LibL6474-C-Library

1.0

Generated by Doxygen 1.8.17

Chapter 1

Stepper Library Lib6474

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

LIBL6474_STEP_ASYNC: This DEFINE is used to switch from blocking synchronous mode to asynchronous non-blocking step mode. This changes the API behavior

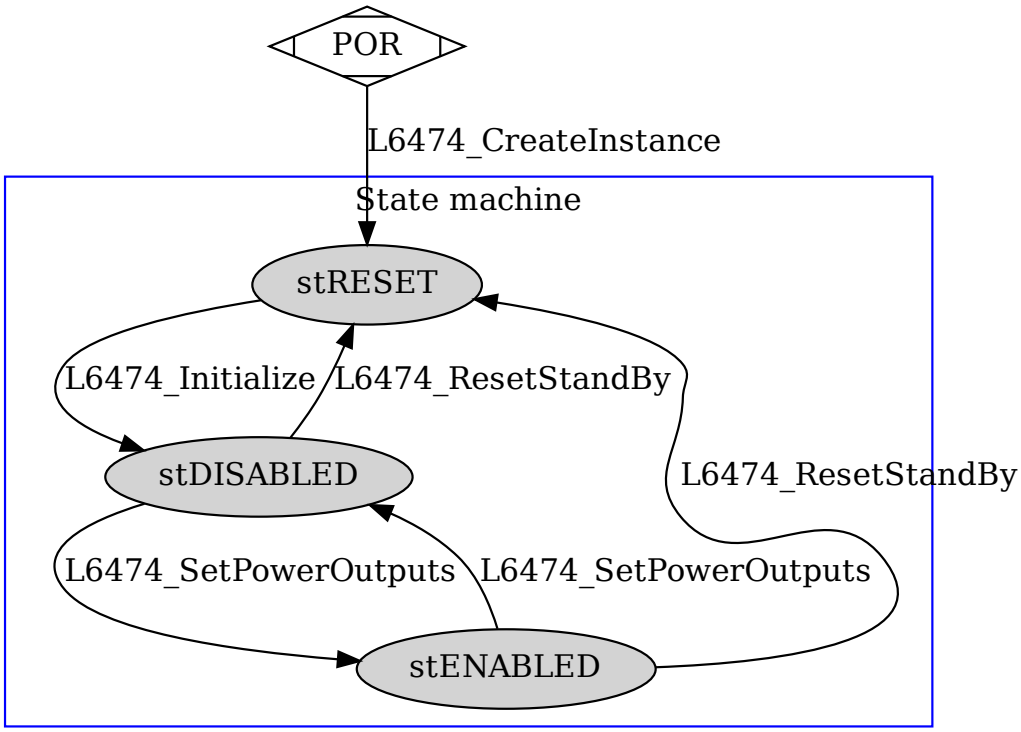
LIBL6474_HAS_LOCKING: This DEFINE is used to enable the lock guard and thread synchronization guard abstraction, which requires additional abstraction functions

LIBL6474_DISABLE_OCD: Should only be used for debugging purpopses and not for productive code! This DE↩ FINE is used to disable the overcurrent detection feature in the library and the stepper driver

LIBL6474_HAS_FLAG: enables the support of the flag pin This DEFINE is used to enable the FLAG pin support, which requires additional abstraction functions

1.3 State diagram

The following state diagram shows the internal state machine handling which follows or represents the state of the stepper driver chip. The fault conditions are not depicted in the diagram but in all regular cases the error hendling is done by resetting and reinitialization of the stepper driver by the library.



1.4 Examples

The following example shows a part of the abstraction functions which are required for the usage of the Stepper library. It also shows how to create a instance of the stepper library.

```
static void* StepLibraryMalloc( unsigned int size )
{
     return malloc(size);
}
static void StepLibraryFree( const void* const ptr )
{
     free((void*)ptr);
}
static int StepDriverSpiTransfer( void* pIO, char* pRX, const char* pTX, unsigned int length )
{
  // byte based access, so keep in mind that only single byte transfers are performed!
  for ( unsigned int i = 0; i < length; i++ )
  {
    ...
  }
  return 0;
}
...
// pass all function pointers required by the stepper library
// to a separate platform abstraction structure
L6474x_Platform_t p;
p.malloc     = StepLibraryMalloc;
p.free       = StepLibraryFree;
p.transfer   = StepDriverSpiTransfer;
p.reset      = StepDriverReset;
p.sleep      = StepLibraryDelay;
p.stepAsync  = StepTimerAsync;
p.cancelStep = StepTimerCancelAsync;
// now create the handle
L6474_Handle_t h = L6474_CreateInstance(&p, null, null, null);
```

The following example shows a really simple instantiation and usage of the library with a simple default and straight forward configuration and no calculation of any step widths or resolutions

```
...
int result = 0;
...
// reset all and take all initialization steps
result |= L6474_ResetStandBy(h);
result |= L6474_Initialize(h, &param);
result |= L6474_SetPowerOutputs(h, 1);
...
// in case we have no error, we can enable the drivers
// and then we step a bit
if ( result == 0 )
{
    result |= L6474_StepIncremental(h, 1000 );
}
else
{
    // error handling
    ...
}
...
```

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 L6474_BaseParameter_t Struct Reference

`#include <LibL6474.h>`

Data Fields

- L6474x_StepMode_t stepMode
- L6474x_OCD_TH_t OcdTh
- char TimeOnMin
- char TimeOffMin
- char TorqueVal
- char TFast

4.1.1 Detailed Description

The L6474_BaseParameter_t structure is used to set default initialization values for the library whenever a reset has been called or the library is used the first time. It can be filled with the global defaults by calling L6474_SetBase↩ Parameter and it then can be adapted or directly passed into L6474_Initialize which always needs to be called after resetting the stepper driver or the library

4.1.2 Field Documentation

4.1.2.1 OcdTh

`L6474x_OCD_TH_t L6474_BaseParameter_t::OcdTh`

OcdTh describes the overcurrent detection threshold, see L6474x_OCD_TH_t

4.1.2.2 stepMode

```
L6474x_StepMode_t L6474_BaseParameter_t::stepMode
```

stepMode describes the stepping operation mode, see L6474x_StepMode_t

4.1.2.3 TFast

```
char L6474_BaseParameter_t::TFast
```

L6474_PROP_TFAST is used to change switching times

4.1.2.4 TimeOffMin

```
char L6474_BaseParameter_t::TimeOffMin
```

TimeOffMin is used to change the minimum time for the OFF-part of the current control loop

4.1.2.5 TimeOnMin

```
char L6474_BaseParameter_t::TimeOnMin
```

TimeOnMin is used to change the minimum time for the ON-part of the current control loop

4.1.2.6 TorqueVal

```
char L6474_BaseParameter_t::TorqueVal
```

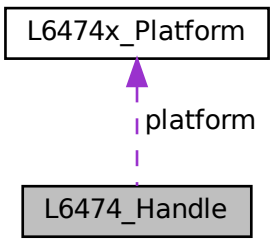
TorqueVal is used to change the current setpoint of the controller which follows directly the torque

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

- /mnt/c/HomeGit/STM32/libs/LibL6474/inc/LibL6474.h

4.2 L6474_Handle Struct Reference

Collaboration diagram for L6474_Handle:



Data Fields

- L6474x_State_t **state**
- int **pending**
- void * **pIO**
- void * **pGPO**
- void * **pPWM**
- L6474x_Platform_t * **platform**

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

- /mnt/c/HomeGit/STM32/libs/LibL6474/src/LibL6474x.c

4.3 L6474_Status Struct Reference

`#include <LibL6474.h>`

Data Fields

- unsigned char HIGHZ
- unsigned char DIR
- unsigned char NOTPERF_CMD
- unsigned char WRONG_CMD
- unsigned char UVLO
- unsigned char TH_WARN
- unsigned char TH_SD
- unsigned char OCD
- unsigned char ONGOING

4.3.1 Detailed Description

The L6474_Status_t enum desribes the status register content of the stepper driver chip. It is reinterpreted so it can be naturally used in the code to get some of the error codes or state bits of the driver.

4.3.2 Field Documentation

4.3.2.1 DIR

`unsigned char L6474_Status::DIR`

direction is clock wise or counter clock wise

4.3.2.2 HIGHZ

```
unsigned char L6474_Status::HIGHZ
```

high impedance state active

4.3.2.3 NOTPERF_CMD

```
unsigned char L6474_Status::NOTPERF_CMD
```

a command has not been performed

4.3.2.4 OCD

```
unsigned char L6474_Status::OCD
```

over current detection (short circuit)

4.3.2.5 ONGOING

```
unsigned char L6474_Status::ONGOING
```

movement is ingoing (pending)

4.3.2.6 TH_SD

```
unsigned char L6474_Status::TH_SD
```

thermal shutdown

4.3.2.7 TH_WARN

```
unsigned char L6474_Status::TH_WARN
```

threshold warning

4.3.2.8 UVLO

```
unsigned char L6474_Status::UVLO
```

under-voltage lock out

4.3.2.9 WRONG_CMD

```
unsigned char L6474_Status::WRONG_CMD
```

a command was invalid

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

- /mnt/c/HomeGit/STM32/libs/LibL6474/inc/LibL6474.h

4.4 L6474x_ParameterDescriptor Struct Reference

Data Fields

- unsigned char **command**
- unsigned char **defined**
- unsigned char **length**
- unsigned int **mask**
- char * **name**
- L6474x_AccessFlags_t **flags**

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

- /mnt/c/HomeGit/STM32/libs/LibL6474/src/LibL6474x.c

4.5 L6474x_Platform Struct Reference

```
#include <LibL6474.h>
```

Data Fields

- void *(* malloc)(unsigned int size)
- void(* free)(const void *const pMem)
- int(* transfer)(void *pIO, char *pRX, const char *pTX, unsigned int length)
- void(* reset)(void *pGPO, const int ena)
- void(* sleep)(unsigned int ms)
- int(* stepAsync)(void *pPWM, int dir, unsigned int numPulses, void(*doneClb)(L6474_Handle_t), L6474_Handle_t)
- int(* cancelStep)(void *pPWM)

4.5.1 Detailed Description

The L6474x_Platform_t structure is used to encapsulate platform specific parameters and to provide environment specific functions in a generalized format. Theses functions are malloc and free or some PWM and GPIO abstractions The lock gurads and thread safety mechanisms are also abstracted by this structure

4.5.2 Field Documentation

4.5.2.1 cancelStep

```
int(* L6474x_Platform::cancelStep) (void *pPWM)
```

the cancelStep function is a synchronous blocking function which cancels the asynchronous prevously started step operation in case it has not been finished

4.5.2.2 free

```
void(* L6474x_Platform::free) (const void *const pMem)
```

classic free function which releases memory previously allocated by malloc

4.5.2.3 malloc

```
void*(* L6474x_Platform::malloc) (unsigned int size)
```

classic malloc pointer which returns null or a memory void* pointer with the given size by the first argument

4.5.2.4 reset

```
void(* L6474x_Platform::reset) (void *pGPO, const int ena)
```

the reset function is used to provide gpio access to the reset of the stepper driver chip

4.5.2.5 sleep

```
void(* L6474x_Platform::sleep) (unsigned int ms)
```

the sleep function implements a wait time with the argument in milliseconds

4.5.2.6 stepAsync

```
int(* L6474x_Platform::stepAsync) (void *pPWM, int dir, unsigned int numPulses, void(*done↩
Clb)(L6474_Handle_t), L6474_Handle_t)
```

the stepAsync function is an asynchronous non-blocking function which e.g. enables a timer which then generates the required amount of pulses given by the numPulses argument. The doneCLB is a callback provided by the library which is called when the asynchronous operation has been done

4.5.2.7 transfer

```
int(* L6474x_Platform::transfer) (void *pIO, char *pRX, const char *pTX, unsigned int length)
```

the tranfer function is used to provide bus access to the stepper driver chip

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

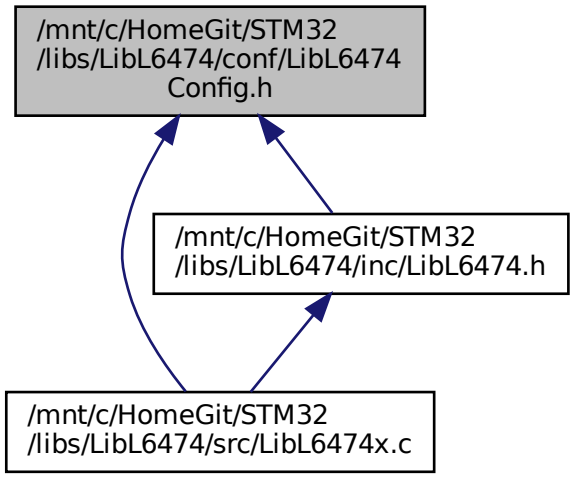
- /mnt/c/HomeGit/STM32/libs/LibL6474/inc/LibL6474.h

Chapter 5

File Documentation

5.1 /mnt/c/HomeGit/STM32/libs/LibL6474/conf/LibL6474Config.h File Reference

This graph shows which files directly or indirectly include this file:



Macros

- #define INC_LIBL6474_CONFIG_H_ INC_LIBL6474_CONFIG_H_
- #define LIBL6474_STEP_ASYNC 1
- #define LIBL6474_HAS_LOCKING 0
- #define LIBL6474_DISABLE_OCD 0
- #define LIBL6474_HAS_FLAG 0

5.1.1 Macro Definition Documentation

5.1.1.1 INC_LIBL6474_CONFIG_H_

```
#define INC_LIBL6474_CONFIG_H_ INC_LIBL6474_CONFIG_H_
```

ATTENTION, this header is only a template, the user must provide its own implementation in the user project

5.1.1.2 LIBL6474_DISABLE_OCD

```
#define LIBL6474_DISABLE_OCD 0
```

This DEFINE is used to disable the overcurrent detection feature in the library and the stepper driver

5.1.1.3 LIBL6474_HAS_FLAG

```
#define LIBL6474_HAS_FLAG 0
```

This DEFINE is used to enable the FLAG pin support, which requires additional abstraction functions

5.1.1.4 LIBL6474_HAS_LOCKING

```
#define LIBL6474_HAS_LOCKING 0
```

This DEFINE is used to enable the lock guard and thread synchronization guard abstraction, which requires additional abstraction functions

5.1.1.5 LIBL6474_STEP_ASYNC

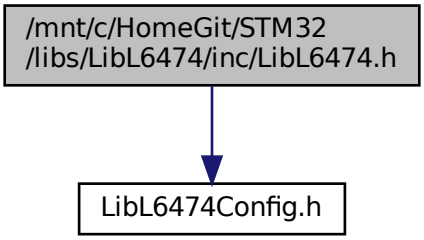
```
#define LIBL6474_STEP_ASYNC 1
```

This DEFINE is used to switch from blocking synchronous mode to asynchronous non-blocking step mode. This changes the API behavior

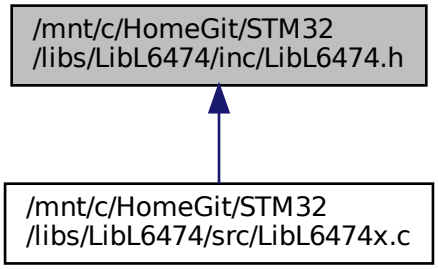
5.2 /mnt/c/HomeGit/STM32/libs/LibL6474/inc/LibL6474.h File Reference

```
#include "LibL6474Config.h"
```

Include dependency graph for LibL6474.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct L6474_Status
- struct L6474x_Platform
- struct L6474_BaseParameter_t

Macros

- #define **INC_LIBL6474_H_** INC_LIBL6474_H_

Typedefs

- typedef enum L6474x_State L6474x_State_t
- typedef enum L6474x_StepMode L6474x_StepMode_t
- typedef enum L6474x_Direction L6474x_Direction_t
- typedef enum L6474x_ErrorCode L6474x_ErrorCode_t
- typedef struct L6474_Status L6474_Status_t
- typedef enum L6474x_OCD_TH L6474x_OCD_TH_t
- typedef struct L6474_Handle * L6474_Handle_t
- typedef struct L6474x_Platform L6474x_Platform_t

Enumerations

- enum L6474x_State { stRESET = 0x00, stDISABLED = 0x01, stENABLED = 0x02, stINVALID = 0x04 }
- enum L6474x_StepMode {
  smFULL = 0x00, smHALF = 0x01, smMICRO4 = 0x02, smMICRO8 = 0x03,
  smMICRO16 = 0x04 }
- enum L6474x_Direction { dirFOREWARD = 0x00, dirBACKWARD = 0x01 }
- enum L6474x_ErrorCode {
  errcNONE = 0, errcINV_ARG = -1, errcNULL_ARG = -2, errcINV_STATE = -3,
  errcINTERNAL = -4, errcLOCKING = -5, errcDEVICE_STATE = -6, errcPENDING = -7 }
- enum L6474x_OCD_TH {
  ocdth375mA = 0x00, **ocdth750mA** = 0x01, **ocdth1125mA** = 0x02, **ocdth1500mA** = 0x03,
  ocdth1875mA = 0x04, **ocdth2250mA** = 0x05, **ocdth2625mA** = 0x06, **ocdth3000mA** = 0x07,
  ocdth3375mA = 0x08, **ocdth3750mA** = 0x09, **ocdth4125mA** = 0x0A, **ocdth4500mA** = 0x0B,
  ocdth4875mA = 0x0C, **ocdth5250mA** = 0x0D, **ocdth5625mA** = 0x0E, **ocdth6000mA** = 0x0F }
- enum L6474_Property_t {
  L6474_PROP_TORQUE = 0x09, L6474_PROP_TON = 0x0F, L6474_PROP_TOFF = 0x10, L6474_PROP_ADC_OUT = 0x12,
  L6474_PROP_OCDTH = 0x13, L6474_PROP_TFAST = 0x0E }

Functions

- L6474_Handle_t L6474_CreateInstance (L6474x_Platform_t *p, void *pIO, void *pGPO, void *pPWM)
- int L6474_ResetStandBy (L6474_Handle_t h)
- int L6474_SetBaseParameter (L6474_BaseParameter_t *p)
- int L6474_Initialize (L6474_Handle_t h, L6474_BaseParameter_t *p)
- int L6474_SetStepMode (L6474_Handle_t h, L6474x_StepMode_t mode)
- int L6474_GetStepMode (L6474_Handle_t h, L6474x_StepMode_t *mode)
- int L6474_SetPowerOutputs (L6474_Handle_t h, int ena)
- int L6474_GetStatus (L6474_Handle_t h, L6474_Status_t *state)
- int L6474_StepIncremental (L6474_Handle_t h, int steps)
- int L6474_StopMovement (L6474_Handle_t h)
- int L6474_IsMoving (L6474_Handle_t h, int *moving)
- int L6474_SetProperty (L6474_Handle_t h, L6474_Property_t prop, int value)
- int L6474_GetProperty (L6474_Handle_t h, L6474_Property_t prop, int *value)
- int L6474_GetAbsolutePosition (L6474_Handle_t h, int *position)
- int L6474_SetAbsolutePosition (L6474_Handle_t h, int position)
- int L6474_GetElectricalPosition (L6474_Handle_t h, int *position)
- int L6474_SetElectricalPosition (L6474_Handle_t h, int position)
- int L6474_GetPositionMark (L6474_Handle_t h, int *position)
- int L6474_SetPositionMark (L6474_Handle_t h, int position)
- int L6474_GetAlarmEnables (L6474_Handle_t h, int *bits)
- int L6474_SetAlarmEnables (L6474_Handle_t h, int bits)

5.2.1 Typedef Documentation

5.2.1.1 L6474_Handle_t

```
typedef struct L6474_Handle* L6474_Handle_t
```

The L6474_Handle_t handle is an instance pointer of the driver library which is generated whenever the L6474_↩ CreateInstance function returns with success. Therefore at least a L6474x_Platform_t platform description pointer is required

5.2.1.2 L6474_Status_t

```
typedef struct L6474_Status L6474_Status_t
```

The L6474_Status_t enum desribes the status register content of the stepper driver chip. It is reinterpreted so it can be naturally used in the code to get some of the error codes or state bits of the driver.

5.2.1.3 L6474x_Direction_t

```
typedef enum L6474x_Direction L6474x_Direction_t
```

The L6474x_Direction_t enum desribes the direction of the driver and therefore the rotation of the stepper axis.

5.2.1.4 L6474x_ErrorCode_t

```
typedef enum L6474x_ErrorCode L6474x_ErrorCode_t
```

The L6474x_ErrorCode_t enum desribes the possible error codes when using the API commands. Those error codes are mostly returned as return value by the API function which has been called previously

5.2.1.5 L6474x_OCD_TH_t

```
typedef enum L6474x_OCD_TH L6474x_OCD_TH_t
```

The L6474x_OCD_TH_t enum desribes the possible overcurrent threshold values which can be set in the library and which are then taken into account by the stepper driver

5.2.1.6 L6474x_Platform_t

```
typedef struct L6474x_Platform L6474x_Platform_t
```

The L6474x_Platform_t structure is used to encapsulate platform specific parameters and to provide environment specific functions in a generalized format. Theses functions are malloc and free or some PWM and GPIO abstractions The lock gurads and thread safety mechanisms are also abstracted by this structure

5.2.1.7 L6474x_State_t

```
typedef enum L6474x_State L6474x_State_t
```

The LibL6474Config.h must be provided by the user project. The header in conf folder is only a template!

The L6474x_State_t enum is used to describe the current state of the driver library which helps the user to decide the next steps to be taken or to get notified about an unexpected state which is caused by an error

5.2.1.8 L6474x_StepMode_t

```
typedef enum L6474x_StepMode L6474x_StepMode_t
```

The L6474x_StepMode_t enum is used to describe the current step mode (resolution) of the driver library and the stepper motor. This changes the internal calculations for speed and position and requires a new reference run in case it will be changed at runtime.

5.2.2 Enumeration Type Documentation

5.2.2.1 L6474_Property_t

```
enum L6474_Property_t
```

The L6474_Property_t enum is a address representation for externally changable properties by the library via the API commands. In case the torque shall be changed, the user can call L6474_SetProperty with the L6474_PRO↩ P_TORQUE entry. Some of the properties can not be changed while the driver outputs are enabled and the library is in stENABLED state

Enumerator

L6474_PROP_TORQUE	L6474_PROP_TORQUE is the current setpoint value of each phase of the stepper which follows the torque directly
L6474_PROP_TON	L6474_PROP_TON is the minimum time for the ON-part of the current control loop
L6474_PROP_TOFF	L6474_PROP_TOFF is the minimum time for the OFF-part of the current control loop
L6474_PROP_ADC_OUT	L6474_PROP_ADC_OUT controls the ADC output behavior
L6474_PROP_OCDTH	L6474_PROP_OCDTH sets the overcurrent threshold detection value
L6474_PROP_TFAST	L6474_PROP_TFAST is used to change switching times

5.2.2.2 L6474x_Direction

```
enum L6474x_Direction
```

The L6474x_Direction_t enum desribes the direction of the driver and therefore the rotation of the stepper axis.

Enumerator

dirFOREWARD	dirFOREWARD is interpreted as clock wise rotation
dirBACKWARD	dirFOREWARD is interpreted as counter clock wise rotation

5.2.2.3 L6474x_ErrorCode

`enum L6474x_ErrorCode`

The L6474x_ErrorCode_t enum desribes the possible error codes when using the API commands. Those error codes are mostly returned as return value by the API function which has been called previously

Enumerator

errcNONE	errcNONE means no error and the operation was successful
errcINV_ARG	errcINV_ARG means that at least one argument was invalid or out of range
errcNULL_ARG	errcNULL_ARG means that at least one argument was a null pointer where it was not expected
errcINV_STATE	errcINV_STATE means that the stepper driver or the library is not in the state of type L6474x_State_t and therefore the operation can not be executed
errcINTERNAL	errcINTERNAL means an error which happens when calling abstraction functions which then return an error code
errcLOCKING	errcLOCKING in case the library is used multi-threaded, it might happen, that a function can not be locked because another caller is still using the library (lock guard principle). In mutex case this should not be returned by the API
errcDEVICE_STATE	errcDEVICE_STATE means that the state of the driver chip permits the operation
errcPENDING	errcPENDING means that another operation is still pending and therefore the operation can not be executed while there is still a pending one

5.2.2.4 L6474x_OCD_TH

`enum L6474x_OCD_TH`

The L6474x_OCD_TH_t enum desribes the possible overcurrent threshold values which can be set in the library and which are then taken into account by the stepper driver

5.2.2.5 L6474x_State

`enum L6474x_State`

The LibL6474Config.h must be provided by the user project. The header in conf folder is only a template!

The L6474x_State_t enum is used to describe the current state of the driver library which helps the user to decide the next steps to be taken or to get notified about an unexpected state which is caused by an error

Enumerator

stRESET	stRESET is used to signal the initial power up state or a full device reset state in which the chip is kept until the user releases a reset state
stDISABLED	stDISABLED is used to signal the ready but disabled state of the chip. It is kept in this step until the user enables the power output stages
stENABLED	stENABLED is used to signal the ready and enabled state of the chip. It is kept in this step until the user disables the power output stages or an error happens
stINVALID	stINVALID is used to signal a state which requires user interaction because something is completely wrong in regular cases, this state can not be present. This is an indicator of memory corruption or misleading usage of the library

5.2.2.6 L6474x_StepMode

```
enum L6474x_StepMode
```

The L6474x_StepMode_t enum is used to describe the current step mode (resolution) of the driver library and the stepper motor. This changes the internal calculations for speed and position and requires a new reference run in case it will be changed at runtime.

Enumerator

smFULL	smFULL requires minimum of 4 steps for a full turn of the motor axis. This mode results in the highest torque but in the most noisy operation and the lowest resolution
smHALF	smHALF requires minimum of 8 steps for a full turn of the motor axis. better resolution but less torque
smMICRO4	smMICRO4 requires minimum of 16 steps for a full turn of the motor axis. better resolution but less torque
smMICRO8	smMICRO8 requires minimum of 32 steps for a full turn of the motor axis. better resolution but less torque
smMICRO16	smMICRO16 requires minimum of 64 steps for a full turn of the motor axis. better resolution but less torque

5.2.3 Function Documentation

5.2.3.1 L6474_CreateInstance()

```
L6474_Handle_t L6474_CreateInstance (
            L6474x_Platform_t * p,
            void * pIO,
            void * pGPO,
            void * pPWM )
```

L6474_CreateInstance is used once to create a library instance which encapsulates the handling for one stepper driver chip It needs a L6474x_Platform_t abstraction pointer and optionally some context pointers which are platform specific. The context pointers are passed to the abstraction functions which are provided by the platform structure.

In case it fails, a null pointer is returned. In case it was successful, a handle pointer is returned which is always required for all other API calls

5.2.3.2 L6474_GetAbsolutePosition()

```
int L6474_GetAbsolutePosition (
            L6474_Handle_t h,
            int * position )
```

func L6474_GetAbsolutePosition is used to read the current stepper position. The library must not be in stRESET state to perform this operation.

The function returns errcNONE in case no error happens or any other error code from L6474x_ErrorCode_t enum in case of an error.

param h is required and can not be null. the handle can be created by calling L6474_CreateInstance before.

param position is required. It returns the position

5.2.3.3 L6474_GetAlarmEnables()

```
int L6474_GetAlarmEnables (
            L6474_Handle_t h,
            int * bits )
```

func L6474_GetAlarmEnables is used to read the current enable bits for the ERROR ALARM and FLAG Pin. The library must not be in stRESET state to perform this operation.

The function returns errcNONE in case no error happens or any other error code from L6474x_ErrorCode_t enum in case of an error.

param h is required and can not be null. the handle can be created by calling L6474_CreateInstance before.

param bits is required. The bitmask can be found in the datasheet

5.2.3.4 L6474_GetElectricalPosition()

```
int L6474_GetElectricalPosition (
            L6474_Handle_t h,
            int * position )
```

func L6474_GetElectricalPosition is used to read the current stepper electrical position. The library must not be in stRESET state to perform this operation. The electrical Position returns the step state and micro step component of the driver.

The function returns errcNONE in case no error happens or any other error code from L6474x_ErrorCode_t enum in case of an error.

param h is required and can not be null. the handle can be created by calling L6474_CreateInstance before.

param position is required.

5.2.3.5 L6474_GetPositionMark()

```
int L6474_GetPositionMark (
            L6474_Handle_t h,
            int * position )
```

func L6474_GetPositionMark is used to read the current stepper position mark. The library must not be in stRESET state to perform this operation.

The function returns errcNONE in case no error happens or any other error code from L6474x_ErrorCode_t enum in case of an error.

param h is required and can not be null. the handle can be created by calling L6474_CreateInstance before.

param position is required.

5.2.3.6 L6474_GetProperty()

```
int L6474_GetProperty (
            L6474_Handle_t h,
            L6474_Property_t prop,
            int * value )
```

func L6474_SetProperty is used to read a property of the type of L6474_Property_t. The library must not be in stRESET state to perform this operation.

The function returns errcNONE in case no error happens or any other error code from L6474x_ErrorCode_t enum in case of an error.

param h is required and can not be null. the handle can be created by calling L6474_CreateInstance before.

param prop is required.

param value is required. The value formats can be found in the datasheet

5.2.3.7 L6474_GetStatus()

```
int L6474_GetStatus (
            L6474_Handle_t h,
            L6474_Status_t * state )
```

func L6474_GetStatus is used to read back the current libraries and devices state. The library must not be in stRESET to perform this operation.

The function returns errcNONE in case no error happens or any other error code from L6474x_ErrorCode_t enum in case of an error.

param h is required and can not be null. the handle can be created by calling L6474_CreateInstance before.

param state is required. And must not be null

5.2.3.8 L6474_GetStepMode()

```
int L6474_GetStepMode (
            L6474_Handle_t h,
            L6474x_StepMode_t * mode )
```

func L6474_SetStepMode is used to read back the step mode and therefore the resolution. The library must not be in stRESET to perform this operation.

The function returns errcNONE in case no error happens or any other error code from L6474x_ErrorCode_t enum in case of an error.

param h is required and can not be null. the handle can be created by calling L6474_CreateInstance before.

param mode is required and is one out of L6474x_StepMode_t

5.2.3.9 L6474_Initialize()

```
int L6474_Initialize (
            L6474_Handle_t h,
            L6474_BaseParameter_t * p )
```

func L6474_Initialize is used to set all basic parameters and take the required steps to bring up the driver into an idle state. It sets the library to stDISABLED in case no error happens.

The function returns errcNONE in case no error happens or any other error code from L6474x_ErrorCode_t enum in case of an error

param h is required and can not be null. the handle can be created by calling L6474_CreateInstance before.

param p is required and can not be null. the struct can be filled by calling L6474_SetBaseParameter before.

5.2.3.10 L6474_IsMoving()

```
int L6474_IsMoving (
            L6474_Handle_t h,
            int * moving )
```

func L6474_IsMoving is used to read the state of movement. In case the stepper moves, it returns true. The library must not be in stRESET state to perform this operation.

The function returns errcNONE in case no error happens or any other error code from L6474x_ErrorCode_t enum in case of an error.

param h is required and can not be null. the handle can be created by calling L6474_CreateInstance before.

param moving is required. It returns 0 when no movement is pending or 1 when there is a movement.

5.2.3.11 L6474_ResetStandBy()

```
int L6474_ResetStandBy (
            L6474_Handle_t h )
```

Calling L6474_ResetStandBy sets the driver in deep power down or reset state and the library will be set to default startup state again. All initialization steps must be done again by at least calling L6474_Initialize. The library is set to stRESET from any other state before.

The function returns errcNONE in case no error happens or any other error code from L6474x_ErrorCode_t enum in case of an error

note that in case the stepper is enabled an moving, the operation will be safely disabled but depending on the speed, the position can not be tracked properly anymore. So a new reference run after re-initialization is required!

param h is required and can not be null. the handle can be created by calling L6474_CreateInstance before.

5.2.3.12 L6474_SetAbsolutePosition()

```
int L6474_SetAbsolutePosition (
            L6474_Handle_t h,
            int position )
```

func L6474_SetAbsolutePosition is used to write the current stepper position. The library must not be in stRESET state to perform this operation.

The function returns errcNONE in case no error happens or any other error code from L6474x_ErrorCode_t enum in case of an error.

param h is required and can not be null. the handle can be created by calling L6474_CreateInstance before.

param position is required. It sets the position

5.2.3.13 L6474_SetAlarmEnables()

```
int L6474_SetAlarmEnables (
            L6474_Handle_t h,
            int bits )
```

func L6474_SetAlarmEnables is used to write the current enable bits for the ERROR ALARM and FLAG Pin. The library must not be in stRESET state to perform this operation.

The function returns errcNONE in case no error happens or any other error code from L6474x_ErrorCode_t enum in case of an error.

param h is required and can not be null. the handle can be created by calling L6474_CreateInstance before.

param bits is required. The bitmask can be found in the datasheet

5.2.3.14 L6474_SetBaseParameter()

```
int L6474_SetBaseParameter (
            L6474_BaseParameter_t * p )
```

Calling L6474_SetBaseParameter sets the basic global default parameters for all properties in the L6474_BaseParameter_t structure. These parameters are required when calling L6474_Initialize

The function returns errcNONE in case no error happens or any other error code from L6474x_ErrorCode_t enum in case of an error

5.2.3.15 L6474_SetElectricalPosition()

```
int L6474_SetElectricalPosition (
            L6474_Handle_t h,
            int position )
```

func L6474_SetElectricalPosition is used to write the current stepper electrical position. The library must not be in stRESET state to perform this operation. The electrical Position consists of the step state and micro step component of the driver.

The function returns errcNONE in case no error happens or any other error code from L6474x_ErrorCode_t enum in case of an error.

param h is required and can not be null. the handle can be created by calling L6474_CreateInstance before.

param position is required.

5.2.3.16 L6474_SetPositionMark()

```
int L6474_SetPositionMark (
            L6474_Handle_t h,
            int position )
```

func L6474_SetPositionMark is used to write the current stepper position mark. The library must not be in stRESET state to perform this operation.

The function returns errcNONE in case no error happens or any other error code from L6474x_ErrorCode_t enum in case of an error.

param h is required and can not be null. the handle can be created by calling L6474_CreateInstance before.

param position is required.

5.2.3.17 L6474_SetPowerOutputs()

```
int L6474_SetPowerOutputs (
            L6474_Handle_t h,
            int ena )
```

func L6474_SetPowerOutputs is used to enable or disable the driver ouptu stages. The library must not be in st↩ RESET to perform this operation. The libraries state will be changed to stENABLED or stDISABLED depending on the ena parameter.

The function returns errcNONE in case no error happens or any other error code from L6474x_ErrorCode_t enum in case of an error.

param h is required and can not be null. the handle can be created by calling L6474_CreateInstance before.

param ena is required and is either 0 when disabling or 1 when enabling is requested

5.2.3.18 L6474_SetProperty()

```
int L6474_SetProperty (
            L6474_Handle_t h,
            L6474_Property_t prop,
            int value )
```

func L6474_SetProperty is used to write a property of the type of L6474_Property_t. The library must not be in stRESET state to perform this operation.

The function returns errcNONE in case no error happens or any other error code from L6474x_ErrorCode_t enum in case of an error.

param h is required and can not be null. the handle can be created by calling L6474_CreateInstance before.

param prop is required.

param value is required. The value formats can be found in the datasheet

5.2.3.19 L6474_SetStepMode()

```
int L6474_SetStepMode (
            L6474_Handle_t h,
            L6474x_StepMode_t mode )
```

func L6474_SetStepMode is used to change the step mode and therefore the resolution. It is required to execute a new refernce run because the position value of the lib does not match the stepping mode anymore. The library must not be in stRESET to perform this operation.

The function returns errcNONE in case no error happens or any other error code from L6474x_ErrorCode_t enum in case of an error.

param h is required and can not be null. the handle can be created by calling L6474_CreateInstance before.

param mode is required and is one out of L6474x_StepMode_t

5.2.3.20 L6474_StepIncremental()

```
int L6474_StepIncremental (
            L6474_Handle_t h,
            int steps )
```

func L6474_StepIncremental is used to issue a movement with the given amount of steps. The library has to be in stENABLED state to perform this operation.

The function returns errcNONE in case no error happens or any other error code from L6474x_ErrorCode_t enum in case of an error.

param h is required and can not be null. the handle can be created by calling L6474_CreateInstance before.

param steps is required. negative values lead to a counter clock wise movement, 0 does not lead to any movement but returns errcNONE in case no other issue is present.

5.2.3.21 L6474_StopMovement()

```
int L6474_StopMovement (
            L6474_Handle_t h )
```

func L6474_StopMovement is used to stop a pending movement in case it has been configured as async. If not, it always return errcNONE in case no other issue is present. The library has to be in stENABLED state to perform this operation.

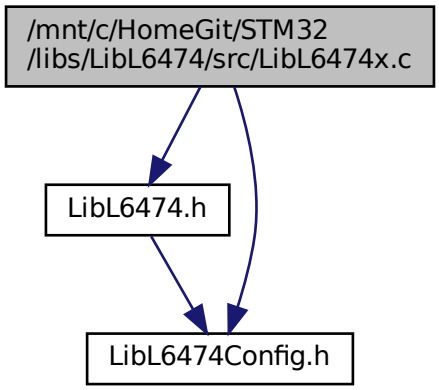
The function returns errcNONE in case no error happens or any other error code from L6474x_ErrorCode_t enum in case of an error.

param h is required and can not be null. the handle can be created by calling L6474_CreateInstance before.

5.3 /mnt/c/HomeGit/STM32/libs/LibL6474/src/LibL6474x.c File Reference

```
#include "LibL6474.h"
#include "LibL6474Config.h"
```

Include dependency graph for LibL6474x.c:



Data Structures

- struct L6474x_ParameterDescriptor
- struct L6474_Handle

Macros

- #define **IN_MILLISEC**(x) (x)
- #define **STEP_CMD_NOP_PREFIX** ((char)0x00)
- #define **STEP_CMD_NOP_LENGTH** 0x01
- #define **STEP_CMD_SET_PREFIX** ((char)0x00)
- #define **STEP_CMD_SET_LENGTH** 0x01
- #define **STEP_CMD_SET_MAX_PAYLOAD** 0x04
- #define **STEP_CMD_GET_PREFIX** ((char)0x20)
- #define **STEP_CMD_GET_LENGTH** 0x01
- #define **STEP_CMD_GET_MAX_PAYLOAD** 0x04
- #define **STEP_CMD_ENA_PREFIX** ((char)0xB8)
- #define **STEP_CMD_ENA_LENGTH** 0x01
- #define **STEP_CMD_DIS_PREFIX** ((char)0xA8)
- #define **STEP_CMD_DIS_LENGTH** 0x01
- #define **STEP_CMD_STA_PREFIX** ((char)0xD0)
- #define **STEP_CMD_STA_LENGTH** 0x03
- #define **STEP_REG_RANGE_MASK** 0x1F
- #define **STEP_REG_ABS_POS** 0x01
- #define **STEP_LEN_ABS_POS** 0x03
- #define **STEP_MASK_ABS_POS** 0x3FFFFF
- #define **STEP_OFFSET_ABS_POS** 0x0
- #define **STEP_REG_EL_POS** 0x02
- #define **STEP_LEN_EL_POS** 0x02
- #define **STEP_MASK_EL_POS** 0x1FF
- #define **STEP_OFFSET_EL_POS** 0x0
- #define **STEP_REG_MARK** 0x03
- #define **STEP_LEN_MARK** 0x03
- #define **STEP_MASK_MARK** 0x3FFFFF
- #define **STEP_OFFSET_MARK** 0x0
- #define **STEP_REG_TVAL** L6474_PROP_TORQUE
- #define **STEP_LEN_TVAL** 0x01
- #define **STEP_MASK_TVAL** 0x7F
- #define **STEP_OFFSET_TVAL** 0x0
- #define **STEP_REG_T_FAST** L6474_PROP_TFAST
- #define **STEP_LEN_T_FAST** 0x01
- #define **STEP_MASK_T_FAST** 0xFF
- #define **STEP_OFFSET_T_FAST** 0x0
- #define **STEP_REG_TON_MIN** L6474_PROP_TON
- #define **STEP_LEN_TON_MIN** 0x01
- #define **STEP_MASK_TON_MIN** 0x7F
- #define **STEP_OFFSET_TON_MIN** 0x0
- #define **STEP_REG_TOFF_MIN** L6474_PROP_TOFF
- #define **STEP_LEN_TOFF_MIN** 0x01
- #define **STEP_MASK_TOFF_MIN** 0x7F
- #define **STEP_OFFSET_TOFF_MIN** 0x0
- #define **STEP_REG_ADC_OUT** L6474_PROP_ADC_OUT
- #define **STEP_LEN_ADC_OUT** 0x01
- #define **STEP_MASK_ADC_OUT** 0x1F
- #define **STEP_OFFSET_ADC_OUT** 0x0
- #define **STEP_REG_OCD_TH** L6474_PROP_OCDTH
- #define **STEP_LEN_OCD_TH** 0x01
- #define **STEP_MASK_OCD_TH** 0x0F
- #define **STEP_OFFSET_OCD_TH** 0x0
- #define **STEP_REG_STEP_MODE** 0x16

- #define **STEP_LEN_STEP_MODE** 0x01
- #define **STEP_MASK_STEP_MODE** 0xFF
- #define **STEP_OFFSET_STEP_MODE** 0x0
- #define **STEP_REG_ALARM_EN** 0x17
- #define **STEP_LEN_ALARM_EN** 0x01
- #define **STEP_MASK_ALARM_EN** 0xFF
- #define **STEP_OFFSET_ALARM_EN** 0x0
- #define **STEP_REG_CONFIG** 0x18
- #define **STEP_LEN_CONFIG** 0x02
- #define **STEP_MASK_CONFIG** 0xFFFF
- #define **STEP_OFFSET_CONFIG** 0x0
- #define **STEP_REG_STATUS** 0x19
- #define **STEP_LEN_STATUS** 0x02
- #define **STEP_MASK_STATUS** 0xFFFF
- #define **STEP_OFFSET_STATUS** 0x0
- #define **STATUS_HIGHZ_MASK** (1 << 0)
- #define **STATUS_DIRECTION_MASK** (1 << 4)
- #define **STATUS_NOTPERF_CMD_MASK** (1 << 7)
- #define **STATUS_WRONG_CMD_MASK** (1 << 8)
- #define **STATUS_UNDERVOLT_MASK** (1 << 9)
- #define **STATUS_THR_WARN_MASK** (1 << 10)
- #define **STATUS_THR_SHORTD_MASK** (1 << 11)
- #define **STATUS_OCD_MASK** (1 << 12)

Typedefs

- typedef enum L6474x_AccessFlags **L6474x_AccessFlags_t**
- typedef struct L6474x_ParameterDescriptor **L6474x_ParameterDescriptor_t**

Enumerations

- enum **L6474x_AccessFlags** { **afNONE** = 0x00, **afREAD** = 0x01, **afWRITE** = 0x02, **afWRITE_HighZ** = 0x04 }

Functions

- static int **L6474_HelperLock** (L6474_Handle_t h)
- static void **L6474_HelperUnlock** (L6474_Handle_t h)
- static void **L6474_HelperReleaseStep** (L6474_Handle_t h)
- static int **L6474_GetStatusCommand** (L6474_Handle_t h)
- static int **L6474_NopCommand** (L6474_Handle_t h)
- static int **L6474_GetParamCommand** (L6474_Handle_t h, int addr)
- static int **L6474_SetParamCommand** (L6474_Handle_t h, int addr, int value)
- static int **L6474_EnableCommand** (L6474_Handle_t h)
- static int **L6474_DisableCommand** (L6474_Handle_t h)
- L6474_Handle_t L6474_CreateInstance (L6474x_Platform_t *p, void *pIO, void *pGPO, void *pPWM)
- int L6474_ResetStandBy (L6474_Handle_t h)
- int L6474_SetBaseParameter (L6474_BaseParameter_t *p)
- int L6474_Initialize (L6474_Handle_t h, L6474_BaseParameter_t *p)
- int L6474_IsMoving (L6474_Handle_t h, int *moving)
- int L6474_SetStepMode (L6474_Handle_t h, L6474x_StepMode_t mode)
- int L6474_GetStepMode (L6474_Handle_t h, L6474x_StepMode_t *mode)

- int L6474_SetPowerOutputs (L6474_Handle_t h, int ena)
- int L6474_GetAbsolutePosition (L6474_Handle_t h, int *position)
- int L6474_SetAbsolutePosition (L6474_Handle_t h, int position)
- int L6474_GetElectricalPosition (L6474_Handle_t h, int *position)
- int L6474_SetElectricalPosition (L6474_Handle_t h, int position)
- int L6474_GetPositionMark (L6474_Handle_t h, int *position)
- int L6474_SetPositionMark (L6474_Handle_t h, int position)
- int L6474_GetAlarmEnables (L6474_Handle_t h, int *bits)
- int L6474_SetProperty (L6474_Handle_t h, L6474_Property_t prop, int value)
- int L6474_GetProperty (L6474_Handle_t h, L6474_Property_t prop, int *value)
- int L6474_SetAlarmEnables (L6474_Handle_t h, int bits)
- int L6474_GetStatus (L6474_Handle_t h, L6474_Status_t *state)
- int L6474_StopMovement (L6474_Handle_t h)
- int L6474_StepIncremental (L6474_Handle_t h, int steps)

Variables

- static const L6474x_ParameterDescriptor_t **L6474_Parameters** [STEP_REG_RANGE_MASK]

5.3.1 Function Documentation

5.3.1.1 L6474_CreateInstance()

```
L6474_Handle_t L6474_CreateInstance (
            L6474x_Platform_t * p,
            void * pIO,
            void * pGPO,
            void * pPWM )
```

L6474_CreateInstance is used once to create a library instance which encapsulates the handling for one stepper driver chip It needs a L6474x_Platform_t abstraction pointer and optionally some context pointers which are platform specific. The context pointers are passed to the abstraction functions which are provided by the platform structure.

In case it fails, a null pointer is returned. In case it was successful, a handle pointer is returned which is always required for all other API calls

5.3.1.2 L6474_GetAbsolutePosition()

```
int L6474_GetAbsolutePosition (
            L6474_Handle_t h,
            int * position )
```

func L6474_GetAbsolutePosition is used to read the current stepper position. The library must not be in stRESET state to perform this operation.

The function returns errcNONE in case no error happens or any other error code from L6474x_ErrorCode_t enum in case of an error.

param h is required and can not be null. the handle can be created by calling L6474_CreateInstance before.

param position is required. It returns the position

5.3.1.3 L6474_GetAlarmEnables()

```
int L6474_GetAlarmEnables (
            L6474_Handle_t h,
            int * bits )
```

func L6474_GetAlarmEnables is used to read the current enable bits for the ERROR ALARM and FLAG Pin. The library must not be in stRESET state to perform this operation.

The function returns errcNONE in case no error happens or any other error code from L6474x_ErrorCode_t enum in case of an error.

param h is required and can not be null. the handle can be created by calling L6474_CreateInstance before.

param bits is required. The bitmask can be found in the datasheet

5.3.1.4 L6474_GetElectricalPosition()

```
int L6474_GetElectricalPosition (
            L6474_Handle_t h,
            int * position )
```

func L6474_GetElectricalPosition is used to read the current stepper electrical position. The library must not be in stRESET state to perform this operation. The electrical Position returns the step state and micro step component of the driver.

The function returns errcNONE in case no error happens or any other error code from L6474x_ErrorCode_t enum in case of an error.

param h is required and can not be null. the handle can be created by calling L6474_CreateInstance before.

param position is required.

5.3.1.5 L6474_GetPositionMark()

```
int L6474_GetPositionMark (
            L6474_Handle_t h,
            int * position )
```

func L6474_GetPositionMark is used to read the current stepper position mark. The library must not be in stRESET state to perform this operation.

The function returns errcNONE in case no error happens or any other error code from L6474x_ErrorCode_t enum in case of an error.

param h is required and can not be null. the handle can be created by calling L6474_CreateInstance before.

param position is required.

5.3.1.6 L6474_GetProperty()

```
int L6474_GetProperty (
            L6474_Handle_t h,
            L6474_Property_t prop,
            int * value )
```

func L6474_SetProperty is used to read a property of the type of L6474_Property_t. The library must not be in stRESET state to perform this operation.

The function returns errcNONE in case no error happens or any other error code from L6474x_ErrorCode_t enum in case of an error.

param h is required and can not be null. the handle can be created by calling L6474_CreateInstance before.

param prop is required.

param value is required. The value formats can be found in the datasheet

5.3.1.7 L6474_GetStatus()

```
int L6474_GetStatus (
            L6474_Handle_t h,
            L6474_Status_t * state )
```

func L6474_GetStatus is used to read back the current libraries and devices state. The library must not be in stRESET to perform this operation.

The function returns errcNONE in case no error happens or any other error code from L6474x_ErrorCode_t enum in case of an error.

param h is required and can not be null. the handle can be created by calling L6474_CreateInstance before.

param state is required. And must not be null

5.3.1.8 L6474_GetStepMode()

```
int L6474_GetStepMode (
            L6474_Handle_t h,
            L6474x_StepMode_t * mode )
```

func L6474_SetStepMode is used to read back the step mode and therefore the resolution. The library must not be in stRESET to perform this operation.

The function returns errcNONE in case no error happens or any other error code from L6474x_ErrorCode_t enum in case of an error.

param h is required and can not be null. the handle can be created by calling L6474_CreateInstance before.

param mode is required and is one out of L6474x_StepMode_t

5.3.1.9 L6474_Initialize()

```
int L6474_Initialize (
            L6474_Handle_t h,
            L6474_BaseParameter_t * p )
```

func L6474_Initialize is used to set all basic parameters and take the required steps to bring up the driver into an idle state. It sets the library to stDISABLED in case no error happens.

The function returns errcNONE in case no error happens or any other error code from L6474x_ErrorCode_t enum in case of an error

param h is required and can not be null. the handle can be created by calling L6474_CreateInstance before.

param p is required and can not be null. the struct can be filled by calling L6474_SetBaseParameter before.

5.3.1.10 L6474_IsMoving()

```
int L6474_IsMoving (
            L6474_Handle_t h,
            int * moving )
```

func L6474_IsMoving is used to read the state of movement. In case the stepper moves, it returns true. The library must not be in stRESET state to perform this operation.

The function returns errcNONE in case no error happens or any other error code from L6474x_ErrorCode_t enum in case of an error.

param h is required and can not be null. the handle can be created by calling L6474_CreateInstance before.

param moving is required. It returns 0 when no movement is pending or 1 when there is a movement.

5.3.1.11 L6474_ResetStandBy()

```
int L6474_ResetStandBy (
            L6474_Handle_t h )
```

Calling L6474_ResetStandBy sets the driver in deep power down or reset state and the library will be set to default startup state again. All initialization steps must be done again by at least calling L6474_Initialize. The library is set to stRESET from any other state before.

The function returns errcNONE in case no error happens or any other error code from L6474x_ErrorCode_t enum in case of an error

note that in case the stepper is enabled an moving, the operation will be safely disabled but depending on the speed, the position can not be tracked properly anymore. So a new reference run after re-initialization is required!

param h is required and can not be null. the handle can be created by calling L6474_CreateInstance before.

5.3.1.12 L6474_SetAbsolutePosition()

```
int L6474_SetAbsolutePosition (
            L6474_Handle_t h,
            int position )
```

func L6474_SetAbsolutePosition is used to write the current stepper position. The library must not be in stRESET state to perform this operation.

The function returns errcNONE in case no error happens or any other error code from L6474x_ErrorCode_t enum in case of an error.

param h is required and can not be null. the handle can be created by calling L6474_CreateInstance before.

param position is required. It sets the position

5.3.1.13 L6474_SetAlarmEnables()

```
int L6474_SetAlarmEnables (
            L6474_Handle_t h,
            int bits )
```

func L6474_SetAlarmEnables is used to write the current enable bits for the ERROR ALARM and FLAG Pin. The library must not be in stRESET state to perform this operation.

The function returns errcNONE in case no error happens or any other error code from L6474x_ErrorCode_t enum in case of an error.

param h is required and can not be null. the handle can be created by calling L6474_CreateInstance before.

param bits is required. The bitmask can be found in the datasheet

5.3.1.14 L6474_SetBaseParameter()

```
int L6474_SetBaseParameter (
            L6474_BaseParameter_t * p )
```

Calling L6474_SetBaseParameter sets the basic global default parameters for all properties in the L6474_BaseParameter_t structure. These parameters are required when calling L6474_Initialize

The function returns errcNONE in case no error happens or any other error code from L6474x_ErrorCode_t enum in case of an error

5.3.1.15 L6474_SetElectricalPosition()

```
int L6474_SetElectricalPosition (
            L6474_Handle_t h,
            int position )
```

func L6474_SetElectricalPosition is used to write the current stepper electrical position. The library must not be in stRESET state to perform this operation. The electrical Position consists of the step state and micro step component of the driver.

The function returns errcNONE in case no error happens or any other error code from L6474x_ErrorCode_t enum in case of an error.

param h is required and can not be null. the handle can be created by calling L6474_CreateInstance before.

param position is required.

5.3.1.16 L6474_SetPositionMark()

```
int L6474_SetPositionMark (
            L6474_Handle_t h,
            int position )
```

func L6474_SetPositionMark is used to write the current stepper position mark. The library must not be in stRESET state to perform this operation.

The function returns errcNONE in case no error happens or any other error code from L6474x_ErrorCode_t enum in case of an error.

param h is required and can not be null. the handle can be created by calling L6474_CreateInstance before.

param position is required.

5.3.1.17 L6474_SetPowerOutputs()

```
int L6474_SetPowerOutputs (
            L6474_Handle_t h,
            int ena )
```

func L6474_SetPowerOutputs is used to enable or disable the driver ouptu stages. The library must not be in st↩ RESET to perform this operation. The libraries state will be changed to stENABLED or stDISABLED depending on the ena parameter.

The function returns errcNONE in case no error happens or any other error code from L6474x_ErrorCode_t enum in case of an error.

param h is required and can not be null. the handle can be created by calling L6474_CreateInstance before.

param ena is required and is either 0 when disabling or 1 when enabling is requested

5.3.1.18 L6474_SetProperty()

```
int L6474_SetProperty (
            L6474_Handle_t h,
            L6474_Property_t prop,
            int value )
```

func L6474_SetProperty is used to write a property of the type of L6474_Property_t. The library must not be in stRESET state to perform this operation.

The function returns errcNONE in case no error happens or any other error code from L6474x_ErrorCode_t enum in case of an error.

param h is required and can not be null. the handle can be created by calling L6474_CreateInstance before.

param prop is required.

param value is required. The value formats can be found in the datasheet

5.3.1.19 L6474_SetStepMode()

```
int L6474_SetStepMode (
            L6474_Handle_t h,
            L6474x_StepMode_t mode )
```

func L6474_SetStepMode is used to change the step mode and therefore the resolution. It is required to execute a new refernce run because the position value of the lib does not match the stepping mode anymore. The library must not be in stRESET to perform this operation.

The function returns errcNONE in case no error happens or any other error code from L6474x_ErrorCode_t enum in case of an error.

param h is required and can not be null. the handle can be created by calling L6474_CreateInstance before.

param mode is required and is one out of L6474x_StepMode_t

5.3.1.20 L6474_StepIncremental()

```
int L6474_StepIncremental (
            L6474_Handle_t h,
            int steps )
```

func L6474_StepIncremental is used to issue a movement with the given amount of steps. The library has to be in stENABLED state to perform this operation.

The function returns errcNONE in case no error happens or any other error code from L6474x_ErrorCode_t enum in case of an error.

param h is required and can not be null. the handle can be created by calling L6474_CreateInstance before.

param steps is required. negative values lead to a counter clock wise movement, 0 does not lead to any movement but returns errcNONE in case no other issue is present.

5.3.1.21 L6474_StopMovement()

```
int L6474_StopMovement (
            L6474_Handle_t h )
```

func L6474_StopMovement is used to stop a pending movement in case it has been configured as async. If not, it always return errcNONE in case no other issue is present. The library has to be in stENABLED state to perform this operation.

The function returns errcNONE in case no error happens or any other error code from L6474x_ErrorCode_t enum in case of an error.

param h is required and can not be null. the handle can be created by calling L6474_CreateInstance before.

5.3.2 Variable Documentation

5.3.2.1 L6474_Parameters

```
const L6474x_ParameterDescriptor_t L6474_Parameters[STEP_REG_RANGE_MASK]  [static]
```

Initial value:

```
= {
    [STEP_REG_ABS_POS]   = { .command = STEP_REG_ABS_POS,   .defined = 1, .length = STEP_LEN_ABS_POS,
       .mask = STEP_MASK_ABS_POS,   .name = "ABS_POS",   .flags = afREAD | afWRITE       },
    [STEP_REG_EL_POS]    = { .command = STEP_REG_EL_POS,    .defined = 1, .length = STEP_LEN_EL_POS,
       .mask = STEP_MASK_EL_POS,    .name = "EL_POS",    .flags = afREAD | afWRITE       },
    [STEP_REG_MARK]      = { .command = STEP_REG_MARK,      .defined = 1, .length = STEP_LEN_MARK,
       .mask = STEP_MASK_MARK,      .name = "MARK",      .flags = afREAD | afWRITE       },
    [STEP_REG_TVAL]      = { .command = STEP_REG_TVAL,      .defined = 1, .length = STEP_LEN_TVAL,
       .mask = STEP_MASK_TVAL,      .name = "TVAL",      .flags = afREAD | afWRITE       },
    [STEP_REG_T_FAST]    = { .command = STEP_REG_T_FAST,    .defined = 1, .length = STEP_LEN_T_FAST,
       .mask = STEP_MASK_T_FAST,    .name = "T_FAST",    .flags = afREAD | afWRITE_HighZ },
    [STEP_REG_TON_MIN]   = { .command = STEP_REG_TON_MIN,   .defined = 1, .length = STEP_LEN_TON_MIN,
       .mask = STEP_MASK_TON_MIN,   .name = "TON_MIN",   .flags = afREAD | afWRITE_HighZ },
    [STEP_REG_TOFF_MIN]  = { .command = STEP_REG_TOFF_MIN,  .defined = 1, .length = STEP_LEN_TOFF_MIN,
       .mask = STEP_MASK_TOFF_MIN,  .name = "TOFF_MIN",  .flags = afREAD | afWRITE_HighZ },
    [STEP_REG_ADC_OUT]   = { .command = STEP_REG_ADC_OUT,   .defined = 1, .length = STEP_LEN_ADC_OUT,
       .mask = STEP_MASK_ADC_OUT,   .name = "ADC_OUT",   .flags = afREAD                 },
    [STEP_REG_OCD_TH]    = { .command = STEP_REG_OCD_TH,    .defined = 1, .length = STEP_LEN_OCD_TH,
       .mask = STEP_MASK_OCD_TH,    .name = "OCD_TH",    .flags = afREAD | afWRITE       },
    [STEP_REG_STEP_MODE] = { .command = STEP_REG_STEP_MODE, .defined = 1, .length = STEP_LEN_STEP_MODE,
       .mask = STEP_MASK_STEP_MODE, .name = "STEP_MODE", .flags = afREAD | afWRITE_HighZ },
    [STEP_REG_ALARM_EN]  = { .command = STEP_REG_ALARM_EN,  .defined = 1, .length = STEP_LEN_ALARM_EN,
       .mask = STEP_MASK_ALARM_EN,  .name = "ALARM_EN",  .flags = afREAD | afWRITE       },
    [STEP_REG_CONFIG]    = { .command = STEP_REG_CONFIG,    .defined = 1, .length = STEP_LEN_CONFIG,
       .mask = STEP_MASK_CONFIG,    .name = "CONFIG",    .flags = afREAD | afWRITE_HighZ },
    [STEP_REG_STATUS]    = { .command = STEP_REG_STATUS,    .defined = 1, .length = STEP_LEN_STATUS,
       .mask = STEP_MASK_STATUS,    .name = "STATUS",    .flags = afREAD                 }
}
```

Index