

## Claw Device Controller

Generated by Doxygen 1.16.0



<b>1 claw</b>	<b>1</b>
<b>2 Class Index</b>	<b>3</b>
2.1 Class List . . . . .	3
<b>3 File Index</b>	<b>5</b>
3.1 File List . . . . .	5
<b>4 Class Documentation</b>	<b>7</b>
4.1 stepper_state Struct Reference . . . . .	7
4.1.1 Detailed Description . . . . .	7
<b>5 File Documentation</b>	<b>9</b>
5.1 claw.c File Reference . . . . .	9
5.1.1 Detailed Description . . . . .	11
5.1.2 Function Documentation . . . . .	11
5.1.2.1 main() . . . . .	11
5.1.2.2 pico_led_init() . . . . .	11
5.1.2.3 pico_set_led() . . . . .	12
5.1.2.4 process_command() . . . . .	12
5.1.2.5 process_led_tick() . . . . .	12
5.1.2.6 process_stdin_input() . . . . .	13
5.1.2.7 process_stepper_movement() . . . . .	13
5.1.2.8 stepper_enable() . . . . .	13
5.1.2.9 stepper_get_status() . . . . .	14
5.1.2.10 stepper_init() . . . . .	14
5.1.2.11 stepper_set_step_period() . . . . .	14
5.1.2.12 stepper_set_target_position() . . . . .	15
5.1.2.13 stepper_stop() . . . . .	15
5.1.2.14 timer_callback() . . . . .	15
5.1.3 Variable Documentation . . . . .	16
5.1.3.1 help_message . . . . .	16
5.1.3.2 led_period . . . . .	16
5.1.3.3 ms_ticks_count . . . . .	16
5.1.3.4 ten_us_ticks_count . . . . .	16
<b>Index</b>	<b>17</b>



# Chapter 1

## claw

Raspberry pico 2 code to drive stepper motor driven claw



## Chapter 2

# Class Index

### 2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

<a href="#">stepper_state</a>	Structure to hold stepper motor state . . . . .	7
-------------------------------	---	---





## Chapter 3

# File Index

### 3.1 File List

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

<a href="#">claw.c</a>	Firmware to control a claw device with stepper motors via USB serial commands . . . . .	9
------------------------	---	---



## Chapter 4

# Class Documentation

### 4.1 stepper\_state Struct Reference

Structure to hold stepper motor state.

#### Public Attributes

- int **current\_position**  
*Current position in steps.*
- int **target\_position**  
*Target position in steps.*
- int **step\_period**  
*Step period in TIMMER\_INTERVAL\_US units.*
- bool **moving**  
*Is the stepper currently moving.*
- bool **enabled**  
*Is the stepper enabled.*

#### 4.1.1 Detailed Description

Structure to hold stepper motor state.

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

- [claw.c](#)



# Chapter 5

## File Documentation

### 5.1 claw.c File Reference

Firmware to control a claw device with stepper motors via USB serial commands.

```
#include <stdio.h>
#include "pico/stdlib.h"
#include "hardware/timer.h"
#include <string.h>
#include "pico/assert.h"
#include <stdlib.h>
#include "hardware/gpio.h"
#include <ctype.h>
#include <math.h>
```

#### Classes

- struct [stepper\\_state](#)  
*Structure to hold stepper motor state.*

#### Macros

- #define LED\_DELAY\_MS 1000
- #define TIMER\_INTERVAL\_US 10
- #define DEFAULT\_STEPPER\_PERIOD 4
- #define MIN\_STEPPER\_PERIOD 4
- #define STEPPER\_STEP\_PIN 2
- #define STEPPER\_DIR\_PIN 3
- #define STEPPER\_ENABLE\_PIN 4
- #define STEPPER\_ENABLE\_PIN\_INVERTED true
- #define STEPPER\_DIRECTION\_FORWARD 1
- #define STEPPER\_DIRECTION\_BACKWARD 0
- #define STEPPER\_STEPS\_PER\_REV 3200
- #define STEPPER\_MAX\_REVOLUTIONS 12
- #define STEPPER\_BUMP\_STEPS (STEPPER\_STEPS\_PER\_REV / 4)
- #define MAX\_STEPPER\_POSITION (STEPPER\_STEPS\_PER\_REV \* STEPPER\_MAX\_REVOLUTIONS)

- `#define MIN_STEPPER_POSITION 0`
- `#define MAX_COMMAND_LENGTH 50`
- `#define CLAW_SET_POSITION_COMMAND "claw_set "`
- `#define LED_PERIOD_COMMAND "led_period "`
- `#define SET_STEPPER_PERIOD_COMMAND "set_stepper_period "`
- `#define SET_STEPPER_ZERO_COMMAND "set_stepper_zero"`
- `#define MOVE_STEPPER_ABSOLUTE_COMMAND "move_stepper_absolute "`
- `#define MOVE_STEPPER_RELATIVE_COMMAND "move_stepper_relative "`
- `#define MOVE_STEPPER_ROTATIONS_COMMAND "move_stepper_rotations "`
- `#define MOVE_STEPPER_BUMP_DOWN_COMMAND "move_stepper_bump_down"`
- `#define STOP_STEPPER_COMMAND "stop_stepper"`
- `#define GET_STEPPER_STATUS_COMMAND "get_stepper_status"`
- `#define ENABLE_STEPPER_COMMAND "enable_stepper"`
- `#define DISABLE_STEPPER_COMMAND "disable_stepper"`

## Typedefs

- typedef struct [stepper\\_state](#) [stepper\\_state\\_t](#)

## Functions

- bool [stepper\\_init](#) ([stepper\\_state\\_t](#) \*stepper, int initial\_position, int step\_period)  
*Initialize the stepper state.*
- bool [stepper\\_set\\_target\\_position](#) ([stepper\\_state\\_t](#) \*stepper, int target\_position)  
*Set the target position for the stepper motor.*
- bool [stepper\\_set\\_step\\_period](#) ([stepper\\_state\\_t](#) \*stepper, int step\_period\_us)  
*Set the step period for the stepper motor.*
- bool [stepper\\_stop](#) ([stepper\\_state\\_t](#) \*stepper)  
*Stop the stepper motor, setting target position to current position.*
- bool [stepper\\_get\\_status](#) ([stepper\\_state\\_t](#) \*stepper)  
*Get the current status of the stepper motor, printing to stdout.*
- bool [stepper\\_enable](#) ([stepper\\_state\\_t](#) \*stepper, bool enable)  
*Enable the stepper motor.*
- int [pico\\_led\\_init](#) (void)  
*Initialise the LED.*
- void [pico\\_set\\_led](#) (bool led\_on)  
*Turn the LED on or off.*
- bool [process\\_stepper\\_movement](#) ([stepper\\_state\\_t](#) \*stepper)  
*Process stepper movement.*
- void [process\\_led\\_tick](#) (void)  
*Process LED timing tick.*
- bool [timer\\_callback](#) (struct repeating\_timer \*)  
*Millisecond timer callback.*
- int [process\\_command](#) (const char \*cmd, [stepper\\_state\\_t](#) \*stepper)  
*Process a command string.*
- char \* [process\\_stdin\\_input](#) (void)  
*Process stdin input.*
- int [main](#) ()  
*Main function.*

## Variables

- volatile int `led_period` = LED\_DELAY\_MS  
*LED blink period in milliseconds.*
- volatile int `ten_us_ticks_count` = 0  
*Global ten microsecond ticks count.*
- volatile int `ms_ticks_count` = 0  
*Global millisecond ticks count.*
- const char \* `help_message`  
*Help message.*

### 5.1.1 Detailed Description

Firmware to control a claw device with stepper motors via USB serial commands.

#### Author

Jon Wade

#### Date

19 Dec 2025

#### Copyright

(c) 2025 Jon Wade. Standard MIT License applies. See LICENSE file.

Command interface for controlling stepper motors and other functions associated with the claw device. This file implements a simple command interface over USB serial to control the claw device.

### 5.1.2 Function Documentation

#### 5.1.2.1 `main()`

```
int main ()
```

Main function.

#### Parameters

	none
--	------

#### Returns

: none

---

#### 5.1.2.2 `pico_led_init()`

Generated by Doxygen

```
int pico_led_init (  
    void )
```

**Parameters**

	none
--	------

**Returns**

: PICO\_OK on success error code on failure

**5.1.2.3 pico\_set\_led()**

```
void pico_set_led (  
    bool led_on)
```

Turn the LED on or off.

**Parameters**

<i>led_on</i>	true to turn on, false to turn off
---------------	------------------------------------

**Returns**

: none

**5.1.2.4 process\_command()**

```
int process_command (  
    const char * cmd,  
    stepper_state_t * stepper)
```

Process a command string.

**Parameters**

<i>cmd</i>	pointer to command string
<i>stepper</i>	pointer to stepper state structure

**Returns**

: 0 on success, error code on failure

**5.1.2.5 process\_led\_tick()**

```
void process_led_tick (  
    void )
```

Process LED timing tick.



**Parameters**

	none
--	------

**Returns**

: none

**5.1.2.6 process\_stdin\_input()**

```
char * process_stdin_input (
    void )
```

Process stdin input.

**Note**

: This function reads characters from stdin, builds commands, and processes them when a newline is received.

This function has a simple lock to prevent re-entrancy.

**Parameters**

	none
--	------

**Returns**

: none

**5.1.2.7 process\_stepper\_movement()**

```
bool process_stepper_movement (
    stepper_state_t * stepper)
```

Process stepper movement.

**Parameters**

<i>stepper</i>	pointer to stepper state structure
----------------	------------------------------------

**Returns**

: true if stepper is still moving, false if it has reached target

**5.1.2.8 stepper\_enable()**

---

Generated by Doxygen

```
bool stepper_enable (
    stepper_state_t * stepper,
    bool enable)
```

**Parameters**

<i>stepper</i>	pointer to stepper state structure, must not be NULL
<i>enable</i>	true to enable, false to disable

**Returns**

: true on success, false on failure

**5.1.2.9 stepper\_get\_status()**

```
bool stepper_get_status (
    stepper_state_t * stepper)
```

Get the current status of the stepper motor, printing to stdout.

**Parameters**

<i>stepper</i>	pointer to stepper state structure, must not be NULL
----------------	--

**Returns**

: true on success, false on failure

**5.1.2.10 stepper\_init()**

```
bool stepper_init (
    stepper_state_t * stepper,
    int initial_position,
    int step_period)
```

Initialize the stepper state.

**Parameters**

<i>stepper</i>	pointer to stepper state structure to initialize, must not be NULL
<i>initial_position</i>	initial position in steps must be between MIN_STEPPER_POSITION and MAX_STEPPER_POSITION
<i>step_period</i>	step period in TIMER_INTERVAL_US must be greater than 1 ms

**Returns**

: true on success, false on failure

**5.1.2.11 stepper\_set\_step\_period()**

```
bool stepper_set_step_period (
    stepper_state_t * stepper,
    int step_period_us)
```

**Parameters**

<i>stepper</i>	pointer to stepper state structure, must not be NULL
<i>step_period</i>	step period in microseconds must be greater than MIN_STEPPER_PERIOD

**Returns**

: true on success, false on failure

**5.1.2.12 stepper\_set\_target\_position()**

```
bool stepper_set_target_position (
    stepper_state_t * stepper,
    int target_position)
```

Set the target position for the stepper motor.

**Parameters**

<i>stepper</i>	pointer to stepper state structure, must not be NULL
<i>target_position</i>	target position in steps must be between MIN_STEPPER_POSITION and MAX_STEPPER_POSITION

**Returns**

: true on success, false on failure

**5.1.2.13 stepper\_stop()**

```
bool stepper_stop (
    stepper_state_t * stepper)
```

Stop the stepper motor, setting target position to current position.

**Parameters**

<i>stepper</i>	pointer to stepper state structure, must not be NULL
----------------	--

**Returns**

: true on success, false on failure

**5.1.2.14 timer\_callback()**

```
bool timer_callback (
    struct repeating_timer * t)
```

Millisecond timer callback.

## Parameters

<i>t</i>	pointer to repeating_timer struct
----------	-----------------------------------

## Returns

: true to keep repeating, false to stop

## 5.1.3 Variable Documentation

### 5.1.3.1 help\_message

```
const char* help_message
```

#### Initial value:

```

"\\n"
"Available commands:\\n"
"  claw_set <position>          - Set the claw position 0 to 100\\n"
"  led_period <ms>              - Set the LED blink period in milliseconds\\n"
"  set_stepper_period <us>      - Set the stepper motor step period in us\\n"
"  set_stepper_zero             - Set the current position to zero\\n"
"  move_stepper_absolute <steps> - Move the stepper to an absolute position\\n"
"  move_stepper_relative <steps> - Move the stepper by a relative number of steps\\n"
"  move_stepper_rotations <rotations> - Move the stepper by a number of rotations\\n"
"  move_stepper_bump_down       - Move the stepper down by a small fixed amount\\n"
"  stop_stepper                 - Stop the stepper motor\\n"
"  get_stepper_status           - Get the current status of the stepper motor\\n"
"  enable_stepper               - Enable the stepper motor\\n"
"  disable_stepper              - Disable the stepper motor\\n"
"  help                         - Show this help message\\n"
"-----\\n"

```

Help message.

This message is displayed when the user requests help or enters an unknown command.

### 5.1.3.2 led\_period

```
volatile int led_period = LED_DELAY_MS
```

LED blink period in milliseconds.

This variable can be modified via command interface to change the LED blink rate.

### 5.1.3.3 ms\_ticks\_count

```
volatile int ms_ticks_count = 0
```

Global millisecond ticks count.

This variable is incremented by the timer callback every 100 calls (1 ms = 100 \* 10 us) and decremented in the main loop to track when the millisecond tasks should run.

### 5.1.3.4 ten\_us\_ticks\_count

```
volatile int ten_us_ticks_count = 0
```

Global ten microsecond ticks count.

This variable is incremented by the timer callback and decremented in the main loop to track when the ten microsecond tasks should run.

# Index

- claw, [1](#)
- claw.c, [9](#)
  - help\_message, [16](#)
  - led\_period, [16](#)
  - main, [11](#)
  - ms\_ticks\_count, [16](#)
  - pico\_led\_init, [11](#)
  - pico\_set\_led, [12](#)
  - process\_command, [12](#)
  - process\_led\_tick, [12](#)
  - process\_stdin\_input, [13](#)
  - process\_stepper\_movement, [13](#)
  - stepper\_enable, [13](#)
  - stepper\_get\_status, [14](#)
  - stepper\_init, [14](#)
  - stepper\_set\_step\_period, [14](#)
  - stepper\_set\_target\_position, [15](#)
  - stepper\_stop, [15](#)
  - ten\_us\_ticks\_count, [16](#)
  - timer\_callback, [15](#)
- help\_message
  - claw.c, [16](#)
- led\_period
  - claw.c, [16](#)
- main
  - claw.c, [11](#)
- ms\_ticks\_count
  - claw.c, [16](#)
- pico\_led\_init
  - claw.c, [11](#)
- pico\_set\_led
  - claw.c, [12](#)
- process\_command
  - claw.c, [12](#)
- process\_led\_tick
  - claw.c, [12](#)
- process\_stdin\_input
  - claw.c, [13](#)
- process\_stepper\_movement
  - claw.c, [13](#)
- stepper\_enable
  - claw.c, [13](#)
- stepper\_get\_status
  - claw.c, [14](#)
- stepper\_init
  - claw.c, [14](#)
- stepper\_set\_step\_period
  - claw.c, [14](#)
- stepper\_set\_target\_position
  - claw.c, [15](#)
- stepper\_state, [7](#)
- stepper\_stop
  - claw.c, [15](#)
- ten\_us\_ticks\_count
  - claw.c, [16](#)
- timer\_callback
  - claw.c, [15](#)