         //auto automatically sets the type, in this case Callable*[] (Callable**)
00223         new LcdString("Ball erkannt, vorsicht", &lcd, 1000), //objects get upcasted to Callable*
00224         new LcdDotAnim(name + "er Ball, drehe Links", &lcd, 0),
00225         new FuncCall([]() {
00226             servo.write(angle);
00227         }, &servoIsDone),
00228         new LcdString("Angekommen", &lcd, 1000),
00229         GEH_ZURUECK //2 elemente
00230     };

```



```

00266     };
00267     callHandler.setCalls(calls, 6); //if the number is too large the program crashes
00268 }
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
00302             "gespiegelt"
00303         }
00304         setLedColor(b, b / 2, 0); //orange
00305         if (random(3) == 0) {
00306             lcd.noBacklight();
00307         } else {
00308             lcd.backlight();
00309         }
00310     }
00311     if (callHandler.running) {
00312         return;
00313     }
00314     Farbe farbe = mesureColor();
00315     servo.setSpeed(SERVO_SPEED_DEFAULT);
00316     switch (farbe) {
00317         case WHITE:
00318             {
00319                 nWhite++;
00320                 Serial.println("white");
00321                 legBallAb<ANGLE_LEFT_HOLE>("Wei\342");
00322                 setLedColor(255, 255, 255);
00323                 break;
00324             }
00325         case BLACK:
00326             {
00327                 nBlack++;
00328                 Serial.println("black");
00329                 legBallAb<ANGLE_RIGHT_HOLE>("Schwarz");
00330                 setLedColor(0, 0, 255);
00331                 break;
00332             }
00333         case NOTHING:
00334             {
00335                 Serial.println("nothing");
00336                 callHandler.deleteCalls();
00337                 /*static*/ auto callsNothing = new Callable*[1] {
00338                     new LcdString(String("Ball einlegen W:") + nWhite + String(" S:") + nBlack + String(" O:") +
nOrange, &lcd, 1000)
00339                 };
00340                 callHandler.setCalls(callsNothing, 1);
00341                 setLedColor(0, 255, 0);
00342                 break;
00343             }
00344         case ORANGE:
00345             {
00346                 doFlicker = true;
00347                 nOrange++;
00348                 Serial.println("orange");
00349                 callHandler.deleteCalls();
00350                 /*static*/ auto callsOrange = new Callable*[9] {
00351                     new LcdDotAnim("\xC0ra\10g\xD9ner Ba\xED1", &lcd, 0),
00352                     new FuncCall([]() {
00353                         servo.write(ANGLE_RIGHT_HOLE);
00354                     }, &servoIsDone),
00355                     new LcdString("Fehler \11rkannt", &lcd, 1000),
00356                     new FuncCall([]() {
00357                         servo.setSpeed(SERVO_SPEED_FAST);
00358                         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, 38](#)
 - [~AnimString](#)
 - [AnimString, 16](#)
 - [~Callable](#)
 - [Callable, 22](#)
 - [~FuncCall](#)
 - [FuncCall, 37](#)
 - [~LcdString](#)
 - [LcdString, 45](#)
- [ANGLE_CENTER](#)
 - [sketch.ino, 75](#)
- [ANGLE_LEFT_HOLE](#)
 - [sketch.ino, 75](#)
- [ANGLE_MIN](#)
 - [sketch.ino, 75](#)
- [ANGLE_RIGHT_HOLE](#)
 - [sketch.ino, 75](#)
- [AnimatableLcd, 9](#)
 - [animString, 15](#)
 - [doAnimation, 15](#)
 - [init, 10](#)
 - [print, 10](#)
 - [printCentered, 11](#)
 - [printPretty, 12](#)
 - [setAnimation, 13](#)
 - [update, 14](#)
- [animLcd.h, 49](#)
 - [LOADING_BAR_OFFSET, 50](#)
- [animLcd.ino, 51](#)
 - [loading_empty_c, 52](#)
 - [loading_full_c, 52](#)
- [animStart](#)
 - [AnimString, 18](#)
- [AnimString, 15](#)
 - [~AnimString, 16](#)
 - [animStart, 18](#)
 - [init, 16](#)
 - [lastRefresh, 18](#)
 - [LcdString, 17](#)
 - [run, 17](#)
 - [stepDuration, 18](#)
- [animString](#)
 - [AnimatableLcd, 15](#)
- [animString.h, 54](#)
- [animString.ino, 56](#)
- [BLACK](#)
 - [sketch.ino, 68](#)
- [ButtonHandler, 19](#)
 - [ButtonHandler, 19](#)
 - [isPressed, 20](#)
 - [onclick, 21](#)
 - [pin, 21](#)
 - [update, 20](#)
- [call](#)
 - [FuncCall, 38](#)
- [Callable, 21](#)
 - [~Callable, 22](#)
 - [isDone, 22](#)
 - [run, 22](#)
- [CallHandler, 23](#)
 - [callPtrs, 26](#)
 - [callsSet, 26](#)
 - [currCallPtr, 27](#)
 - [deleteCalls, 24](#)
 - [lastCallPtr, 27](#)
 - [lastCallT, 27](#)
 - [running, 27](#)
 - [setCalls, 24](#)
 - [update, 25](#)
- [callHandler](#)
 - [sketch.ino, 76](#)
- [callHandler.h, 58](#)
- [callHandler.ino, 59](#)
- [callPtrs](#)
 - [CallHandler, 26](#)
- [callsSet](#)
 - [CallHandler, 26](#)
- [callStart](#)
 - [LcdString, 46](#)
- [currCallPtr](#)
 - [CallHandler, 27](#)
- [CustomServo, 28](#)
 - [done, 34](#)
 - [isDone, 29](#)
 - [setSpeed, 30](#)
 - [speed, 35](#)
 - [start, 31](#)
 - [startAngle, 35](#)
 - [startMove, 31](#)
 - [startTime, 35](#)
 - [stop, 31](#)
 - [targetAngle, 35](#)
 - [updatePos, 32](#)
 - [write, 32, 33](#)
 - [writeDirect, 34](#)
- [customServo.h, 61](#)

- customServo.ino, 62
- deleteCalls
 - CallHandler, 24
- doAnimation
 - AnimatableLcd, 15
- doFlicker
 - sketch.ino, 76
- done
 - CustomServo, 34
- duration
 - LcdString, 47
- Farbe
 - sketch.ino, 68
- func_t
 - header.h, 64
- FuncCall, 36
 - _isDone, 38
 - ~FuncCall, 37
 - call, 38
 - FuncCall, 36, 37
 - isDone, 37
 - run, 37
- GEH_ZURUECK
 - sketch.ino, 67
- header.h, 64
 - func_t, 64
 - time_t, 65
- index.md, 65
- init
 - AnimatableLcd, 10
 - AnimString, 16
 - LcdDotAnim, 40
 - LcdLoadingAnim, 42
- isDone
 - Callable, 22
 - CustomServo, 29
 - FuncCall, 37
 - LcdString, 45
- isPressed
 - ButtonHandler, 20
- lastCallPtr
 - CallHandler, 27
- lastCallT
 - CallHandler, 27
- lastRefresh
 - AnimString, 18
- lcd
 - LcdString, 47
 - sketch.ino, 76
- LcdDotAnim, 39
 - init, 40
 - LcdDotAnim, 39
 - update, 40
- LcdLoadingAnim, 41
 - init, 42
 - update, 42
- LcdString, 43
 - ~LcdString, 45
 - AnimString, 17
 - callStart, 46
 - duration, 47
 - isDone, 45
 - lcd, 47
 - LcdString, 45
 - run, 45
 - text, 47
 - update, 46
- legBallAb
 - sketch.ino, 68
- LOADING_BAR_OFFSET
 - animLcd.h, 50
- LOADING_DURATION
 - sketch.ino, 76
- loading_empty_c
 - animLcd.ino, 52
- loading_full_c
 - animLcd.ino, 52
- loop
 - sketch.ino, 69
- measureColor
 - sketch.ino, 71
- nBlack
 - sketch.ino, 76
- nOrange
 - sketch.ino, 77
- NOTHING
 - sketch.ino, 68
- nWhite
 - sketch.ino, 77
- onclick
 - ButtonHandler, 21
- ORANGE
 - sketch.ino, 68
- pin
 - ButtonHandler, 21
- PIN_BLUE
 - sketch.ino, 77
- PIN_GREEN
 - sketch.ino, 77
- PIN_RED
 - sketch.ino, 77
- PIN_SERVO
 - sketch.ino, 78
- PIN_STOPBUTTON
 - sketch.ino, 78
- print
 - AnimatableLcd, 10
- printCentered
 - AnimatableLcd, 11

printPretty
 AnimatableLcd, 12

run
 AnimString, 17
 Callable, 22
 FuncCall, 37
 LcdString, 45

running
 CallHandler, 27

servo
 sketch.ino, 78
SERVO_SPEED_DEFAULT
 sketch.ino, 78
SERVO_SPEED_FAST
 sketch.ino, 78
servosDone
 sketch.ino, 72
setAnimation
 AnimatableLcd, 13
setCalls
 CallHandler, 24
setLedColor
 sketch.ino, 73
setSpeed
 CustomServo, 30
setup
 sketch.ino, 73
sketch.ino, 65
 ANGLE_CENTER, 75
 ANGLE_LEFT_HOLE, 75
 ANGLE_MIN, 75
 ANGLE_RIGHT_HOLE, 75
 BLACK, 68
 callHandler, 76
 doFlicker, 76
 Farbe, 68
 GEH_ZURUECK, 67
 lcd, 76
 legBallAb, 68
 LOADING_DURATION, 76
 loop, 69
 measureColor, 71
 nBlack, 76
 nOrange, 77
 NOTHING, 68
 nWhite, 77
 ORANGE, 68
 PIN_BLUE, 77
 PIN_GREEN, 77
 PIN_RED, 77
 PIN_SERVO, 78
 PIN_STOPBUTTON, 78
 servo, 78
 SERVO_SPEED_DEFAULT, 78
 SERVO_SPEED_FAST, 78
 servosDone, 72
 setLedColor, 73
 setup, 73
 stopButton, 79
 stopButtonClicked, 74
 WHITE, 68
 speed
 CustomServo, 35
 start
 CustomServo, 31
 startAngle
 CustomServo, 35
 startMove
 CustomServo, 31
 startTime
 CustomServo, 35
 stepDuration
 AnimString, 18
 stop
 CustomServo, 31
 stopButton
 sketch.ino, 79
 stopButtonClicked
 sketch.ino, 74
 targetAngle
 CustomServo, 35
 text
 LcdString, 47
 time_t
 header.h, 65
 update
 AnimatableLcd, 14
 ButtonHandler, 20
 CallHandler, 25
 LcdDotAnim, 40
 LcdLoadingAnim, 42
 LcdString, 46
 updatePos
 CustomServo, 32
 WHITE
 sketch.ino, 68
 write
 CustomServo, 32, 33
 writeDirect
 CustomServo, 34