Modellbahn

Erzeugt von Doxygen 1.9.0

Kapitel 1

Klassen-Verzeichnis

1.1 Auflistung der Klassen

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

Kapitel 2

Datei-Verzeichnis

2.1 Auflistung der Dateien

Hier folgt die Aufzählung aller dokumentierten Dateien mit einer Kurzbeschreibung:

Kapitel 3

Klassen-Dokumentation

3.1 Button Klassenreferenz

Erstelle einen Button, der für eine bestimmte Zeit nachdem er gedrückt wurde deaktiviert wird. Während der Button deaktiviert ist, ist sein Licht ausgeschaltet und er reagiert nicht wenn er nochmals gedrückt wird.

`#include <Button.h>`

Öffentliche Methoden

- Button (byte inputPin, byte lightPin, int delayTime=60)

  Erstelle eine Instanz der Klasse Button.
- void setCallback (const lambda_callback_t &action)

  Definiere eine Funktion, die ausgeführt wird, sobald der Button gedrückt wurde.
- void handleButton ()

  Überprüfe, ob der Button gedrückt wurde oder ob er wieder aktiviert werden kann. Diese Methode muss in der loop() aufgerufen werden.

3.1.1 Ausführliche Beschreibung

Erstelle einen Button, der für eine bestimmte Zeit nachdem er gedrückt wurde deaktiviert wird. Während der Button deaktiviert ist, ist sein Licht ausgeschaltet und er reagiert nicht wenn er nochmals gedrückt wird.

Alle Instanzen müssen vor oder im setup erzeugt werden. In der loop muss handleButton() ausgeführt werden. Bei der Verwendung von mehreren Buttons wird die verwendung von ButtonManager empfohlen.

Definiert in Zeile 131 der Datei Button.h.

3.1.2 Beschreibung der Konstruktoren und Destruktoren

3.1.2.1 Button()

```
Button::Button (
            byte inputPin,
            byte lightPin,
            int delayTime = 60 )
```

Erstelle eine Instanz der Klasse Button.

Parameter

inputPin	Pin, an den der Button angeschlossen ist
lightPin	Pin, an den das Licht des Buttons angeschlossen ist
delayTime	Zeit in Sekunden, die der Button nachdem er gedrückt wurde deaktiviert ist. Während dieser Zeit wird auch das Licht des Buttons deaktiviert. Der Standardwert ist 60

Rückgabe

Referenz zur Instanz von Button

3.1.3 Dokumentation der Elementfunktionen

3.1.3.1 handleButton()

```
void Button::handleButton ( )
```

Überprüfe, ob der Button gedrückt wurde oder ob er wieder aktiviert werden kann. Diese Methode muss in der loop() aufgerufen werden.

Wurde ein aktiver Button gedrückt, wird die durch setCallback festgelegte Funktion ausgeführt.

3.1.3.2 setCallback()

```
void Button::setCallback (
            const lambda_callback_t & action )  [inline]
```

Definiere eine Funktion, die ausgeführt wird, sobald der Button gedrückt wurde.

Parameter

action	Lampdafunktion die ausgeführt wird, sobald der Button gedrückt wurde

Definiert in Zeile 154 der Datei Button.h.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

- include/Button.h

3.2 ButtonListItem Klassenreferenz

Ein Element einer Linked List in der alle Buttons gespeichert werden. So können beliebig viele Buttons erzeugt werden.

```
#include <ButtonManager.h>
```

Öffentliche Attribute

- Button * **button**
- ButtonListItem * **next**

3.2.1 Ausführliche Beschreibung

Ein Element einer Linked List in der alle Buttons gespeichert werden. So können beliebig viele Buttons erzeugt werden.

Definiert in Zeile 21 der Datei ButtonManager.h.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

- include/ButtonManager.h

3.3 ButtonManager Klassenreferenz

Verwalte einfach mehrere Buttons in einem Programm.

```
#include <ButtonManager.h>
```

Öffentliche, statische Methoden

- static void addButton (Button *newButton)

  Füge einen neuen Button hinzu der verwaltet werden soll.
- static void handleButtons ()

3.3.1 Ausführliche Beschreibung

Verwalte einfach mehrere Buttons in einem Programm.

Es werden keine Instanzen dieser Klasse erstellt

Definiert in Zeile 32 der Datei ButtonManager.h.

3.3.2 Dokumentation der Elementfunktionen

3.3.2.1 addButton()

```
static void ButtonManager::addButton (
            Button * newButton )  [static]
```

Füge einen neuen Button hinzu der verwaltet werden soll.

Parameter

newButton	Zeiger auf einen Button

3.3.2.2 handleButtons()

```
static void ButtonManager::handleButtons ( )  [static]
```

```
@brief Überprüfe, ob ein Button gedrückt wurde oder ob einer wieder aktiviert
werden kann. Diese Methode muss in der loop() aufgerufen werden
```

Wurde ein aktiver Button gedrückt, wird die durch setCallback festgelegte Funktion ausgeführt.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

- include/ButtonManager.h

3.4 Invocative< Resultant, Parametric > Template-Strukturreferenz

Öffentliche Typen

- using **Interfacial** = Resultant(const void *const, Parametric...)

Öffentliche Methoden

- Resultant **operator()** (Parametric... arguments) const

Öffentliche Attribute

- Interfacial * **interface**
- const void * **locality**

3.4.1 Ausführliche Beschreibung

template<typename Resultant, typename... Parametric>
struct Invocative< Resultant, Parametric >

Definiert in Zeile 25 der Datei Button.h.

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

- include/Button.h

3.5 lambda_callback_t Strukturreferenz

Öffentliche Typen

- using **invoke_t** = Invocative< CALLBACK_TYPE_LIST >
- using **remover_t** = void(void *)

Öffentliche Methoden

- **lambda_callback_t** (const lambda_callback_t &copy)
- void **replace** (const lambda_callback_t &with)
- void **cleanup** ()

Öffentliche Attribute

- invoke_t **invoke**
- void * **lambda**
- remover_t * **remove**

3.5.1 Ausführliche Beschreibung

Definiert in Zeile 64 der Datei Button.h.

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

- include/Button.h

3.6 NeoPixel Klassenreferenz

Öffentliche Methoden

- **NeoPixel** (int numPixel, byte pin)
- void **fill** (CRGB color, int start=0, int end=numPixel - 1)
- void **fill** (int r, int g, int b, int start=0, int end=numPixel - 1)
- void **setColor** (int pixel, CRGB color)
- void **setColor** (int pixel, int r, int g, int b)
- void **off** ()

3.6.1 Ausführliche Beschreibung

Definiert in Zeile 7 der Datei NeoPixel.h.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

- include/NeoPixel.h

Kapitel 4

Datei-Dokumentation

4.1 include/Button.h-Dateireferenz

Erstelle einen Button, der für eine bestimmte Zeit nachdem er gedrückt wurde deaktiviert wird.

```
#include "Arduino.h"
#include <Wire.h>
```

Klassen

- struct Invocative< Resultant, Parametric >
- struct lambda_callback_t
- class Button

  Erstelle einen Button, der für eine bestimmte Zeit nachdem er gedrückt wurde deaktiviert wird. Während der Button deaktiviert ist, ist sein Licht ausgeschaltet und er reagiert nicht wenn er nochmals gedrückt wird.

Makrodefinitionen

- #define **CALLBACK_TYPE_LIST** int

Funktionen

- template<typename Procedural , typename Resultant , typename... Parametric>
  Resultant **InvokeProcedure** (const void *locality, Parametric... arguments)

4.1.1 Ausführliche Beschreibung

Erstelle einen Button, der für eine bestimmte Zeit nachdem er gedrückt wurde deaktiviert wird.

Autor

Dennis Moschina

Datum

28 Dec 2020

Copyright

2020 Dennis Moschina

4.2 include/ButtonManager.h-Dateireferenz

Verwalte einfach mehrere Buttons in einem Programm.

```
#include <Arduino.h>
#include "Button.h"
```

Klassen

- class ButtonListItem

  Ein Element einer Linked List in der alle Buttons gespeichert werden. So können beliebig viele Buttons erzeugt werden.
- class ButtonManager

  Verwalte einfach mehrere Buttons in einem Programm.

4.2.1 Ausführliche Beschreibung

Verwalte einfach mehrere Buttons in einem Programm.

Autor

Dennis Moschina

Datum

28 Dec 2020

Copyright

2020 Dennis Moschina

4.3 include/Modellbahn.h-Dateireferenz

Eine einfache Sammlung der Libraries. Es kann Modellbahn.h eingebunden werden und es sind alle Libraries verwendbar.

```
#include "ButtonManager.h"
#include "Button.h"
#include "NeoPixel.h"
```

4.3.1 Ausführliche Beschreibung

Eine einfache Sammlung der Libraries. Es kann Modellbahn.h eingebunden werden und es sind alle Libraries verwendbar.

Autor

Dennis Moschina

Datum

28 Dec 2020

Copyright

2020 Dennis Moschina