LibSpindle-C-Library

1.0

Generated on Sun Feb 16 2025 00:30:29 for LibSpindle-C-Library by Doxygen 1.13.2

Sun Feb 16 2025 00:30:29

Chapter 1

FreeRTOS Spindle Library

1.1 Introduction

The following code documentation is a set of tipps and diagramms as well as function and structure descriptions which help when using the library. It is used to increase the implementation speed. Some examples are attached in this documentation as well

1.2 static compile flags of the library

There are no additional static compile flags or DEFINES which the user can configure

1.3 library instantiation

The library is implemented as a singleton pattern. This means that only one instance can be created at runtime. Calling the SPINDLE_CreateInstance multiple times will succeed but always the pointer of the first instance will be returned.

1.4 Examples

The following example shows how to create a instance of the spindle controller library.

```
// set parameters for the physical system
SpindlePhysicalParams_t s;
s.maxRPM                =  9000.0f;
s.minRPM                = -9000.0f;
s.absMinRPM             =  1600.0f;
s.setDirection          = SPINDLE_SetDirection;
s.setDutyCycle          = SPINDLE_SetDutyCycle;
s.enaPWM                = SPINDLE_EnaPWM;
s.context               = NULL;
SPINDLE_CreateInstance( 4*configMINIMAL_STACK_SIZE, configMAX_PRIORITIES - 3, c, &s);

...
```

The following example shows the usage of the spindle library via console. With the spindle command the user can start or stop the spindle and it can also change the spindle speed with the start command The command returns

OK or FAIL in the given cases of failure or success. There is also a status command. It returns the state of the spindle (turning) and the RPM which has been set. It also terminates with "OK" or "FAIL"

```
// starts the spindle
$> spindle start <RPM>

// sets another spped of the spindle
$> spindle start <RPM>

// stops the spindle
$> spindle stop

// get the status the spindle
$> spindle status
```

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 4

Data Structure Documentation

4.1 CtrlCommand Struct Reference

Collaboration diagram for CtrlCommand:

Data Fields

- struct {
    int **requestID**
    CtrlCommandType_t **type**
  } **head**

- struct {
    union {
      struct {
        float **speed**
      } **asStart**
    } **args**
    SemaphoreHandle_t **syncEvent**
  } **request**

- StepCommandResponse_t ∗ **response**

The documentation for this struct was generated from the following file:

- C:/HomeGit/STM32/libs/LibSpindle/src/Spindle.c

4.2 SpindleHandle Struct Reference

Collaboration diagram for SpindleHandle:

Data Fields

- int **nextRequestID**
- ConsoleHandle_t **consoleH**
- TaskHandle_t **tHandle**
- QueueHandle_t **cmdQueue**
- int **cancel**
- SpindlePhysicalParams_t **physical**
- float **currentSpeed**
- struct {
    SemaphoreHandle_t **lockGuard**
  } **syncEventPool**

The documentation for this struct was generated from the following file:

- C:/HomeGit/STM32/libs/LibSpindle/src/Spindle.c

4.3 SpindlePhysicalParams Struct Reference

```
#include <Spindle.h>
```

Data Fields

- void(∗ setDirection)(SpindleHandle_t h, void ∗context, int backward)
- void(∗ setDutyCycle)(SpindleHandle_t h, void ∗context, float dutyCycle)
- void(∗ enaPWM)(SpindleHandle_t h, void ∗context, int ena)
- float maxRPM
- float absMinRPM
- float minRPM
- void ∗ context

4.3.1 Detailed Description

The SpindlePhysicalParams_t structure is represents the abstraction functions and members as a container. The objetcs are passed as structure pointer when calling SPINDLE_CreateInstance. It contains function pointers to platform specific functions which enable the spindle or regulate the speed and direction of it. There are also limit values for the maximum, minimum and absolute minimum RPM and an additional context pointer which is passed the the abstraction functions. The user must provide a configured structure in its design when calling SPINDLE_↩
CreateInstance.

4.3.2 Field Documentation

4.3.2.1 absMinRPM

```
float SpindlePhysicalParams::absMinRPM
```

specifies the minimum possible absolute RPM value which is usable with the spindle a requested RPM value lower than this value is limited and therefore its not possible to let the spindle run slower. This value is used for clock wise and counter clock wise spindle rotation in the same way.

4.3.2.2 context

```
void* SpindlePhysicalParams::context
```

optional context pointer for platform abstraction. can be null.

4.3.2.3 enaPWM

```
void(* SpindlePhysicalParams::enaPWM) (SpindleHandle_t h, void *context, int ena)
```

This function pointer is used to enable or disable the PWM output and to set the enable outputs od the half bridges of the spindle

The pointer must not be null

4.3.2.4 maxRPM

```
float SpindlePhysicalParams::maxRPM
```

specifies the maximum positive RPM value which represents a duty cycle of 1.0

4.3.2.5 minRPM

```
float SpindlePhysicalParams::minRPM
```

specifies the minimum negative RPM value. Mostly this value is symmetrical to the max value which means it can be the same value

4.3.2.6 setDirection

```
void(* SpindlePhysicalParams::setDirection) (SpindleHandle_t h, void *context, int backward)
```

This function pointer is used to change the direction of the spindle. It is always called before setDutyCycle and is never called without any other combination of setDutyCycle or enaPWM. This means that it can simply be used to set a variable in the user design instead of handling all required PWM steps to change the direction of the spindle.

The pointer must not be null

4.3.2.7 setDutyCycle

```
void(* SpindlePhysicalParams::setDutyCycle) (SpindleHandle_t h, void *context, float dutyCycle)
```

This function pointer is used to change the duty cycle of the spindle. This function is necessary for changing the PWM generation itself

The pointer must not be null

The documentation for this struct was generated from the following file:

- C:/HomeGit/STM32/libs/LibSpindle/inc/Spindle.h

4.4 StepCommandResponse Struct Reference

Data Fields

- int **code**
- int **requestID**
- union {
    struct {
      float **speed**
      int **running**
    } **asStatus**
  } **args**

The documentation for this struct was generated from the following file:

- C:/HomeGit/STM32/libs/LibSpindle/src/Spindle.c

4.5 stepSyncEventElement Struct Reference

Data Fields

- struct {
    int **allocated**
    SemaphoreHandle_t **event**
  } **content**

The documentation for this struct was generated from the following file:

- C:/HomeGit/STM32/libs/LibSpindle/src/Spindle.c

Chapter 5

File Documentation

5.1 C:/HomeGit/STM32/libs/LibSpindle/inc/Spindle.h File Reference

`#include "Console.h"`
Include dependency graph for Spindle.h: This graph shows which files directly or indirectly include this file:

Data Structures

- struct SpindlePhysicalParams

Typedefs

- typedef struct SpindleHandle ∗ SpindleHandle_t
- typedef struct SpindlePhysicalParams SpindlePhysicalParams_t

Functions

- SpindleHandle_t SPINDLE_CreateInstance (unsigned int uxStackDepth, int xPrio, ConsoleHandle_t cH, SpindlePhysicalParams_t ∗p)

5.1.1 Typedef Documentation

5.1.1.1 SpindleHandle_t

```
typedef struct SpindleHandle* SpindleHandle_t
```

The SpindleHandle_t handle is an instance pointer of the spindle library which is generated whenever the SPINDLE_CreateInstance function returns with success.

5.1.1.2 SpindlePhysicalParams_t

```
typedef struct SpindlePhysicalParams SpindlePhysicalParams_t
```

The SpindlePhysicalParams_t structure is represents the abstraction functions and members as a container. The objetcs are passed as structure pointer when calling SPINDLE_CreateInstance. It contains function pointers to platform specific functions which enable the spindle or regulate the speed and direction of it. There are also limit values for the maximum, minimum and absolute minimum RPM and an additional context pointer which is passed the the abstraction functions. The user must provide a configured structure in its design when calling SPINDLE_↩ CreateInstance.

5.1.2 Function Documentation

5.1.2.1 SPINDLE_CreateInstance()

```
SpindleHandle_t SPINDLE_CreateInstance (
            unsigned int uxStackDepth,
            int xPrio,
            ConsoleHandle_t cH,
            SpindlePhysicalParams_t * p)
```

The SPINDLE_CreateInstance function is used to create the spindle controller. There is a singleton pattern implemented for the controller so there is only one instance possible for the design. In case it is called multiple times, it returns the instance pointer of the first created spindle controller

The return value of SPINDLE_CreateInstance is a null pointer in case an error occured or a pointer of type SpindleHandle_t.

param uxStackDepth is the stack depth of the console processor thread in words param xPrio is the spindle controller priority. param cH of type ConsoleHandle_t is the handle pointer of a console processor instance which is required to register the console command for the spindle controller param p of type SpindlePhysicalParams_t∗ is a pointer to the platform abstraction functions

ATTENTION: This function is not locked or guarded and therefore its never thread safe when calling from two different threads at the same or nearly the same time while no instance has been created before! One pointer and its ressources could be lost which leads to a memory leak and/or to console command registration issues at runtime when using the spindle command!

5.2 Spindle.h

Go to the documentation of this file.

```
/*
 * Controller.h
 *
 *  Created on: Dec 9, 2024
 *      Author: Thorsten
 */


#ifndef INC_SPINDLE_CONTROLLER_H_
#define INC_SPINDLE_CONTROLLER_H_

#include "Console.h"

typedef struct SpindleHandle* SpindleHandle_t;

typedef struct SpindlePhysicalParams
{
    void (*setDirection)(SpindleHandle_t h, void* context, int backward);

    void (*setDutyCycle)(SpindleHandle_t h, void* context, float dutyCycle);

    void (*enaPWM)      (SpindleHandle_t h, void* context, int ena);


    float        maxRPM;

    float        absMinRPM;

    float        minRPM;

    void*        context;

} SpindlePhysicalParams_t;

SpindleHandle_t SPINDLE_CreateInstance( unsigned int uxStackDepth, int xPrio, ConsoleHandle_t cH,
SpindlePhysicalParams_t* p );




#endif /* INC_STEPPER_CONTROLLER_H_ */
```

5.3 C:/HomeGit/STM32/libs/LibSpindle/src/Spindle.c File Reference

```
#include "Spindle.h"
#include "FreeRTOS.h"
#include "task.h"
#include "semphr.h"
#include "queue.h"
#include "timers.h"
#include <malloc.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/queue.h>
#include <math.h>
```

Include dependency graph for Spindle.c:

Index