

## Sortiermaschine

Erzeugt von Doxygen 1.9.5

<b>1 Sortiermaschine</b>	<b>1</b>
1.1 Zur Dokumentation	1
1.1.1 Klassen	1
1.1.2 Code	2
1.2 Begründungen	2
1.2.1 Warum werden Calls im Heap gespeichert?	2
1.2.2 Warum verschiedene Callable Klassen?	2
1.3 Optimierungsideen	4
1.3.1 Reflexion	4
<b>2 Hierarchie-Verzeichnis</b>	<b>4</b>
2.1 Klassenhierarchie	4
<b>3 Klassen-Verzeichnis</b>	<b>5</b>
3.1 Auflistung der Klassen	5
<b>4 Datei-Verzeichnis</b>	<b>6</b>
4.1 Auflistung der Dateien	6
<b>5 Klassen-Dokumentation</b>	<b>6</b>
5.1 AnimatableLcd Klassenreferenz	6
5.1.1 Ausführliche Beschreibung	6
5.1.2 Dokumentation der Elementfunktionen	8
5.1.3 Dokumentation der Datenelemente	13
5.2 AnimString Klassenreferenz	14
5.2.1 Ausführliche Beschreibung	14
5.2.2 Beschreibung der Konstruktoren und Destruktoren	15
5.2.3 Dokumentation der Elementfunktionen	15
5.2.4 Dokumentation der Datenelemente	16
5.3 ButtonHandler Klassenreferenz	17
5.3.1 Ausführliche Beschreibung	17
5.3.2 Beschreibung der Konstruktoren und Destruktoren	18
5.3.3 Dokumentation der Elementfunktionen	18
5.3.4 Dokumentation der Datenelemente	19
5.4 Callable Klassenreferenz	19
5.4.1 Ausführliche Beschreibung	19
5.4.2 Beschreibung der Konstruktoren und Destruktoren	20
5.4.3 Dokumentation der Elementfunktionen	20
5.5 CallHandler Klassenreferenz	21
5.5.1 Ausführliche Beschreibung	21
5.5.2 Dokumentation der Elementfunktionen	22
5.5.3 Dokumentation der Datenelemente	25
5.6 CustomServo Klassenreferenz	27
5.6.1 Ausführliche Beschreibung	27

5.6.2 Dokumentation der Elementfunktionen	28
5.6.3 Dokumentation der Datenelemente	34
5.7 FuncCall Strukturreferenz	35
5.7.1 Ausführliche Beschreibung	35
5.7.2 Beschreibung der Konstruktoren und Destruktoren	36
5.7.3 Dokumentation der Elementfunktionen	36
5.7.4 Dokumentation der Datenelemente	37
5.8 LcdDotAnim Klassenreferenz	38
5.8.1 Ausführliche Beschreibung	38
5.8.2 Beschreibung der Konstruktoren und Destruktoren	39
5.8.3 Dokumentation der Elementfunktionen	39
5.9 LcdLoadingAnim Klassenreferenz	40
5.9.1 Ausführliche Beschreibung	40
5.9.2 Dokumentation der Elementfunktionen	42
5.10 LcdString Strukturreferenz	43
5.10.1 Ausführliche Beschreibung	43
5.10.2 Beschreibung der Konstruktoren und Destruktoren	44
5.10.3 Dokumentation der Elementfunktionen	45
5.10.4 Dokumentation der Datenelemente	46
<b>6 Datei-Dokumentation</b>	<b>47</b>
6.1 animLcd.h-Dateireferenz	47
6.1.1 Ausführliche Beschreibung	47
6.1.2 Variablen-Dokumentation	48
6.2 animLcd.h	48
6.3 animLcd.ino-Dateireferenz	48
6.3.1 Ausführliche Beschreibung	48
6.3.2 Variablen-Dokumentation	49
6.4 animLcd.ino	50
6.5 animString.h-Dateireferenz	51
6.5.1 Ausführliche Beschreibung	51
6.6 animString.h	53
6.7 animString.ino-Dateireferenz	53
6.7.1 Ausführliche Beschreibung	53
6.8 animString.ino	54
6.9 callHandler.h-Dateireferenz	55
6.9.1 Ausführliche Beschreibung	55
6.10 callHandler.h	56
6.11 callHandler.ino-Dateireferenz	57
6.11.1 Ausführliche Beschreibung	57
6.12 callHandler.ino	58
6.13 customServo.h-Dateireferenz	58

6.13.1 Ausführliche Beschreibung . . . . .	58
6.14 customServo.h . . . . .	59
6.15 customServo.ino-Dateireferenz . . . . .	59
6.15.1 Ausführliche Beschreibung . . . . .	59
6.16 customServo.ino . . . . .	60
6.17 header.h-Dateireferenz . . . . .	61
6.17.1 Ausführliche Beschreibung . . . . .	61
6.17.2 Dokumentation der benutzerdefinierten Typen . . . . .	61
6.18 header.h . . . . .	62
6.19 index.md-Dateireferenz . . . . .	62
6.20 sketch.ino-Dateireferenz . . . . .	62
6.20.1 Ausführliche Beschreibung . . . . .	62
6.20.2 Makro-Dokumentation . . . . .	64
6.20.3 Dokumentation der Aufzählungstypen . . . . .	64
6.20.4 Dokumentation der Funktionen . . . . .	65
6.20.5 Variablen-Dokumentation . . . . .	72
6.21 sketch.ino . . . . .	76
<b>Index</b>	<b>79</b>

# 1 Sortiermaschine

Hallo, dies ist die Dokumentation für den Code der Sortiermaschine von Johannes und Arne  
[Online Dokumentation](#) (empfohlen)

## 1.1 Zur Dokumentation

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

### 1.1.1 Klassen

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

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

### 1.1.2 Code

Der Code ist interaktiv, man kann Variablen, Funktionen Methoden und Klassen anklicken um zu Ihrer Beschreibung zu gelangen.

## 1.2 Begründungen

### Zu beachten

Ich empfehle sich vor diesem Abschnitt ein wenig die Dokumentation zu "erforschen"

### 1.2.1 Warum werden Calls im Heap gespeichert?

Calls ([CallHandler::callPtrs](#)) werden mithilfe des `new` Keywords im `Heap` gespeichert um sie weiter benutzen zu können nachdem die Funktion in der sie instanziiert wurden abgeschlossen ist. Sie müssen deshalb aber auch manuell mit [CallHandler::deleteCalls](#) gelöscht werden.

### 1.2.2 Warum verschiedene Callable Klassen?

Es wäre möglich gewesen statt mehrerer `Callable` Klassen einfach eine zu benutzen und dann die Art des Calls in einer Variable zu speichern. Der Nachteil dieser Methode wäre, dass bei jeder Funktion die etwas mit der Klasse zu tun hat ([Callable::run](#), [Callable::isDone](#), etc.) überprüft werden müsste, was die Art des Calls ist. Das würde zu einer schlechteren Lesbarkeit des Codes führen. Das war in einer älteren Version der Fall.

### Code

```
class LcdHandler {
public:
    enum AnimType {
        DOT,
        LOADING,
        NO_ANIMATION
    };
    struct LcdString {
        long duration;
        AnimType animType;
        String text;
        LcdString(String text, long duration, AnimType animType = NO_ANIMATION): text(text),
            duration(duration), animType(animType) {}
        operator String() const {
            return text;
        }
    };
    bool running = false;
private:
    LcdString * currString;
    long t;
    LcdString * lastString;
    LcdString * lcdStrings;
    long stepDuration;
    long lastRefresh;
    void( * callback)(); //function pointer
public:
    ~LcdHandler() {
        delete[] lcdStrings;
    }
    void init() {
        lcd.init();
        lcd.backlight();
        lcd.createChar(0, loading_empty_c);
        lcd.createChar(1, loading_full_c);
    }
    void setStrings(LcdString newLcdStrings[], size_t numStrings, void( * newCallback)() = NULL, int
        newStepDuration = 1000) {
        delete[] lcdStrings; //Speicherplatz frei machen
        lcdStrings = newLcdStrings;
        currString = newLcdStrings;
    }
};
```

```

lastString = newLcdStrings + numStrings - 1;
t = millis();
stepDuration = newStepDuration;
lastRefresh = millis();
running = true;
callback = newCallback;
prepareAnimation(currString);
}
void prepareAnimation(LcdString* currString) {
    switch (currString->animType) {
        case LOADING:
            stepDuration = currString->duration / 9;
            if (currString->text.length() > 16) {
                Serial.print("text given for loading animation is to long, text: ");
                Serial.println(*currString);
            }
            printCentered(*currString);
            lcd.setCursor(LOADING_BAR_OFFSET, 1);
            for (int i = 0; i < 8; i++) {
                lcd.write(0);
            }
            lcd.print("0% ");
            break;
        case DOT:
            printPretty(currString->text + String("  "));
            break;
        default:
            printPretty(*currString);
    }
}
void printCentered(String text, int length = -1, int row = 0) { //length<=16
    if (length == -1) {
        length = text.length();
    }
    int offset = (16 - length) / 2; //rundet immer ab, da int
    lcd.setCursor(offset, row);
    lcd.print(text);
}
void printPretty(String text) { //handelt zeilenumbrüche und schreibt zentriert
    lcd.clear();
    int length = text.length();
    if (length <= 16) {
        printCentered(text, length);
        return 0;
    }
    int spacePos = -1;
    for (int i = 15; i >= 0; i--) {
        if (text[i] == ' ') {
            spacePos = i;
            break;
        }
    }
    String row1, row2;
    if (spacePos != -1) {
        row1 = text.substring(0, spacePos);
        row2 = text.substring(spacePos + 1);
    } else {
        row1 = text.substring(0, 16);
        row2 = text.substring(16);
    }
    printCentered(row1, row1.length(), 0);
    printCentered(row2, row2.length(), 1);
}
void printAnimated() {
    long time = millis();
    AnimType type = currString->animType;
    if (type == DOT) {
        if ((time - lastRefresh) < stepDuration) {
            return;
        }
        lastRefresh = time;
        int numDots = ((time - t) / stepDuration) % 4;
        char dots[4];
        for (int i = 0; i < 3; i++) {
            if (i < numDots) {
                dots[i] = '.';
            } else {
                dots[i] = ' ';
            }
        }
        dots[3] = '\0';
        printPretty(currString->text + dots);
    } else if (type == LOADING) {
        short percent = (time - t) * 100 / currString->duration;
        if (time - lastRefresh > stepDuration) {
            short nToFill = percent * 8 / 100;
            if (nToFill == 0) {

```

```

        return;
    }
    lcd.setCursor(nToFill + LOADING_BAR_OFFSET - 1, 1);
    lcd.write(1);
    lastRefresh = time;
}
lcd.setCursor(8 + LOADING_BAR_OFFSET, 1);
lcd.print(percent);
lcd.print("%");
} else {
    Serial.println("unknown animation type");
}
}
void animate() {
    if (!running) {
        return;
    }
    int timePassed = millis() - t;
    if (timePassed > currString -> duration) {
        if (currString + 1 > lastString) {
            running = false;
            if (callback != NULL) {
                callback();
            }
            return;
        }
        currString++;
        t = millis();
        prepareAnimation(currString);
        return;
    }
    if (currString->animType == NO_ANIMATION) {
        return;
    }
    printAnimated();
}
};

```

## 1.3 Optimierungsideen

Beim Ausführen des Programms mangelte es manchmal an Speicherplatz. Um dieses Problem zu umgehen hätte man statt der **Arduino String Klasse** auch C-Strings also `char []` benutzen können. Das hätte den Code leider aber auch an ein Paar Stellen komplizierter gemacht. Ein anderer Fehler der sehr häufig auftrat war, dass mit Speicheradressen falsch umgegangen wurde und dadurch das Programm ohne jegliche Fehlermeldung abstürzte, oft sogar an völlig anderen Stellen. Eine mögliche Lösung wäre es hier eine Programmiersprache zu benutzen, die Speicherplatzfehler beim kompilieren, das heißt beim umwandeln des Programmcodes in Maschinencode aufdeckt. Eine Möglichkeit dafür wäre diese **Arduino Bibliothek für Rust ( Beispiel)**. Solche Bibliotheken sind aber auch weniger Dokumentiert und haben weniger Features als die Standard C++ Arduino Bibliothek und deshalb alles in allem für ein kleines Schulprojekt keine gute Option.

### 1.3.1 Reflexion

Nächstes mal immer mehrmals den Code lesen den man geschrieben hat, wenn es um Pointer geht. Heap bestmöglichst vermeiden.

## 2 Hierarchie-Verzeichnis

### 2.1 Klassenhierarchie

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

**ButtonHandler**

**17**

<b>Callable</b>	<b>19</b>
<b>FuncCall</b>	<b>35</b>
<b>LcdString</b>	<b>43</b>
<b>AnimString</b>	<b>14</b>
<b>LcdDotAnim</b>	<b>38</b>
<b>LcdLoadingAnim</b>	<b>40</b>
<b>CallHandler</b>	<b>21</b>
LiquidCrystal_I2C	
<b>AnimatableLcd</b>	<b>6</b>
Servo	
<b>CustomServo</b>	<b>27</b>

## 3 Klassen-Verzeichnis

### 3.1 Auflistung der Klassen

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

<b>AnimatableLcd</b>	
Eigener Lcd, ermöglicht es Animationen auf dem Lcd Display anzuzeigen	<b>6</b>
<b>AnimString</b>	
Die Klasse für animierbare LcdStrings	<b>14</b>
<b>ButtonHandler</b>	
Kleine Klasse die Knopfdrücke verarbeitet	<b>17</b>
<b>Callable</b>	
Ein Call der vom <b>CallHandler</b> aufgerufen werden kann	<b>19</b>
<b>CallHandler</b>	
Klasse, die Calls nacheinander aufruft	<b>21</b>
<b>CustomServo</b>	
Eine erweiterte Version der <b>Servo-Klasse</b> , die es ermöglicht den Servo mit verschiedenen Geschwindigkeiten zu bewegen	<b>27</b>
<b>FuncCall</b>	
Ein Call der eine Funktion ausführt	<b>35</b>
<b>LcdDotAnim</b>	
Die Klasse der Lcd Punktanimationen	<b>38</b>
<b>LcdLoadingAnim</b>	
Die Klasse der Lcd Ladeanimationen	<b>40</b>
<b>LcdString</b>	
Ein String der auf dem <b>AnimatableLcd</b> angezeigt werden kann	<b>43</b>



## 4 Datei-Verzeichnis

### 4.1 Auflistung der Dateien

Hier folgt die Aufzählung aller Dateien mit einer Kurzbeschreibung:

<a href="#">animLcd.h</a>	Header-Datei für den animierbaren lcd ( <a href="#">AnimatableLcd</a> )	47
<a href="#">animLcd.ino</a>	Implementation für die <a href="#">AnimatableLcd</a> Klasse	48
<a href="#">animString.h</a>	Header datei für eine Mehrzahl von animierbaren Strings und der <a href="#">Callable</a> Klasse	51
<a href="#">animString.ino</a>	Implementationen der <a href="#">Callable</a> und <a href="#">LcdString</a> Klassen	53
<a href="#">callHandler.h</a>	Header datei für den <a href="#">CallHandler</a>	55
<a href="#">callHandler.ino</a>	Umsetzung der <a href="#">CallHandler</a> Klasse	57
<a href="#">customServo.h</a>	Header Datei der <a href="#">CustomServo</a> Klasse	58
<a href="#">customServo.ino</a>	Umsetzung der <a href="#">CustomServo</a> Klasse	59
<a href="#">header.h</a>	Definiert variablen-types die überall im Programm benutzt werden	61
<a href="#">sketch.ino</a>	Hauptdatei, wichtigste Funktionen sind <a href="#">setup()</a> und <a href="#">loop()</a>	62

## 5 Klassen-Dokumentation

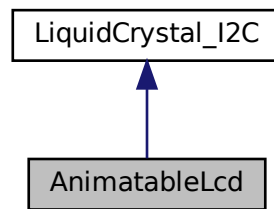
### 5.1 AnimatableLcd Klassenreferenz

#### 5.1.1 Ausführliche Beschreibung

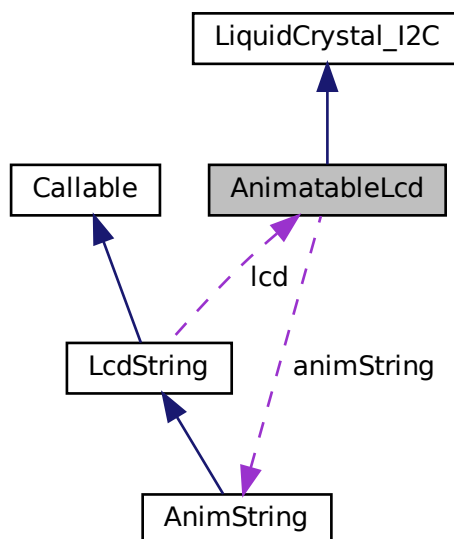
Eigener Lcd, ermöglicht es Animationen auf dem Lcd Display anzuzeigen.

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

Klassendiagramm für AnimatableLcd:



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.*
- void `print` (const String &&text)

### Öffentliche Attribute

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

### Private Attribute

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

## 5.1.2 Dokumentation der Elementfunktionen

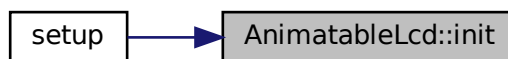
### 5.1.2.1 `init()` void AnimatableLcd::init ( )

Überschreibt die normale lcd init function.

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

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

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



### 5.1.2.2 print() [1/2] void AnimatableLcd::print (const String && text )

Definiert in Zeile 83 der Datei [animLcd.ino](#).

```
00084 {
00085     print(text);
00086 }
```

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



### 5.1.2.3 print() [2/2] 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

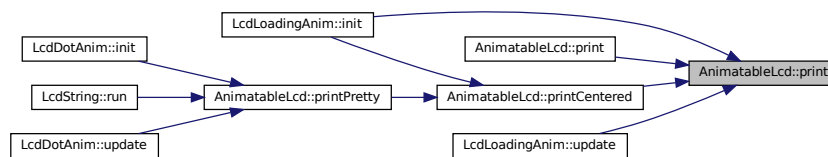
#### Parameter

<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 (\1n for the nth character)
00069     //and it will be converted to the custom character (\1n so that \0 doesn't appear in the string,
    because it means end of string)
00070     for(char c:text){
00071         if(c>=8&&c<=15){//if it is a custom character
00072             write(c-8);
00073         }else if(c==1){//defining a non-newline space
00074             LiquidCrystal_I2C::print(" ");
00075         }else if(c==2){//defining a "random" character https://arduino.stackexchange.com/a/46833
00076             LiquidCrystal_I2C::print(String((char) random(33,256)));
00077         }
00078         else{
00079             LiquidCrystal_I2C::print(c);
00080         }
00081     }
00082 }
```

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



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

Gibt einen String zentriert auf dem Lcd-Display aus.

#### Parameter

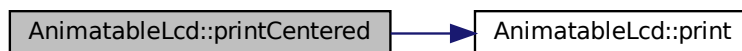
<i>text</i>	
<i>length</i>	Länge des Textes, wird neu berechnet wenn Nichts angegeben
<i>row</i>	Zeile in der der Text ausgegeben werden soll

Definiert in Zeile 94 der Datei [animLcd.ino](#).

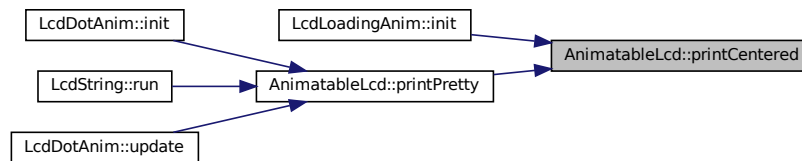
```

00095 {
00096   if (length == -1) {
00097     length = text.length();
00098   }
00099   int offset = (16 - length) / 2; //rundet immer ab, da int
00100   setCursor(offset, row);
00101   print(text);
00102 }
  
```

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

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

Parameter

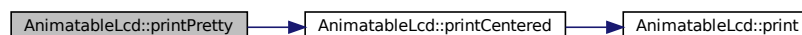
<i>text</i>	
-------------	--

Definiert in Zeile 108 der Datei [animLcd.ino](#).

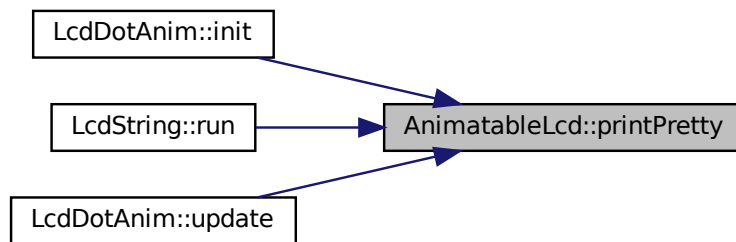
```

00109 {
00110   clear();
00111   int length = text.length();
00112   if (length <= 16) {
00113     printCentered(text, length);
00114     return 0;
00115   }
00116   int spacePos = -1;
00117   for (int i = 15; i >= 0; i--) {
00118     if (text[i] == ' ') {
00119       spacePos = i;
00120       break;
00121     }
00122   }
00123   String row1, row2;
00124   if (spacePos != -1) {
00125     row1 = text.substring(0, spacePos);
00126     row2 = text.substring(spacePos + 1);
00127   } else {
00128     row1 = text.substring(0, 16);
00129     row2 = text.substring(16);
00130   }
00131   printCentered(row1, row1.length(), 0);
00132   printCentered(row2, row2.length(), 1);
00133 }
  
```

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



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



**5.1.2.6 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:



**5.1.2.7 update()** `void AnimatableLcd::update ( )`

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

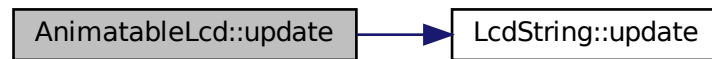
Definiert in Zeile 138 der Datei `animLcd.ino`.

```

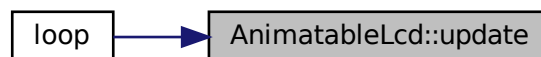
00139 {
  
```

```
00140     if (!doAnimation) {  
00141         return;  
00142     }  
00143     animString->update();  
00144 }
```

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



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



### 5.1.3 Dokumentation der Datenelemente

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

Die zurzeit laufende Animation.

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

#### 5.1.3.2 doAnimation `bool AnimatableLcd::doAnimation = false`

Gibt an, ob der Monitor animiert werden soll.

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

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

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



## 5.2 AnimString Klassenreferenz

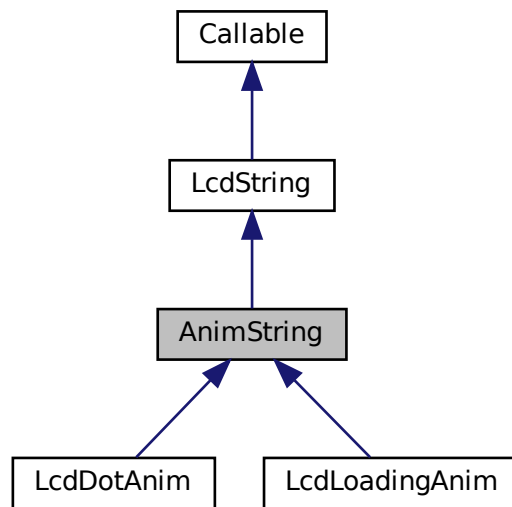
### 5.2.1 Ausführliche Beschreibung

Die Klasse für animierbare LcdStrings.

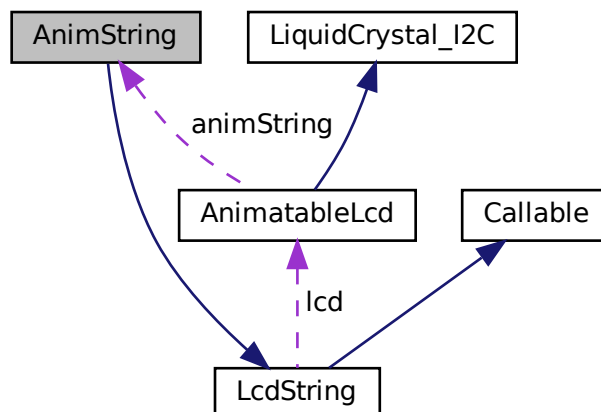
Wird nie selbst instanziiert aber Lcd Animationen erben von dieser Klasse

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

Klassendiagramm für AnimString:



Zusammengehörigkeiten von AnimString:



### Öffentliche Methoden

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

### Geschützte Attribute

- [time\\_t](#) [stepDuration](#)  
*wie lange ein Schritt der Animation dauert*
- [time\\_t](#) [animStart](#)  
*wann die animation begann*
- [time\\_t](#) [lastRefresh](#)  
*wann das letzte mal die Anzeige erneuert wurde*

### Weitere Geerbte Elemente

#### 5.2.2 Beschreibung der Konstruktoren und Destruktoren

##### 5.2.2.1 [~AnimString\(\)](#) virtual AnimString::~~AnimString ( ) [inline], [virtual]

Definiert in Zeile [95](#) der Datei [animString.h](#).  
00095 {}

#### 5.2.3 Dokumentation der Elementfunktionen

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

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

Definiert in Zeile [96](#) der Datei [animString.h](#).  
00096 {}

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



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

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

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

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

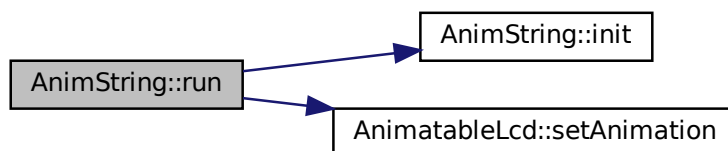
setzt Variablen die für alle Animationen notwendig sind und ruft dann ihre eigenen init Funktionen auf

Erneute Implementation von [LcdString](#).

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

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

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



## 5.2.4 Dokumentation der Datenelemente

**5.2.4.1 animStart** `time_t AnimString::animStart [protected]`

wann die animation begann

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

#### 5.2.4.2 lastRefresh `time_t` `AnimString::lastRefresh` [protected]

wann das letzte mal die Anzeige erneuert wurde

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

#### 5.2.4.3 stepDuration `time_t` `AnimString::stepDuration` [protected]

wie lange ein Schritt der Animation dauert

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

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

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

## 5.3 ButtonHandler Klassenreferenz

### 5.3.1 Ausführliche Beschreibung

Kleine Klasse die Knopfdrücke verarbeitet.

Definiert in Zeile 108 der Datei [sketch.ino](#).

#### Öffentliche Methoden

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

#### Öffentliche Attribute

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

#### Private Attribute

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

### 5.3.2 Beschreibung der Konstruktoren und Destruktoren

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

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

```
00125 {}
```

#### 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 132 der Datei [sketch.ino](#).

```
00132 : pin(pin), onclick(onclick) {
00133     pinMode(pin, INPUT_PULLUP);
00134 }
```

### 5.3.3 Dokumentation der Elementfunktionen

#### 5.3.3.1 update() void ButtonHandler::update ( ) [inline]

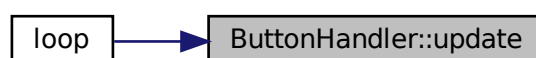
Prüft, ob der Knopf gedrückt/losgelassen wurde.

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

Definiert in Zeile 139 der Datei [sketch.ino](#).

```
00140 {
00141     bool isPressedNew = digitalRead(pin) == HIGH;
00142     if (isPressedNew != isPressed) { //is not being pressed now, but was being pressed
00143         if (isPressed) {
00144             Serial.println("click");
00145             onclick();
00146         }
00147     }
00148     isPressed = isPressedNew;
00149 }
```

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 118 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 124 der Datei [sketch.ino](#).

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

Der Pin an dem der Knopf angeschlossen ist.

Definiert in Zeile 113 der Datei [sketch.ino](#).

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

- [sketch.ino](#)

## 5.4 Callable Klassenreferenz

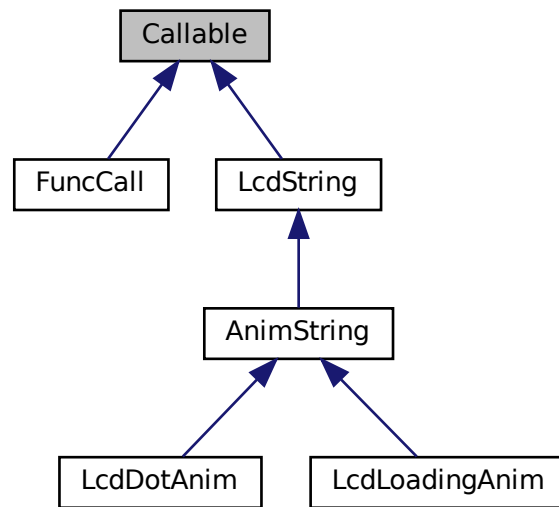
### 5.4.1 Ausführliche Beschreibung

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

Wird nie selbst instanziiert sondern nur Abgeleitete Klassen

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

Klassendiagramm für Callable:



#### Private Methoden

- virtual void `run()`
- virtual bool `isDone()`
- virtual `~Callable()`

### 5.4.2 Beschreibung der Konstruktoren und Destruktoren

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

Definiert in Zeile 27 der Datei `animString.h`.

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

### 5.4.3 Dokumentation der Elementfunktionen

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

Erneute Implementation in `FuncCall` und `LcdString`.

Definiert in Zeile 26 der Datei `animString.h`.

```
00026 {}
```

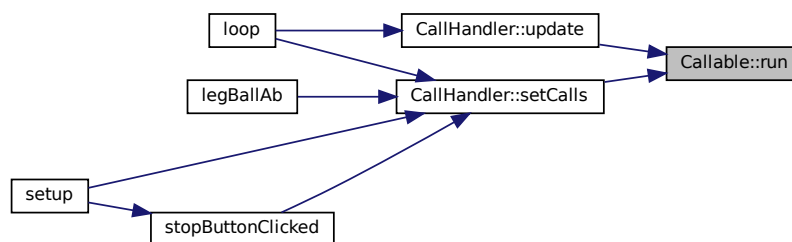
**5.4.3.2 run()** `virtual void Callable::run ( ) [inline], [private], [virtual]`

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

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

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

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



Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

- [animString.h](#)

## 5.5 CallHandler Klassenreferenz

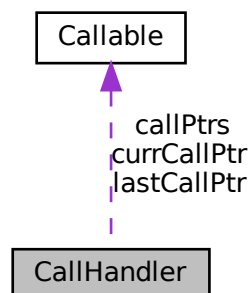
### 5.5.1 Ausführliche Beschreibung

Klasse, die Calls nacheinander aufruft.

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

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

Zusammengehörigkeiten von CallHandler:





## Öffentliche Methoden

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

## Öffentliche Attribute

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

## Private Attribute

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

## 5.5.2 Dokumentation der Elementfunktionen

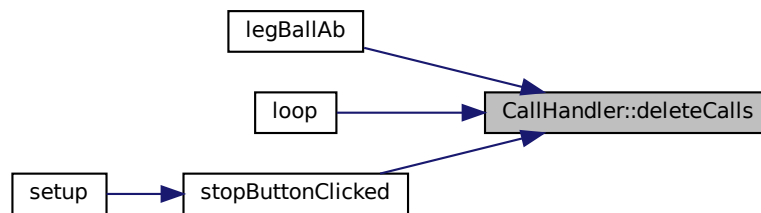
### 5.5.2.1 `deleteCalls()` void CallHandler::deleteCalls ( )

setzt den Speicherplatz der von den Calls besetzt wurde mithilfe von delete 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.

#### Zu beachten

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). Sie müssen aber auch manuell mithilfe von **CallHandler::deleteCalls** gelöscht werden

#### Warnung

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

#### Parameter

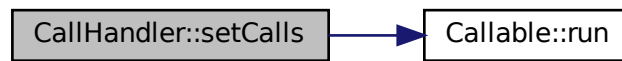
<code>newCallPtrs</code>	
<code>nCalls</code>	

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

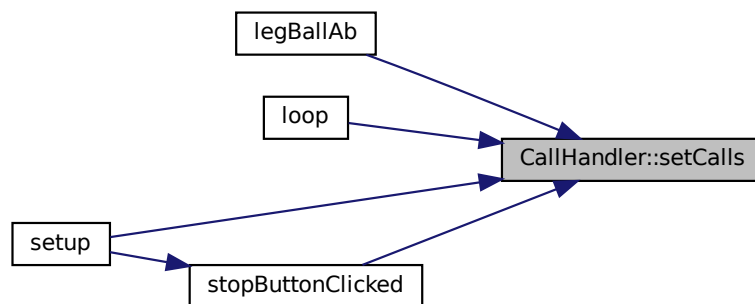
```

00031 {
00032     /*if(callsSet){ //doing this would result in two sets of calls being in heap at once
00033         deleteCalls(); //solution is to delete previous calls before initializing a new one
00034     }*/
00035     callPtrs = newCallPtrs;
00036     callsSet = true;
00037     currCallPtr = callPtrs;
00038     (*currCallPtr)->run();
00039     lastCallPtr = callPtrs + nCalls - 1;
00040     lastCallT = millis();
00041     running = true;
00042 }
```

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



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



### 5.5.2.3 update() void CallHandler::update ( )

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

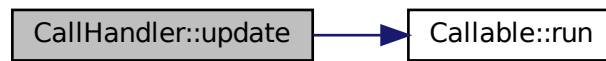
Wird von `loop()` aufgerufen

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

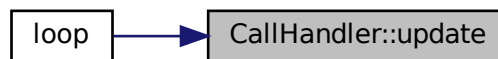
```

00048 {
00049     if (!running) {
00050         return;
00051     }
00052     time_t timePassed = millis() - lastCallT;
00053     if ((*currCallPtr)->isDone()) { //->currCall->isDone()
00054         if (currCallPtr == lastCallPtr) {
00055             running = false;
00056             return;
00057         }
00058         currCallPtr++;
00059         (*currCallPtr)->run(); //->currCall->run();
00060     }
00061 }
  
```

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



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



### 5.5.3 Dokumentation der Datenelemente

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

Die liste der aktuellen Calls.

Wird mithilfe von `CallHandler::setCalls` gesetzt.

##### Zu beachten

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

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

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

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

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

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

Der Call der zurzeit ausgeführt wird.

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

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

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

Der letzte Call.

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

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

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

Der Zeitpunkt an dem der letzte Call ausgeführt wurde.

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

**5.5.3.6 running** `bool CallHandler::running = false`

Gibt an, ob der `CallHandler` fertig ist.

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

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

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

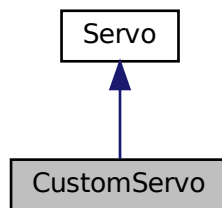
## 5.6 CustomServo Klassenreferenz

### 5.6.1 Ausführliche Beschreibung

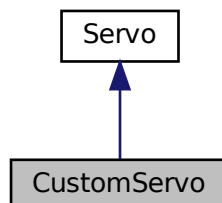
Eine erweiterte Version der `Servo-Klasse`, die es ermöglicht den Servo mit verschiedenen Geschwindigkeiten zu bewegen.

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

Klassendiagramm für CustomServo:



Zusammengehörigkeiten von CustomServo:



### Öffentliche Methoden

- void `write` (short newAngle)  
*Bewegt den Servo mit einer vorher spezifizierten Geschwindigkeit.*
- void `write` (short newAngle, `time_t` duration)  
*Bewegt den Servo in duration Millisekunden an den angegebenen Winkel.*
- void `writeDirect` (short angle)  
*Steuert den Servo direkt an, entspricht dem normalen `Servo::write`*
- void `setSpeed` (float newSpeed)  
*Setzt eine neue Geschwindigkeit des Servos.*
- void `updatePos` ()

- Aktualisiert die Position des Servomotors.*
- void `stop` ()  
*Stoppt den Servo.*
- void `start` ()  
*Lässt den Servo weiterlaufen.*
- bool `isDone` ()  
*Gibt an, ob der Servo angekommen ist.*

### Öffentliche Attribute

- bool `done` =true

### Private Methoden

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

### Private Attribute

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

## 5.6.2 Dokumentation der Elementfunktionen

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

Gibt an, ob der Servo angekommen ist.

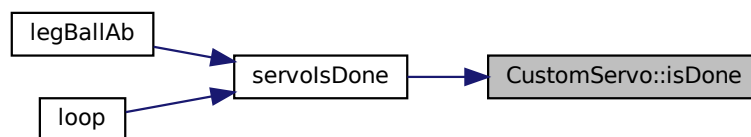
**Rückgabe**

true  
false

Definiert in Zeile 91 der Datei [customServo.ino](#).

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

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



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

Setzt eine neue Geschwindigkeit des Servos.

Kann auch ausgeführt werden während der Servo sich schon bewegt

**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 }
```

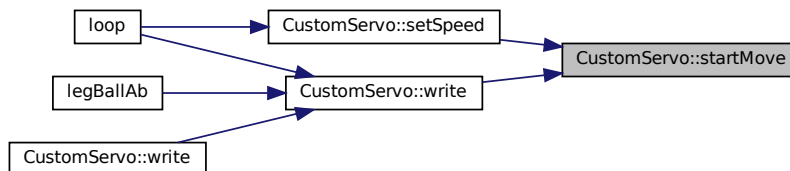
#### 5.6.2.4 `startMove()` `void CustomServo::startMove ( ) [private]`

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:



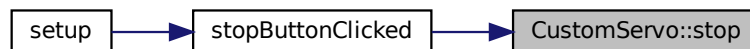
### 5.6.2.5 stop() void CustomServo::stop ( )

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 ( )

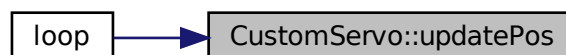
Aktualisiert die Position des Servomotors.

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.

#### Warnung

Winkel überprüfen! Wenn dieser zu klein ist schlägt der Arm gegen den Stopper

#### Parameter

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

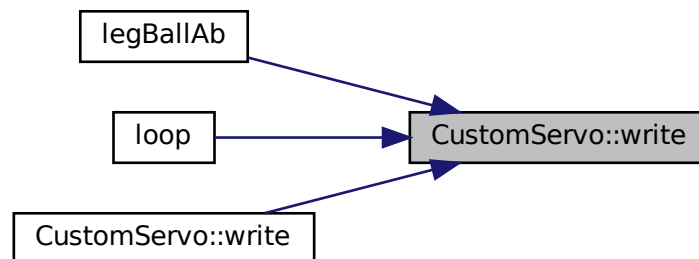
Definiert in Zeile 22 der Datei [customServo.ino](#).

```
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.

#### Warnung

Winkel überprüfen! Wenn dieser zu klein ist schlägt der Arm gegen den Stopper

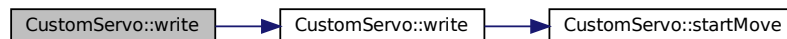
#### Parameter

<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`

#### Warnung

Winkel überprüfen! Wenn dieser zu klein ist schlägt der Arm gegen den Stopper

#### Parameter

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

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

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

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



### 5.6.3 Dokumentation der Datenelemente

**5.6.3.1 done** `bool CustomServo::done =true`

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

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

Die Geschwindigkeit des Servos in Grad pro Millisekunde.

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

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

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

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

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

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

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

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

Der Zielwinkel.

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

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

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

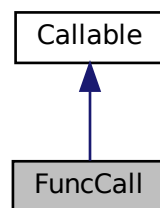
## 5.7 FuncCall Strukturreferenz

### 5.7.1 Ausführliche Beschreibung

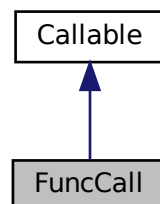
Ein Call der eine Funktion ausführt.

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

Klassendiagramm für FuncCall:



Zusammengehörigkeiten von FuncCall:



## Öffentliche Methoden

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

*ruft die angegebene Funktion auf*

- `bool isDone ()`

*Gibt zurück, ob der nächste Call ausgeführt werden sollte.*

## Öffentliche Attribute

- `func_t< void > call`

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

- `func_t< bool > _isDone`

*bestimmt, ob dieser Call vorbei ist*

## 5.7.2 Beschreibung der Konstruktoren und Destruktoren

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

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

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

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

Definiert in Zeile 46 der Datei `animString.h`.

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

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

Definiert in Zeile 48 der Datei `animString.h`.

```
00048 {}
```

## 5.7.3 Dokumentation der Elementfunktionen

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

Gibt zurück, ob der nächste Call ausgeführt werden sollte.

**Rückgabe**

true

false

Erneute Implementation von [Callable](#).

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

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

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

ruft die angegebene Funktion auf

Erneute Implementation von [Callable](#).

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

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

**5.7.4 Dokumentation der Datenelemente****5.7.4.1 \_isDone** `func_t<bool> FuncCall::_isDone`

bestimmt, ob dieser Call vorbei ist

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

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

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

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

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Dateien:

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



## 5.8 LcdDotAnim Klassenreferenz

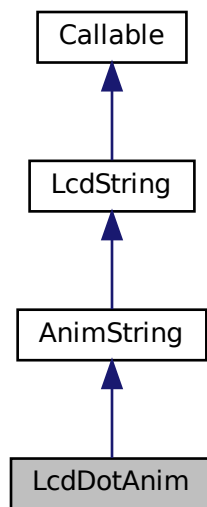
### 5.8.1 Ausführliche Beschreibung

Die Klasse der Lcd Punktanimationen.

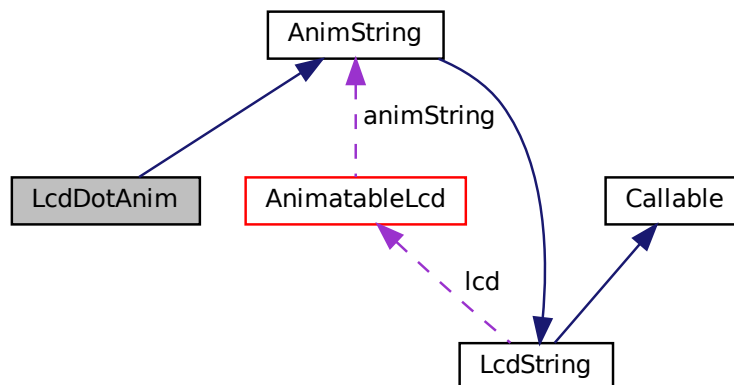
zeigt immer wieder keinen, dann einen, dann zwei, dann drei und letztendlich wieder keinen Punkt nach dem Text an

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

Klassendiagramm für LcdDotAnim:



Zusammengehörigkeiten von LcdDotAnim:



## Öffentliche Methoden

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

## Weitere Geerbte Elemente

### 5.8.2 Beschreibung der Konstruktoren und Destruktoren

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

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

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

### 5.8.3 Dokumentation der Elementfunktionen

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

initialisiert die Punktanimation

Erneute Implementation von `AnimString`.

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

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

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



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

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

Erneute Implementation von [LcdString](#).

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

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

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



Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

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

## 5.9 LcdLoadingAnim Klassenreferenz

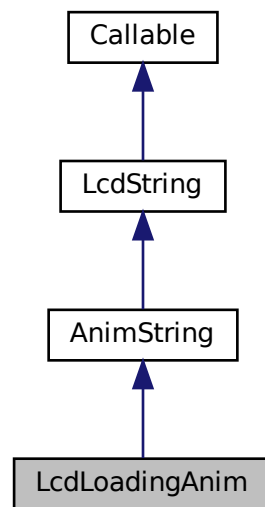
### 5.9.1 Ausführliche Beschreibung

Die Klasse der Lcd Ladeanimationen.

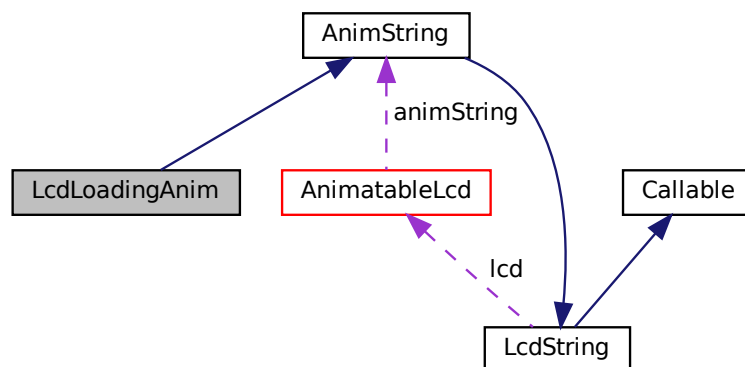
Zeigt Acht Ladebalken und den Fortschritt in Prozent an

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

Klassendiagramm für LcdLoadingAnim:



Zusammengehörigkeiten von LcdLoadingAnim:



### Öffentliche Methoden

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

## Weitere Geerbte Elemente

### 5.9.2 Dokumentation der Elementfunktionen

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

initialisierung der Ladeanimation

Erneute Implementation von [AnimString](#).

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

```
00065 {
00066     stepDuration = duration / 9;
00067     if (text.length() > 16) {
00068         Serial.print("warning: text given for loading animation is 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

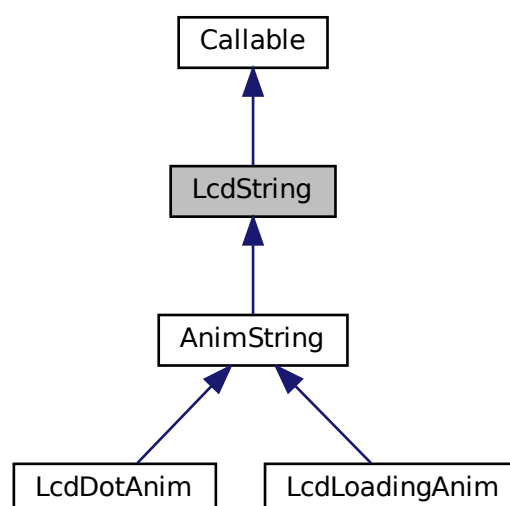
### 5.10.1 Ausführliche Beschreibung

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

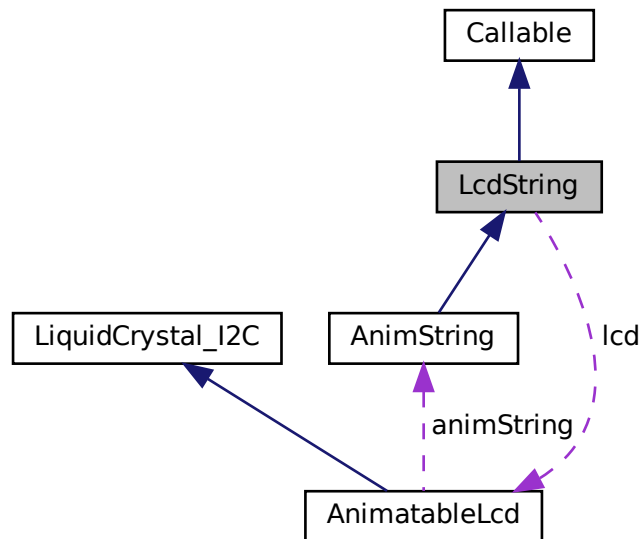
Animationen werden von dieser Klasse abgeleitet

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

Klassendiagramm für LcdString:



Zusammengehörigkeiten von LcdString:



### Öffentliche Methoden

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

### Öffentliche Attribute

- String text  
der Text der angezeigt wird
- AnimatableLcd \* lcd
- time\_t duration
- time\_t callStart

## 5.10.2 Beschreibung der Konstruktoren und Destruktoren

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

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

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

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

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

```
00067 {}
```

### 5.10.3 Dokumentation der Elementfunktionen

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

Gibt zurück, ob die duration überschritten ist.

Rückgabe

true

false

Erneute Implementation von [Callable](#).

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

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

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

gibt den String auf dem Lcd-Display aus

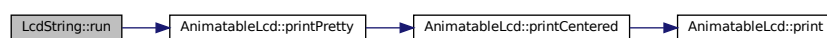
Erneute Implementation von [Callable](#).

Erneute Implementation in [AnimString](#).

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

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

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





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

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

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

```
00070 {}
```

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



## 5.10.4 Dokumentation der Datenelemente

**5.10.4.1 callStart** `time_t LcdString::callStart`

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

**5.10.4.2 duration** `time_t LcdString::duration`

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

**5.10.4.3 lcd** `AnimatableLcd* LcdString::lcd`

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

**5.10.4.4 text** `String LcdString::text`

der Text der angezeigt wird

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

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Dateien:

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

## 6 Datei-Dokumentation

### 6.1 animLcd.h-Dateireferenz

#### 6.1.1 Ausführliche Beschreibung

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

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

```
#include <LiquidCrystal_I2C.h>
#include "animString.h"
```

Include-Abhängigkeitsdiagramm für animLcd.h:



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



## Klassen

- class [AnimatableLcd](#)

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

## Variablen

- const int [LOADING\\_BAR\\_OFFSET](#) = 2

### 6.1.2 Variablen-Dokumentation

#### 6.1.2.1 [LOADING\\_BAR\\_OFFSET](#) const int [LOADING\\_BAR\\_OFFSET](#) = 2

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

## 6.2 animLcd.h

[gehe zur Dokumentation dieser Datei](#)

```
00001
00006 #ifndef ANIMLCD_H
00007 #define ANIMLCD_H
00008 #include <LiquidCrystal_I2C.h>
00009 class AnimatableLcd;
00010 #include "animString.h"
00011
00012 const int LOADING_BAR_OFFSET = 2;
00017 class AnimatableLcd: public LiquidCrystal_I2C {
00022     AnimString* animString;
00023 public:
00028     bool doAnimation = false;
00029     using LiquidCrystal_I2C::LiquidCrystal_I2C; //using the LiquidCrystal constructor
00030     void setAnimation(AnimString* _animString);
00031     void printCentered(String text, int length = -1, int row = 0);
00032     void printPretty(String text);
00033     void update();
00034     void init();
00035     using LiquidCrystal_I2C::print; //übernehme den Standart print befehl
00036     void print(const String& text); //überschreibt den standart print Befehl für String& (text wird
    als Variable übergeben, z.B. lcd.print(text);)
00037     void print(const String&& text); //und String&& (text wird direkt als Argument gegeben, z.B.
    lcd.print("...");)
00038 };
00039 #endif
```

## 6.3 animLcd.ino-Dateireferenz

### 6.3.1 Ausführliche Beschreibung

Implementation für die [AnimatableLcd](#) Klasse.

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

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

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

Include-Abhängigkeitsdiagramm für animLcd.ino:



## Variablen

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

### 6.3.2 Variablen-Dokumentation

#### 6.3.2.1 `loading_empty_c` const byte loading\_empty\_c[8]

##### Initialisierung:

```

= {
    B11111,
    B10001,
    B10001,
    B10001,
    B10001,
    B10001,
    B10001,
    B10001,
    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         }else if(c==2){//defining a "random" character https://arduino.stackexchange.com/a/46833
00076             LiquidCrystal_I2C::print(String((char)random(33,256)));
00077         }
00078         else{
00079             LiquidCrystal_I2C::print(c);
00080         }
00081     }
00082 }
```

```

00083 void AnimatableLcd::print(const String&& text)
00084 {
00085     print(text);
00086 }
00094 void AnimatableLcd::printCentered(String text, int length = -1, int row = 0) //length<=16
00095 {
00096     if (length == -1) {
00097         length = text.length();
00098     }
00099     int offset = (16 - length) / 2; //rundet immer ab, da int
00100     setCursor(offset, row);
00101     print(text);
00102 }
00108 void AnimatableLcd::printPretty(String text) //handelt zeilenumbrüche und schreibt zentriert
00109 {
00110     clear();
00111     int length = text.length();
00112     if (length <= 16) {
00113         printCentered(text, length);
00114         return 0;
00115     }
00116     int spacePos = -1;
00117     for (int i = 15; i >= 0; i--) {
00118         if (text[i] == ' ') {
00119             spacePos = i;
00120             break;
00121         }
00122     }
00123     String row1, row2;
00124     if (spacePos != -1) {
00125         row1 = text.substring(0, spacePos);
00126         row2 = text.substring(spacePos + 1);
00127     } else {
00128         row1 = text.substring(0, 16);
00129         row2 = text.substring(16);
00130     }
00131     printCentered(row1, row1.length(), 0);
00132     printCentered(row2, row2.length(), 1);
00133 }
00138 void AnimatableLcd::update()
00139 {
00140     if (!doAnimation) {
00141         return;
00142     }
00143     animString->update();
00144 }

```

## 6.5 animString.h-Dateireferenz

### 6.5.1 Ausführliche Beschreibung

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

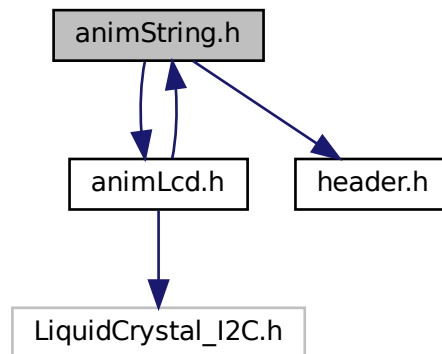
Definiert in Datei [animString.h](#).

```

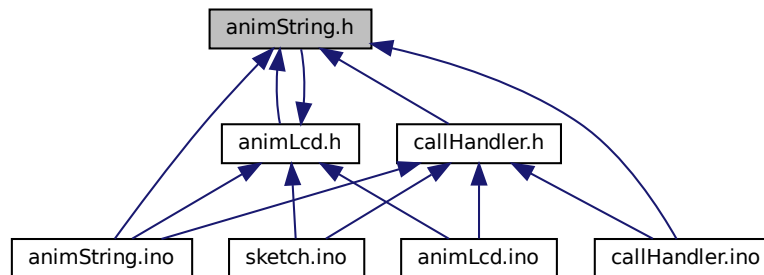
#include "animLcd.h"
#include "header.h"

```

Include-Abhängigkeitsdiagramm für animString.h:



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



## Klassen

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

## 6.6 animString.h

[gehe zur Dokumentation dieser Datei](#)

```

00001
00006 //the implementations for the animatable string class
00007 //animatable strings are strings that can be passed to the animatable lcd
00008 #ifndef ANIMSTRING_H
00009 #define ANIMSTRING_H
00010 struct Callable;
00011 struct FuncCallable;
00012 struct LcdString;
00013 class LcdLoadingAnim;
00014 class LcdDotAnim;
00015
00016 #include "animLcd.h"
00017 #include "header.h"
00018
00024 class Callable {
00025     virtual void run() {} //virtual->must be implemented by derived classes
00026     virtual bool isDone() {}
00027     virtual ~Callable() {} //let's derived classes free their own memory. ~functions are called when the
        object is deleted
00028 };
00033 struct FuncCall: public Callable {
00038     func_t<void> call;
00043     func_t<bool> _isDone;
00044     FuncCall(func_t<void> call, func_t<bool> _isDone):
00045         call(call), _isDone(_isDone) {}
00046     FuncCall(func_t<void> call): //when no isDone function is provided, isDone defaults to true
00047         call(call), _isDone([]() {return true;}) {}
00048     virtual ~FuncCall() {}
00049     void run();
00050     bool isDone();
00051 };
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

### 6.7.1 Ausführliche Beschreibung

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



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

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

Include-Abhängigkeitsdiagramm für animString.ino:



## 6.8 animString.ino

[gehe zur Dokumentation dieser Datei](#)

```
00001
00006 #include "animLcd.h"
00007 #include "callHandler.h"
00008 #include "header.h"
00009 #include "animString.h"
00014 void FuncCall::run()
00015 {
00016     call();
00017 }
00024 bool FuncCall::isDone()
00025 {
00026     return _isDone();
00027 }
00032 void LcdString::run()
00033 {
00034     callStart = millis();
00035     lcd->doAnimation = false;
00036     lcd->printPretty(this->text);
00037 }
00044 bool LcdString::isDone()
00045 {
00046     return millis() - callStart > duration;
00047 }
00052 void AnimString::run()
00053 {
00054     callStart = millis();
00055     lcd->clear();
00056     lcd->setAnimation(this);
00057     animStart = millis();
00058     lastRefresh = millis();
00059     init();
00060 }
00064 void LcdLoadingAnim::init()
00065 {
```

```

00066     stepDuration = duration / 9;
00067     if (text.length() > 16) {
00068         Serial.print("warning: text given for loading animation is 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     }
00127     dots[3] = '\0';
00128     lcd->printPretty(text + dots);
00129 }

```

## 6.9 callHandler.h-Dateireferenz

### 6.9.1 Ausführliche Beschreibung

header datei für den [CallHandler](#)

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

```

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

```

Include-Abhängigkeitsdiagramm für callHandler.h:



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



## Klassen

- class `CallHandler`

*Klasse, die Calls nacheinander aufruft.*

## 6.10 callHandler.h

[gehe zur Dokumentation dieser Datei](#)

```

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

```

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

```

## 6.11 callHandler.ino-Dateireferenz

### 6.11.1 Ausführliche Beschreibung

Umsetzung der [CallHandler](#) Klasse.

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

```

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

```

Include-Abhängigkeitsdiagramm für callHandler.ino:



## 6.12 callHandler.ino

gehe zur Dokumentation dieser Datei

```

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

```

## 6.13 customServo.h-Dateireferenz

### 6.13.1 Ausführliche Beschreibung

Header Datei der [CustomServo](#) Klasse.

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

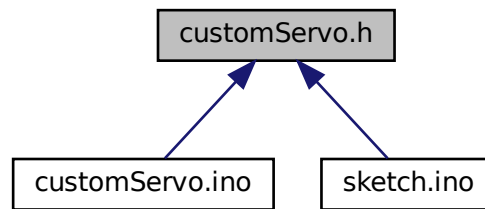
```
#include <Servo.h>
```

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

Include-Abhängigkeitsdiagramm für customServo.h:



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



## Klassen

- class [CustomServo](#)

*Eine erweiterte Version der [Servo-Klasse](#), die es ermöglicht den Servo mit verschiedenen Geschwindigkeiten zu bewegen.*

## 6.14 customServo.h

[gehe zur Dokumentation dieser Datei](#)

```

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

## 6.15 customServo.ino-Dateireferenz

### 6.15.1 Ausführliche Beschreibung

Umsetzung der [CustomServo](#) Klasse.

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

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

Include-Abhängigkeitsdiagramm für customServo.ino:



## 6.16 customServo.ino

[gehe zur Dokumentation dieser Datei](#)

```

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

```

```

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

```

## 6.17 header.h-Dateireferenz

### 6.17.1 Ausführliche Beschreibung

Definiert variablen-types die überall im Programm benutzt werden.

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

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



### Typdefinitionen

- using [time\\_t](#) = unsigned long  
*Ein Zeit Typ.*
- template<typename ReturnT = void>  
using [func\\_t](#) = ReturnT(\*)()  
*Ein Funktions Typ.*

### 6.17.2 Dokumentation der benutzerdefinierten Typen

**6.17.2.1 func\_t** template<typename ReturnT = void>  
using [func\\_t](#) = ReturnT(\*)()

Ein Funktions Typ.



## Template-Parameter

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

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

### 6.17.2.2 time\_t using time\_t = unsigned long

Ein Zeit Typ.

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

## 6.18 header.h

[gehe zur Dokumentation dieser Datei](#)

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

## 6.19 index.md-Dateireferenz

## 6.20 sketch.ino-Dateireferenz

### 6.20.1 Ausführliche Beschreibung

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

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

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

Include-Abhängigkeitsdiagramm für sketch.ino:



## Funktionen

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

## Klassen

- class `ButtonHandler`  
*Kleine Klasse die Knopfdrücke verarbeitet.*

## Makrodefinitionen

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

## Aufzählungen

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

## Variablen

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

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

## 6.20.2 Makro-Dokumentation

### 6.20.2.1 GEH\_ZURUECK `#define GEH_ZURUECK`

Wert:

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

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

Definiert in Zeile 237 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 155 der Datei `sketch.ino`.

```
00156 {
00157     WHITE,
00158     BLACK,
00159     ORANGE,
00160     NOTHING
00161 };
```

## 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 250 der Datei `sketch.ino`.

```
00251 {
00252     callHandler.deleteCalls();
00253     /*static*/ auto calls = new Callable*[6] {
00254         //static so that the space for the calls is only allocated once
00255         //didn't end up being necessary because at there is only one object in heap at one point in time
00256         (callHandler.deleteCalls())
00257         // new so that it is allocated on the heap
00258         //auto automatically sets the type, in this case Callable*[] (Callable**)
00259         new LcdString("Ball erkannt, vorsicht", &lcd, 1000), //objects get upcasted to Callable*
00260         new LcdDotAnim(name + "er Ball, drehe Links", &lcd, 0),
00261         new FuncCall([]() {
00262             servo.write(angle);
00263             }, &servoIsDone),
00264         new LcdString("Angekommen", &lcd, 1000),
00265         GEH_ZURUECK //2 elemente
00266     };
00267     callHandler.setCalls(calls, 6); //if the number is too large the program crashes
00268 }
```

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



Abbildung 1 vereinfachter Programmlaufplan für die loop() Funktion

Definiert in Zeile 293 der Datei sketch.ino.

```

00294 {
00295   callHandler.update();
00296   lcd.update();
00297   servo.updatePos();
00298   stopButton.update();
00299   if (doFlicker) {
00300     unsigned short b = (millis() / 2) % 513; //helligkeit: 0-512
00301     if (b > 255) {
00302       b = 511 - b; //wenn b größer als 255, wird die helligkeit kleiner, Werte über 255 werden also
"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*[10] {
00351         new LcdString("Oran\2er B\211", &lcd, 1000),
00352         new LcdString("\2\2\20\2\2OR\2R\2A\2N\2G\2\2E\2\2\2!\2\2", &lcd, 0),
00353         new FuncCall([]() {
00354           servo.write(ANGLE_RIGHT_HOLE);
00355         }, &servoIsDone),
00356         new LcdString("Fehler \11rkannt", &lcd, 1000),
00357         new FuncCall([]() {
00358           servo.setSpeed(SERVO_SPEED_FAST);
00359           servo.write(ANGLE_MIN);
00360         }, &servoIsDone),
00361         new FuncCall([]() {
00362           servo.setSpeed(SERVO_SPEED_DEFAULT);
00363         }, &servoIsDone),
00364         new FuncCall([]() {
00365           doFlicker = false;
00366           setLedColor(0, 255, 0);
00367           lcd.backlight();
00368         },),
00369         new LcdString("Fehler beseitigt", &lcd, 2000),
00370         GEH_ZURUECK
00371       };
00372       callHandler.setCalls(callsOrange, 10);
00373       break;
00374     }
00375   }
00376 }

```

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



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

Misst mithilfe des Reflexoptokopplers die Farbe des Balls.

##### Rückgabe

Farbe Die Farbe des Balls

Definiert in Zeile 168 der Datei `sketch.ino`.

```

00169 {
00170     return ORANGE; //inputs hardcoden, für Testzwecke
00171     int hue = random(0, 1000); //inputs simulieren
00172     //int hue=analogRead(A0); //tatsächlich Farbe messen
00173     if (hue <= 100) {
00174         return ORANGE;
00175     }
00176     if (hue < 500) {
00177         return WHITE;
00178     }
00179     if (hue < 830) {
00180         return NOTHING;
00181     }
00182     return BLACK;
00183 }
  
```

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 229 der Datei [sketch.ino](#).

```
00230 {  
00231     return servo.isDone();  
00232 }
```

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 191 der Datei [sketch.ino](#).

```
00192 {
00193   analogWrite(PIN_RED, r);
00194   analogWrite(PIN_GREEN, g);
00195   analogWrite(PIN_BLUE, b);
00196 }
```

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 272 der Datei [sketch.ino](#).

```
00273 {
00274   Serial.begin(9600);
00275   Serial.println("setup");
00276   servo.attach(PIN_SERVO);
00277   lcd.init();
00278   servo.writeDirect(ANGLE_CENTER);
00279   callHandler.setCalls(new Callable*[1] {
00280     new LcdLoadingAnim("Lade", &lcd, LOADING_DURATION),
00281   }, 1);
00282   randomSeed(analogRead(A1));
00283   stopButton = ButtonHandler(PIN_STOPBUTTON, &stopButtonClicked);
00284 }
```

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 201 der Datei sketch.ino.

```

00202 {
00203   static bool isStopped = false; //wird nur einmal initialisiert
00204   isStopped = !isStopped;
00205   if (isStopped) {
00206     Serial.println("stopping servo");
00207     servo.stop();
00208     callHandler.deleteCalls();
00209     /*static*/ auto call = new Callable*[1] {
00210       new LcdDotAnim("gestoppt, warte auf start", &lcd, 1000000000000000) //ja, sollte ich vermutlich
00211       besser implementieren
00212     };
00212     callHandler.setCalls(call, 1);
00213     setLedColor(255, 0, 0);
00214     doFlicker = false;
00215   } else {
00216     callHandler.running = false;
00217   }
00218 }
  
```

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 [28](#) 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 [18](#) 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 [33](#) 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 [23](#) der Datei [sketch.ino](#).

**6.20.5.5 callHandler** `CallHandler callHandler`

Die `CallHandler` Instanz.

Definiert in Zeile 79 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 102 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 13 der Datei `sketch.ino`.

**6.20.5.9 nBlack** `int nBlack = 0`

Die Anzahl schwarzer Bälle, die schon sortiert wurden.

Definiert in Zeile 93 der Datei `sketch.ino`.

**6.20.5.10 nOrange** `int nOrange = 0`

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

Definiert in Zeile 97 der Datei `sketch.ino`.

**6.20.5.11 nWhite** `int nWhite = 0`

Die Anzahl weißer Bälle, die schon sortiert wurden.

Definiert in Zeile [89](#) 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 [58](#) 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 [53](#) 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 [48](#) 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 [38](#) 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 [43](#) der Datei [sketch.ino](#).

**6.20.5.17 servo** `CustomServo servo`

Der Servo, eine `CustomServo` Instanz.

Definiert in Zeile 84 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 63 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 69 der Datei `sketch.ino`.

**6.20.5.20 stopButton** `ButtonHandler stopButton`

der Stop Knopf

Definiert in Zeile 223 der Datei `sketch.ino`.

## 6.21 sketch.ino

gehe zur Dokumentation dieser Datei

```

00001
00005 #include "customServo.h"
00006 #include "animLcd.h"
00007 #include "callHandler.h"
00008 #include "header.h"
00013 const int LOADING_DURATION = 3000;
00018 const int ANGLE_LEFT_HOLE = 180;
00023 const int ANGLE_RIGHT_HOLE = 90;
00028 const int ANGLE_CENTER = 130;
00033 const int ANGLE_MIN = 45;
00038 const int PIN_SERVO = 6;
00043 const int PIN_STOPBUTTON = 13;
00048 const int PIN_RED = 11;
00053 const int PIN_GREEN = 10;
00058 const int PIN_BLUE = 9;
00063 const float SERVO_SPEED_DEFAULT = 0.01f;
00069 const float SERVO_SPEED_FAST = 0.5f;
00074 AnimatableLcd lcd(0x27, 16, 2);
00079 CallHandler callHandler;
00084 CustomServo servo;
00085
00089 int nWhite = 0;
00093 int nBlack = 0;
00097 int nOrange = 0;
00102 bool doFlicker = false;
00103
00108 class ButtonHandler { //handels button clicks
00113     int pin;
00118     bool isPressed = false;
00119 public:
00124     void (*onclick)();
00125     ButtonHandler() {}
00132     ButtonHandler(int pin, void (*onclick)()): pin(pin), onclick(onclick) {
00133         pinMode(pin, INPUT_PULLUP);
00134     }
00139     void update()
00140     {
00141         bool isPressedNew = digitalRead(pin) == HIGH;
00142         if (isPressedNew != isPressed) { //is not being pressed now, but was being pressed
00143             if (isPressed) {
00144                 Serial.println("click");
00145                 onclick();
00146             }
00147         }
00148         isPressed = isPressedNew;
00149     }
00150 };
00155 enum Farbe
00156 {
00157     WHITE,
00158     BLACK,
00159     ORANGE,
00160     NOTHING
00161 };
00162 Farbe measureColor(); //sonst erkennt Arduino Farbe nicht als typ an
//https://forum.arduino.cc/t/syntax-for-a-function-returning-an-enumerated-type/107241
00168 Farbe measureColor()
00169 {
00170     return ORANGE; //inputs hardcoden, für Testzwecke
00171     int hue = random(0, 1000); //inputs simulieren
00172     //int hue=analogRead(A0); //tatsächlich Farbe messen
00173     if (hue <= 100) {
00174         return ORANGE;
00175     }
00176     if (hue < 500) {
00177         return WHITE;
00178     }
00179     if (hue < 830) {
00180         return NOTHING;
00181     }
00182     return BLACK;
00183 }
00191 void setLedColor(unsigned char r, unsigned char g, unsigned char b) //unsigned char: 0-255
00192 {
00193     analogWrite(PIN_RED, r);
00194     analogWrite(PIN_GREEN, g);
00195     analogWrite(PIN_BLUE, b);
00196 }
00201 void stopButtonClicked()
00202 {
00203     static bool isStopped = false; //wird nur einmal initialisiert
00204     isStopped = !isStopped;
00205     if (isStopped) {

```

```

00206     Serial.println("stopping servo");
00207     servo.stop();
00208     callHandler.deleteCalls();
00209     /*static*/ auto call = new Callable*[1] {
00210         new LcdDotAnim("gestoppt, warte auf start", &lcd, 1000000000000000) //ja, sollte ich vermutlich
        besser implementieren
00211     };
00212     callHandler.setCalls(call, 1);
00213     setLedColor(255, 0, 0);
00214     doFlicker = false;
00215     } else {
00216         callHandler.running = false;
00217     }
00218 }
00223 ButtonHandler stopButton;
00229 bool servoIsDone()
00230 {
00231     return servo.isDone();
00232 }
00237 #define GEH_ZURUECK \
00238     new LcdDotAnim("Gehe zur\365ck",&lcd),\
00239     new FuncCall([]() {\
00240         servo.setSpeed(SERVO_SPEED_FAST);\
00241         servo.write(ANGLE_CENTER);\
00242     },&servoIsDone)
00249 template <short angle>
00250 void legBallAb(String name)
00251 {
00252     callHandler.deleteCalls();
00253     /*static*/ auto calls = new Callable*[6] {
00254         //static so that the space for the calls is only allocated once
        (https://cpp4arduino.com/2018/11/06/what-is-heap-fragmentation.html),
00255         //didn't end up being necessary because at there is only one object in heap at one point in time
        (callHandler.deleteCalls())
00256         // new so that it is allocated on the heap
        //auto automatically sets the type, in this case Callable*[] (Callable**)
00257         new LcdString("Ball erkannt, vorsicht", &lcd, 1000), //objects get upcasted to Callable*
00258         new LcdDotAnim(name + "er Ball, drehe Links", &lcd, 0),
00259         new FuncCall([]() {
00260             servo.write(angle);
00261         }, &servoIsDone),
00262         new LcdString("Angekommen", &lcd, 1000),
00263         GEH_ZURUECK //2 elemente
00264     };
00265     callHandler.setCalls(calls, 6); //if the number is too large the program crashes
00266 }
00272 void setup()
00273 {
00274     Serial.begin(9600);
00275     Serial.println("setup");
00276     servo.attach(PIN_SERVO);
00277     lcd.init();
00278     servo.writeDirect(ANGLE_CENTER);
00279     callHandler.setCalls(new Callable*[1] {
00280         new LcdLoadingAnim("Lade", &lcd, LOADING_DURATION),
00281     }, 1);
00282     randomSeed(analogRead(A1));
00283     stopButton = ButtonHandler(PIN_STOPBUTTON, &stopButtonClicked);
00284 }
00285
00286
00293 void loop()
00294 {
00295     callHandler.update();
00296     lcd.update();
00297     servo.updatePos();
00298     stopButton.update();
00299     if (doFlicker) {
00300         unsigned short b = (millis() / 2) % 513; //helligkeit: 0-512
00301         if (b > 255) {
00302             b = 511 - b; //wenn b größer als 255, wird die helligkeit kleiner, Werte über 255 werden also
            "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);
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*[10] {
00351             new LcdString("Oran\2er B\211", &lcd, 1000),
00352             new LcdString("\2\2\2\20\2\20R\2\2R\2\2A\2\2N\2\2G\2\2E\2\2\2\2!\2\2", &lcd, 0),
00353             new FuncCall([]() {
00354                 servo.write(ANGLE_RIGHT_HOLE);
00355             }, &servoIsDone),
00356             new LcdString("Fehler \1rkannt", &lcd, 1000),
00357             new FuncCall([]() {
00358                 servo.setSpeed(SERVO_SPEED_FAST);
00359                 servo.write(ANGLE_MIN);
00360             }, &servoIsDone),
00361             new FuncCall([]() {
00362                 servo.setSpeed(SERVO_SPEED_DEFAULT);
00363             }, &servoIsDone),
00364             new FuncCall([]() {
00365                 doFlicker = false;
00366                 setLedColor(0, 255, 0);
00367                 lcd.backlight();
00368             },
00369             new LcdString("Fehler beseitigt", &lcd, 2000),
00370             GEH_ZURUECK
00371         );
00372         callHandler.setCalls(callsOrange, 10);
00373         break;
00374     }
00375 }
00376 }

```

## Index

- `_isDone`
    - `FuncCall`, [37](#)
  - `~AnimString`
    - `AnimString`, [15](#)
  - `~Callable`
    - `Callable`, [20](#)
  - `~FuncCall`
    - `FuncCall`, [36](#)
  - `~LcdString`
    - `LcdString`, [45](#)
- `ANGLE_CENTER`
  - `sketch.ino`, [72](#)
- `ANGLE_LEFT_HOLE`
  - `sketch.ino`, [72](#)
- `ANGLE_MIN`
  - `sketch.ino`, [72](#)
- `ANGLE_RIGHT_HOLE`
  - `sketch.ino`, [72](#)
- `AnimatableLcd`, [6](#)
  - `animString`, [13](#)
  - `doAnimation`, [13](#)
  - `init`, [8](#)
  - `print`, [8](#), [9](#)
  - `printCentered`, [10](#)
  - `printPretty`, [11](#)
  - `setAnimation`, [12](#)
  - `update`, [12](#)
- `animLcd.h`, [47](#)
  - `LOADING_BAR_OFFSET`, [48](#)
- `animLcd.ino`, [48](#)
  - `loading_empty_c`, [49](#)
  - `loading_full_c`, [49](#)
- `animStart`
  - `AnimString`, [16](#)
- `AnimString`, [14](#)
  - `~AnimString`, [15](#)
  - `animStart`, [16](#)
  - `init`, [15](#)
  - `lastRefresh`, [16](#)
  - `LcdString`, [15](#)
  - `run`, [16](#)
  - `stepDuration`, [17](#)
- `animString`
  - `AnimatableLcd`, [13](#)
- `animString.h`, [51](#)
- `animString.ino`, [53](#)
- `BLACK`
  - `sketch.ino`, [65](#)
- `ButtonHandler`, [17](#)
  - `ButtonHandler`, [18](#)
  - `isPressed`, [19](#)
  - `onclick`, [19](#)
  - `pin`, [19](#)
  - `update`, [18](#)
- `call`
  - `FuncCall`, [37](#)
- `Callable`, [19](#)
  - `~Callable`, [20](#)
  - `isDone`, [20](#)
  - `run`, [20](#)
- `CallHandler`, [21](#)
  - `callPtrs`, [25](#)
  - `callsSet`, [25](#)
  - `currCallPtr`, [25](#)
  - `deleteCalls`, [22](#)
  - `lastCallPtr`, [26](#)
  - `lastCallT`, [26](#)
  - `running`, [26](#)
  - `setCalls`, [23](#)
  - `update`, [24](#)
- `callHandler`
  - `sketch.ino`, [72](#)
- `callHandler.h`, [55](#)
- `callHandler.ino`, [57](#)
- `callPtrs`
  - `CallHandler`, [25](#)
- `callsSet`
  - `CallHandler`, [25](#)
- `callStart`
  - `LcdString`, [46](#)
- `currCallPtr`
  - `CallHandler`, [25](#)
- `CustomServo`, [27](#)
  - `done`, [34](#)
  - `isDone`, [28](#)
  - `setSpeed`, [29](#)
  - `speed`, [34](#)
  - `start`, [30](#)
  - `startAngle`, [34](#)
  - `startMove`, [30](#)
  - `startTime`, [34](#)
  - `stop`, [30](#)
  - `targetAngle`, [34](#)
  - `updatePos`, [31](#)
  - `write`, [31](#), [32](#)
  - `writeDirect`, [33](#)
- `customServo.h`, [58](#)
- `customServo.ino`, [59](#)
- `deleteCalls`
  - `CallHandler`, [22](#)
- `doAnimation`
  - `AnimatableLcd`, [13](#)
- `doFlicker`
  - `sketch.ino`, [73](#)
- `done`
  - `CustomServo`, [34](#)
- `duration`
  - `LcdString`, [46](#)

Farbe  
  [sketch.ino, 64](#)

func\_t  
  [header.h, 61](#)

FuncCall, [35](#)  
  [\\_isDone, 37](#)  
  [~FuncCall, 36](#)  
  [call, 37](#)  
  [FuncCall, 36](#)  
  [isDone, 36](#)  
  [run, 37](#)

GEH\_ZURUECK  
  [sketch.ino, 64](#)

[header.h, 61](#)  
  [func\\_t, 61](#)  
  [time\\_t, 62](#)

[index.md, 62](#)

init  
  [AnimatableLcd, 8](#)  
  [AnimString, 15](#)  
  [LcdDotAnim, 39](#)  
  [LcdLoadingAnim, 42](#)

isDone  
  [Callable, 20](#)  
  [CustomServo, 28](#)  
  [FuncCall, 36](#)  
  [LcdString, 45](#)

isPressed  
  [ButtonHandler, 19](#)

lastCallPtr  
  [CallHandler, 26](#)

lastCallT  
  [CallHandler, 26](#)

lastRefresh  
  [AnimString, 16](#)

lcd  
  [LcdString, 46](#)  
  [sketch.ino, 73](#)

LcdDotAnim, [38](#)  
  [init, 39](#)  
  [LcdDotAnim, 39](#)  
  [update, 39](#)

LcdLoadingAnim, [40](#)  
  [init, 42](#)  
  [update, 42](#)

LcdString, [43](#)  
  [~LcdString, 45](#)  
  [AnimString, 15](#)  
  [callStart, 46](#)  
  [duration, 46](#)  
  [isDone, 45](#)  
  [lcd, 46](#)  
  [LcdString, 44](#)  
  [run, 45](#)  
  [text, 46](#)  
  [update, 45](#)

legBallAb  
  [sketch.ino, 65](#)

LOADING\_BAR\_OFFSET  
  [animLcd.h, 48](#)

LOADING\_DURATION  
  [sketch.ino, 73](#)

loading\_empty\_c  
  [animLcd.ino, 49](#)

loading\_full\_c  
  [animLcd.ino, 49](#)

loop  
  [sketch.ino, 66](#)

measureColor  
  [sketch.ino, 68](#)

nBlack  
  [sketch.ino, 73](#)

nOrange  
  [sketch.ino, 73](#)

NOTHING  
  [sketch.ino, 65](#)

nWhite  
  [sketch.ino, 73](#)

onclick  
  [ButtonHandler, 19](#)

ORANGE  
  [sketch.ino, 65](#)

pin  
  [ButtonHandler, 19](#)

PIN\_BLUE  
  [sketch.ino, 74](#)

PIN\_GREEN  
  [sketch.ino, 74](#)

PIN\_RED  
  [sketch.ino, 74](#)

PIN\_SERVO  
  [sketch.ino, 74](#)

PIN\_STOPBUTTON  
  [sketch.ino, 74](#)

print  
  [AnimatableLcd, 8, 9](#)

printCentered  
  [AnimatableLcd, 10](#)

printPretty  
  [AnimatableLcd, 11](#)

run  
  [AnimString, 16](#)  
  [Callable, 20](#)  
  [FuncCall, 37](#)  
  [LcdString, 45](#)

running  
  [CallHandler, 26](#)

servo  
  [sketch.ino, 74](#)

SERVO\_SPEED\_DEFAULT  
  [sketch.ino, 75](#)

SERVO\_SPEED\_FAST  
  [sketch.ino, 75](#)

servosDone  
  [sketch.ino, 69](#)

setAnimation  
  [AnimatableLcd, 12](#)

setCalls  
  [CallHandler, 23](#)

setLedColor  
  [sketch.ino, 69](#)

setSpeed  
  [CustomServo, 29](#)

setup  
  [sketch.ino, 70](#)

[sketch.ino, 62](#)  
  ANGLE\_CENTER, [72](#)  
  ANGLE\_LEFT\_HOLE, [72](#)  
  ANGLE\_MIN, [72](#)  
  ANGLE\_RIGHT\_HOLE, [72](#)  
  BLACK, [65](#)  
  callHandler, [72](#)  
  doFlicker, [73](#)  
  Farbe, [64](#)  
  GEH\_ZURUECK, [64](#)  
  lcd, [73](#)  
  legBallAb, [65](#)  
  LOADING\_DURATION, [73](#)  
  loop, [66](#)  
  measureColor, [68](#)  
  nBlack, [73](#)  
  nOrange, [73](#)  
  NOTHING, [65](#)  
  nWhite, [73](#)  
  ORANGE, [65](#)  
  PIN\_BLUE, [74](#)  
  PIN\_GREEN, [74](#)  
  PIN\_RED, [74](#)  
  PIN\_SERVO, [74](#)  
  PIN\_STOPBUTTON, [74](#)  
  servo, [74](#)  
  SERVO\_SPEED\_DEFAULT, [75](#)  
  SERVO\_SPEED\_FAST, [75](#)  
  servosDone, [69](#)  
  setLedColor, [69](#)  
  setup, [70](#)  
  stopButton, [75](#)  
  stopButtonClicked, [71](#)  
  WHITE, [65](#)

speed  
  [CustomServo, 34](#)

start  
  [CustomServo, 30](#)

startAngle  
  [CustomServo, 34](#)

startMove  
  [CustomServo, 30](#)

startTime  
  [CustomServo, 34](#)

stepDuration  
  [AnimString, 17](#)

stop  
  [CustomServo, 30](#)

stopButton  
  [sketch.ino, 75](#)

stopButtonClicked  
  [sketch.ino, 71](#)

targetAngle  
  [CustomServo, 34](#)

text  
  [LcdString, 46](#)

time\_t  
  [header.h, 62](#)

update  
  [AnimatableLcd, 12](#)  
  [ButtonHandler, 18](#)  
  [CallHandler, 24](#)  
  [LcdDotAnim, 39](#)  
  [LcdLoadingAnim, 42](#)  
  [LcdString, 45](#)

updatePos  
  [CustomServo, 31](#)

WHITE  
  [sketch.ino, 65](#)

write  
  [CustomServo, 31, 32](#)

writeDirect  
  [CustomServo, 33](#)