# MyOs

0.0.0

Generated by Doxygen 1.9.1

# Chapter 1

# Data Structure Index

## 1.1 Data Structures

Here are the data structures with brief descriptions:

# Chapter 2

# File Index

## 2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

# Data Structure Documentation

## 3.1 ctimer_t Struct Reference

```
#include <ctimer.h>
```

Collaboration diagram for ctimer_t:

### Data Fields

- ptimer_t ptimer
- process_t ∗ context
- ctimer_callback_t callback
- void ∗ data

### 3.1.1 Detailed Description

Definition at line 68 of file ctimer.h.

### 3.1.2 Field Documentation

#### 3.1.2.1 callback

```
ctimer_callback_t callback
```

Callback function to be called when process timer expires

Definition at line 71 of file ctimer.h.

#### 3.1.2.2 context

```
process_t* context
```

Context in which to invoke the callback function

Definition at line 70 of file ctimer.h.

#### 3.1.2.3 data

```
void* data
```

Definition at line 72 of file ctimer.h.

#### 3.1.2.4 ptimer

```
ptimer_t ptimer
```

Instance of process timer control structure

Definition at line 69 of file ctimer.h.

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

- myos/os/ctimer.h

## 3.2 dlist_node_t Struct Reference

```
#include <dlist.h>
```

Collaboration diagram for dlist_node_t:

### Data Fields

- dlist_node_t ∗ next
- dlist_node_t ∗ prev

### 3.2.1 Detailed Description

Definition at line 261 of file dlist.h.

### 3.2.2 Field Documentation

#### 3.2.2.1 next

```
dlist_node_t* next
```

Definition at line 262 of file dlist.h.

#### 3.2.2.2 prev

```
dlist_node_t* prev
```

Definition at line 263 of file dlist.h.

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

- myos/lib/dlist.h

## 3.3 etimer_t Struct Reference

```
#include <etimer.h>
```

Collaboration diagram for etimer_t:

### Data Fields

- ptimer_t ptimer
- process_event_t evt

### 3.3.1 Detailed Description

Definition at line 42 of file etimer.h.

### 3.3.2 Field Documentation

#### 3.3.2.1 evt

```
process_event_t evt
```

Definition at line 44 of file etimer.h.

#### 3.3.2.2 ptimer

```
ptimer_t ptimer
```

Definition at line 43 of file etimer.h.

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

- myos/os/etimer.h

## 3.4 myos_timer_t Struct Reference

```
#include <timer.h>
```

### Data Fields

- timestamp_t start
- timespan_t span

### 3.4.1 Detailed Description

Definition at line 51 of file timer.h.

### 3.4.2 Field Documentation

#### 3.4.2.1 span

```
timespan_t span
```

Definition at line 53 of file timer.h.

#### 3.4.2.2 start

```
timestamp_t start
```

Definition at line 52 of file timer.h.

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

- myos/os/timer.h

## 3.5 process_event_t Struct Reference

```
#include <process.h>
```

Collaboration diagram for process_event_t:

### Data Fields

- process_event_id_t id
- void ∗ data
- process_t ∗ from
- process_t ∗ to

### 3.5.1 Detailed Description

Definition at line 105 of file process.h.

### 3.5.2 Field Documentation

#### 3.5.2.1 data

```
void* data
```

Definition at line 107 of file process.h.

#### 3.5.2.2 from

```
process_t* from
```

Definition at line 108 of file process.h.

#### 3.5.2.3 id

```
process_event_id_t id
```

Definition at line 106 of file process.h.

#### 3.5.2.4 to

```
process_t* to
```

Definition at line 109 of file process.h.

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

- myos/os/process.h

## 3.6 process_t Struct Reference

```
#include <process.h>
```

Collaboration diagram for process_t:

### Data Fields

- PLIST_NODE_TYPE
- process_thread_t thread
- void ∗ data
- pt_t pt
- bool pollreq

### 3.6.1 Detailed Description

Definition at line 91 of file process.h.

### 3.6.2 Field Documentation

#### 3.6.2.1 data

```
void* data
```

Definition at line 94 of file process.h.

#### 3.6.2.2 PLIST_NODE_TYPE

```
PLIST_NODE_TYPE
```

Definition at line 92 of file process.h.

#### 3.6.2.3 pollreq

```
bool pollreq
```

Definition at line 101 of file process.h.

#### 3.6.2.4 pt

```
pt_t pt
```

Definition at line 95 of file process.h.

#### 3.6.2.5 thread

```
process_thread_t thread
```

Definition at line 93 of file process.h.

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

- myos/os/process.h

## 3.7 pt_t Struct Reference

```
#include <pt.h>
```

### Data Fields

- lc_t lc

### 3.7.1 Detailed Description

Definition at line 65 of file pt.h.

### 3.7.2 Field Documentation

#### 3.7.2.1 lc

```
lc_t lc
```

Definition at line 66 of file pt.h.

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

- myos/os/pt.h

## 3.8 ptimer_t Struct Reference

```
#include <ptimer.h>
```

Collaboration diagram for ptimer_t:

### Data Fields

- DLIST_NODE_TYPE
- myos_timer_t timer
- ptimer_handler_t handler
- bool running

### 3.8.1 Detailed Description

Definition at line 82 of file ptimer.h.

### 3.8.2 Field Documentation

#### 3.8.2.1 DLIST_NODE_TYPE

```
DLIST_NODE_TYPE
```

Definition at line 83 of file ptimer.h.

#### 3.8.2.2 handler

```
ptimer_handler_t handler
```

Definition at line 85 of file ptimer.h.

#### 3.8.2.3 running

```
bool running
```

Definition at line 86 of file ptimer.h.

#### 3.8.2.4 timer

```
myos_timer_t timer
```

Definition at line 84 of file ptimer.h.

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

- myos/os/ptimer.h

## 3.9 rtimer_t Struct Reference

```
#include <rtimer.h>
```

### Data Fields

- rtimer_timestamp_t start
- rtimer_timespan_t span
- rtimer_callback_t callback
- void * data

### 3.9.1 Detailed Description

Definition at line 53 of file rtimer.h.

### 3.9.2 Field Documentation

#### 3.9.2.1 callback

```
rtimer_callback_t callback
```

Definition at line 56 of file rtimer.h.

#### 3.9.2.2 data

```
void* data
```

Definition at line 57 of file rtimer.h.

#### 3.9.2.3 span

```
rtimer_timespan_t span
```

Definition at line 55 of file rtimer.h.

#### 3.9.2.4 start

```
rtimer_timestamp_t start
```

Definition at line 54 of file rtimer.h.

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

- myos/os/rtimer.h

## 3.10 slist_node_t Struct Reference

```
#include <slist.h>
```

Collaboration diagram for slist_node_t:

### Data Fields

- slist_node_t ∗ next

### 3.10.1 Detailed Description

Definition at line 292 of file slist.h.

### 3.10.2 Field Documentation

#### 3.10.2.1 next

```
slist_node_t* next
```

Pointer to the next node in the list.

Definition at line 293 of file slist.h.

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

- myos/lib/slist.h

## 3.11 uibutton_t Struct Reference

```
#include <uibuttons.h>
```

### Data Fields

- uibuttons_get_t get
- uint8_t prev_state

### 3.11.1 Detailed Description

Definition at line 62 of file uibuttons.h.

### 3.11.2 Field Documentation

#### 3.11.2.1 get

```
uibuttons_get_t get
```

Definition at line 63 of file uibuttons.h.

#### 3.11.2.2 prev_state

```
uint8_t prev_state
```

Definition at line 64 of file uibuttons.h.

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

- myos/ui/uibuttons.h

## 3.12 uigfx_font_t Struct Reference

```
#include <uigfx.h>
```

### Data Fields

- const uint8_t xsz
- const uint8_t ysz
- const void ∗ data
- const void(∗ draw )(uint8_t ∗buf, int16_t x0, int16_t y0, char ch, uigfx_color_t c)

### 3.12.1 Detailed Description

Definition at line 47 of file uigfx.h.

### 3.12.2 Field Documentation

#### 3.12.2.1 data

```
const void* data
```

Definition at line 50 of file uigfx.h.

#### 3.12.2.2 draw

```
const void(* draw) (uint8_t *buf, int16_t x0, int16_t y0, char ch, uigfx_color_t c)
```

Definition at line 51 of file uigfx.h.

#### 3.12.2.3 xsz

```
const uint8_t xsz
```

Definition at line 48 of file uigfx.h.

#### 3.12.2.4 ysz

```
const uint8_t ysz
```

Definition at line 49 of file uigfx.h.

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

- myos/ui/uigfx/uigfx.h

## 3.13 uigfx_image_t Struct Reference

```
#include <uigfx.h>
```

### Data Fields

- const uint8_t ∗ data
- uint16_t xres
- uint16_t yres
- uint8_t bbp

### 3.13.1 Detailed Description

Definition at line 59 of file uigfx.h.

### 3.13.2 Field Documentation

#### 3.13.2.1 bbp

```
uint8_t bbp
```

Definition at line 63 of file uigfx.h.

#### 3.13.2.2 data

```
const uint8_t* data
```

Definition at line 60 of file uigfx.h.

#### 3.13.2.3 xres

```
uint16_t xres
```

Definition at line 61 of file uigfx.h.

#### 3.13.2.4 yres

```
uint16_t yres
```

Definition at line 62 of file uigfx.h.

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

- myos/ui/uigfx/uigfx.h

## 3.14 uigfx_widget_t Struct Reference

```
#include <uigfx.h>
```

### Data Fields

- int16_t xpos
- int16_t ypos
- uint16_t xres
- uint16_t yres

### 3.14.1 Detailed Description

Definition at line 54 of file uigfx.h.

### 3.14.2 Field Documentation

#### 3.14.2.1 xpos

```
int16_t xpos
```

Definition at line 55 of file uigfx.h.

#### 3.14.2.2 xres

```
uint16_t xres
```

Definition at line 56 of file uigfx.h.

#### 3.14.2.3 ypos

```
int16_t ypos
```

Definition at line 55 of file uigfx.h.

#### 3.14.2.4 yres

```
uint16_t yres
```

Definition at line 56 of file uigfx.h.

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

- myos/ui/uigfx/uigfx.h

## 3.15 uigfx_window_t Struct Reference

```
#include <uigfx_win.h>
```

Collaboration diagram for uigfx_window_t:

### Data Fields

- char ∗ title
- uigfx_widget_t title_widget
- uigfx_widget_t window_widget

## 3.15.1 Detailed Description

**Copyright**

Definition at line 37 of file uigfx_win.h.

## 3.15.2 Field Documentation

### 3.15.2.1 title

```
char* title
```

Definition at line 38 of file uigfx_win.h.

### 3.15.2.2 title_widget

```
uigfx_widget_t title_widget
```

Definition at line 39 of file uigfx_win.h.

#### 3.15.2.3 window_widget

```
uigfx_widget_t window_widget
```

Definition at line 40 of file uigfx_win.h.

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

- myos/ui/uigfx/uigfx_win.h

## 3.16 uileds_state_t Struct Reference

```
#include <uileds.h>
```

### Data Fields

- unsigned lstate:1
- unsigned duration:7

### 3.16.1 Detailed Description

Definition at line 46 of file uileds.h.

### 3.16.2 Field Documentation

#### 3.16.2.1 duration

```
unsigned duration
```

Definition at line 48 of file uileds.h.

#### 3.16.2.2 lstate

```
unsigned lstate
```

Definition at line 47 of file uileds.h.

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

- myos/ui/uileds.h

## 3.17 uileds_t Struct Reference

```
#include <uileds.h>
```

Collaboration diagram for uileds_t:

### Data Fields

- uileds_set_t set_led
- uileds_state_t ∗ pattern
- unsigned pstate:7
- unsigned inverted:1
- unsigned timer:8

### 3.17.1 Detailed Description

Definition at line 53 of file uileds.h.

### 3.17.2 Field Documentation

#### 3.17.2.1 inverted

```
unsigned inverted
```

Definition at line 57 of file uileds.h.

#### 3.17.2.2 pattern

```
uileds_state_t∗ pattern
```

Definition at line 55 of file uileds.h.

#### 3.17.2.3 pstate

```
unsigned pstate
```

Definition at line 56 of file uileds.h.

#### 3.17.2.4 set_led

```
uileds_set_t set_led
```

Definition at line 54 of file uileds.h.

#### 3.17.2.5 timer

```
unsigned timer
```

Definition at line 58 of file uileds.h.

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

- myos/ui/uileds.h

# Chapter 4

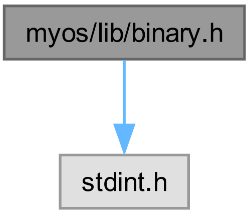
# File Documentation

## 4.1 myos/lib/binary.h File Reference

Provides preprocessor macros to directly code binary integral representations.

```
#include <stdint.h>
```

Include dependency graph for binary.h:



### Macros

- #define BINARY8(val) (((uint8_t)(b##val))&0xFF)
- #define BINARY16(val1, val2) ((((uint16_t)BINARY8(val1))<<8)|((uint16_t)BINARY8(val2)))
- #define BINARY32(val1, val2, val3, val4) ((((uint32_t)BINARY16(val1,val2))<<16)|((uint32_t)BINARY16(val3,val4)))
- #define b0 0x0
- #define b1 0x1
- #define b00 0x0
- #define b01 0x1
- #define b10 0x2
- #define b11 0x3
- #define b000 0x0
- #define b001 0x1
- #define b010 0x2
- #define b011 0x3

- #define b100 0x4
- #define b101 0x5
- #define b110 0x6
- #define b111 0x7
- #define b0000 0x0
- #define b0001 0x1
- #define b0010 0x2
- #define b0011 0x3
- #define b0100 0x4
- #define b0101 0x5
- #define b0110 0x6
- #define b0111 0x7
- #define b1000 0x8
- #define b1001 0x9
- #define b1010 0xA
- #define b1011 0xB
- #define b1100 0xC
- #define b1101 0xD
- #define b1110 0xE
- #define b1111 0xF
- #define b00000 0x0
- #define b00001 0x1
- #define b00010 0x2
- #define b00011 0x3
- #define b00100 0x4
- #define b00101 0x5
- #define b00110 0x6
- #define b00111 0x7
- #define b01000 0x8
- #define b01001 0x9
- #define b01010 0xA
- #define b01011 0xB
- #define b01100 0xC
- #define b01101 0xD
- #define b01110 0xE
- #define b01111 0xF
- #define b10000 0x10
- #define b10001 0x11
- #define b10010 0x12
- #define b10011 0x13
- #define b10100 0x14
- #define b10101 0x15
- #define b10110 0x16
- #define b10111 0x17
- #define b11000 0x18
- #define b11001 0x19
- #define b11010 0x1A
- #define b11011 0x1B
- #define b11100 0x1C
- #define b11101 0x1D
- #define b11110 0x1E
- #define b11111 0x1F
- #define b000000 0x0
- #define b000001 0x1
- #define b000010 0x2

- #define b000011 0x3
- #define b000100 0x4
- #define b000101 0x5
- #define b000110 0x6
- #define b000111 0x7
- #define b001000 0x8
- #define b001001 0x9
- #define b001010 0xA
- #define b001011 0xB
- #define b001100 0xC
- #define b001101 0xD
- #define b001110 0xE
- #define b001111 0xF
- #define b010000 0x10
- #define b010001 0x11
- #define b010010 0x12
- #define b010011 0x13
- #define b010100 0x14
- #define b010101 0x15
- #define b010110 0x16
- #define b010111 0x17
- #define b011000 0x18
- #define b011001 0x19
- #define b011010 0x1A
- #define b011011 0x1B
- #define b011100 0x1C
- #define b011101 0x1D
- #define b011110 0x1E
- #define b011111 0x1F
- #define b100000 0x20
- #define b100001 0x21
- #define b100010 0x22
- #define b100011 0x23
- #define b100100 0x24
- #define b100101 0x25
- #define b100110 0x26
- #define b100111 0x27
- #define b101000 0x28
- #define b101001 0x29
- #define b101010 0x2A
- #define b101011 0x2B
- #define b101100 0x2C
- #define b101101 0x2D
- #define b101110 0x2E
- #define b101111 0x2F
- #define b110000 0x30
- #define b110001 0x31
- #define b110010 0x32
- #define b110011 0x33
- #define b110100 0x34
- #define b110101 0x35
- #define b110110 0x36
- #define b110111 0x37
- #define b111000 0x38
- #define b111001 0x39

- #define b111010 0x3A
- #define b111011 0x3B
- #define b111100 0x3C
- #define b111101 0x3D
- #define b111110 0x3E
- #define b111111 0x3F
- #define b0000000 0x0
- #define b0000001 0x1
- #define b0000010 0x2
- #define b0000011 0x3
- #define b0000100 0x4
- #define b0000101 0x5
- #define b0000110 0x6
- #define b0000111 0x7
- #define b0001000 0x8
- #define b0001001 0x9
- #define b0001010 0xA
- #define b0001011 0xB
- #define b0001100 0xC
- #define b0001101 0xD
- #define b0001110 0xE
- #define b0001111 0xF
- #define b0010000 0x10
- #define b0010001 0x11
- #define b0010010 0x12
- #define b0010011 0x13
- #define b0010100 0x14
- #define b0010101 0x15
- #define b0010110 0x16
- #define b0010111 0x17
- #define b0011000 0x18
- #define b0011001 0x19
- #define b0011010 0x1A
- #define b0011011 0x1B
- #define b0011100 0x1C
- #define b0011101 0x1D
- #define b0011110 0x1E
- #define b0011111 0x1F
- #define b0100000 0x20
- #define b0100001 0x21
- #define b0100010 0x22
- #define b0100011 0x23
- #define b0100100 0x24
- #define b0100101 0x25
- #define b0100110 0x26
- #define b0100111 0x27
- #define b0101000 0x28
- #define b0101001 0x29
- #define b0101010 0x2A
- #define b0101011 0x2B
- #define b0101100 0x2C
- #define b0101101 0x2D
- #define b0101110 0x2E
- #define b0101111 0x2F
- #define b0110000 0x30

- #define b0110001 0x31
- #define b0110010 0x32
- #define b0110011 0x33
- #define b0110100 0x34
- #define b0110101 0x35
- #define b0110110 0x36
- #define b0110111 0x37
- #define b0111000 0x38
- #define b0111001 0x39
- #define b0111010 0x3A
- #define b0111011 0x3B
- #define b0111100 0x3C
- #define b0111101 0x3D
- #define b0111110 0x3E
- #define b0111111 0x3F
- #define b1000000 0x40
- #define b1000001 0x41
- #define b1000010 0x42
- #define b1000011 0x43
- #define b1000100 0x44
- #define b1000101 0x45
- #define b1000110 0x46
- #define b1000111 0x47
- #define b1001000 0x48
- #define b1001001 0x49
- #define b1001010 0x4A
- #define b1001011 0x4B
- #define b1001100 0x4C
- #define b1001101 0x4D
- #define b1001110 0x4E
- #define b1001111 0x4F
- #define b1010000 0x50
- #define b1010001 0x51
- #define b1010010 0x52
- #define b1010011 0x53
- #define b1010100 0x54
- #define b1010101 0x55
- #define b1010110 0x56
- #define b1010111 0x57
- #define b1011000 0x58
- #define b1011001 0x59
- #define b1011010 0x5A
- #define b1011011 0x5B
- #define b1011100 0x5C
- #define b1011101 0x5D
- #define b1011110 0x5E
- #define b1011111 0x5F
- #define b1100000 0x60
- #define b1100001 0x61
- #define b1100010 0x62
- #define b1100011 0x63
- #define b1100100 0x64
- #define b1100101 0x65
- #define b1100110 0x66
- #define b1100111 0x67

- #define b1101000 0x68
- #define b1101001 0x69
- #define b1101010 0x6A
- #define b1101011 0x6B
- #define b1101100 0x6C
- #define b1101101 0x6D
- #define b1101110 0x6E
- #define b1101111 0x6F
- #define b1110000 0x70
- #define b1110001 0x71
- #define b1110010 0x72
- #define b1110011 0x73
- #define b1110100 0x74
- #define b1110101 0x75
- #define b1110110 0x76
- #define b1110111 0x77
- #define b1111000 0x78
- #define b1111001 0x79
- #define b1111010 0x7A
- #define b1111011 0x7B
- #define b1111100 0x7C
- #define b1111101 0x7D
- #define b1111110 0x7E
- #define b1111111 0x7F
- #define b00000000 0x00
- #define b00000001 0x01
- #define b00000010 0x02
- #define b00000011 0x03
- #define b00000100 0x04
- #define b00000101 0x05
- #define b00000110 0x06
- #define b00000111 0x07
- #define b00001000 0x08
- #define b00001001 0x09
- #define b00001010 0x0A
- #define b00001011 0x0B
- #define b00001100 0x0C
- #define b00001101 0x0D
- #define b00001110 0x0E
- #define b00001111 0x0F
- #define b00010000 0x10
- #define b00010001 0x11
- #define b00010010 0x12
- #define b00010011 0x13
- #define b00010100 0x14
- #define b00010101 0x15
- #define b00010110 0x16
- #define b00010111 0x17
- #define b00011000 0x18
- #define b00011001 0x19
- #define b00011010 0x1A
- #define b00011011 0x1B
- #define b00011100 0x1C
- #define b00011101 0x1D
- #define b00011110 0x1E

- #define b00011111 0x1F
- #define b00100000 0x20
- #define b00100001 0x21
- #define b00100010 0x22
- #define b00100011 0x23
- #define b00100100 0x24
- #define b00100101 0x25
- #define b00100110 0x26
- #define b00100111 0x27
- #define b00101000 0x28
- #define b00101001 0x29
- #define b00101010 0x2A
- #define b00101011 0x2B
- #define b00101100 0x2C
- #define b00101101 0x2D
- #define b00101110 0x2E
- #define b00101111 0x2F
- #define b00110000 0x30
- #define b00110001 0x31
- #define b00110010 0x32
- #define b00110011 0x33
- #define b00110100 0x34
- #define b00110101 0x35
- #define b00110110 0x36
- #define b00110111 0x37
- #define b00111000 0x38
- #define b00111001 0x39
- #define b00111010 0x3A
- #define b00111011 0x3B
- #define b00111100 0x3C
- #define b00111101 0x3D
- #define b00111110 0x3E
- #define b00111111 0x3F
- #define b01000000 0x40
- #define b01000001 0x41
- #define b01000010 0x42
- #define b01000011 0x43
- #define b01000100 0x44
- #define b01000101 0x45
- #define b01000110 0x46
- #define b01000111 0x47
- #define b01001000 0x48
- #define b01001001 0x49
- #define b01001010 0x4A
- #define b01001011 0x4B
- #define b01001100 0x4C
- #define b01001101 0x4D
- #define b01001110 0x4E
- #define b01001111 0x4F
- #define b01010000 0x50
- #define b01010001 0x51
- #define b01010010 0x52
- #define b01010011 0x53
- #define b01010100 0x54
- #define b01010101 0x55

- #define b01010110 0x56
- #define b01010111 0x57
- #define b01011000 0x58
- #define b01011001 0x59
- #define b01011010 0x5A
- #define b01011011 0x5B
- #define b01011100 0x5C
- #define b01011101 0x5D
- #define b01011110 0x5E
- #define b01011111 0x5F
- #define b01100000 0x60
- #define b01100001 0x61
- #define b01100010 0x62
- #define b01100011 0x63
- #define b01100100 0x64
- #define b01100101 0x65
- #define b01100110 0x66
- #define b01100111 0x67
- #define b01101000 0x68
- #define b01101001 0x69
- #define b01101010 0x6A
- #define b01101011 0x6B
- #define b01101100 0x6C
- #define b01101101 0x6D
- #define b01101110 0x6E
- #define b01101111 0x6F
- #define b01110000 0x70
- #define b01110001 0x71
- #define b01110010 0x72
- #define b01110011 0x73
- #define b01110100 0x74
- #define b01110101 0x75
- #define b01110110 0x76
- #define b01110111 0x77
- #define b01111000 0x78
- #define b01111001 0x79
- #define b01111010 0x7A
- #define b01111011 0x7B
- #define b01111100 0x7C
- #define b01111101 0x7D
- #define b01111110 0x7E
- #define b01111111 0x7F
- #define b10000000 0x80
- #define b10000001 0x81
- #define b10000010 0x82
- #define b10000011 0x83
- #define b10000100 0x84
- #define b10000101 0x85
- #define b10000110 0x86
- #define b10000111 0x87
- #define b10001000 0x88
- #define b10001001 0x89
- #define b10001010 0x8A
- #define b10001011 0x8B
- #define b10001100 0x8C

- #define b10001101 0x8D
- #define b10001110 0x8E
- #define b10001111 0x8F
- #define b10010000 0x90
- #define b10010001 0x91
- #define b10010010 0x92
- #define b10010011 0x93
- #define b10010100 0x94
- #define b10010101 0x95
- #define b10010110 0x96
- #define b10010111 0x97
- #define b10011000 0x98
- #define b10011001 0x99
- #define b10011010 0x9A
- #define b10011011 0x9B
- #define b10011100 0x9C
- #define b10011101 0x9D
- #define b10011110 0x9E
- #define b10011111 0x9F
- #define b10100000 0xA0
- #define b10100001 0xA1
- #define b10100010 0xA2
- #define b10100011 0xA3
- #define b10100100 0xA4
- #define b10100101 0xA5
- #define b10100110 0xA6
- #define b10100111 0xA7
- #define b10101000 0xA8
- #define b10101001 0xA9
- #define b10101010 0xAA
- #define b10101011 0xAB
- #define b10101100 0xAC
- #define b10101101 0xAD
- #define b10101110 0xAE
- #define b10101111 0xAF
- #define b10110000 0xB0
- #define b10110001 0xB1
- #define b10110010 0xB2
- #define b10110011 0xB3
- #define b10110100 0xB4
- #define b10110101 0xB5
- #define b10110110 0xB6
- #define b10110111 0xB7
- #define b10111000 0xB8
- #define b10111001 0xB9
- #define b10111010 0xBA
- #define b10111011 0xBB
- #define b10111100 0xBC
- #define b10111101 0xBD
- #define b10111110 0xBE
- #define b10111111 0xBF
- #define b11000000 0xC0
- #define b11000001 0xC1
- #define b11000010 0xC2
- #define b11000011 0xC3

- #define b11000100 0xC4
- #define b11000101 0xC5
- #define b11000110 0xC6
- #define b11000111 0xC7
- #define b11001000 0xC8
- #define b11001001 0xC9
- #define b11001010 0xCA
- #define b11001011 0xCB
- #define b11001100 0xCC
- #define b11001101 0xCD
- #define b11001110 0xCE
- #define b11001111 0xCF
- #define b11010000 0xD0
- #define b11010001 0xD1
- #define b11010010 0xD2
- #define b11010011 0xD3
- #define b11010100 0xD4
- #define b11010101 0xD5
- #define b11010110 0xD6
- #define b11010111 0xD7
- #define b11011000 0xD8
- #define b11011001 0xD9
- #define b11011010 0xDA
- #define b11011011 0xDB
- #define b11011100 0xDC
- #define b11011101 0xDD
- #define b11011110 0xDE
- #define b11011111 0xDF
- #define b11100000 0xE0
- #define b11100001 0xE1
- #define b11100010 0xE2
- #define b11100011 0xE3
- #define b11100100 0xE4
- #define b11100101 0xE5
- #define b11100110 0xE6
- #define b11100111 0xE7
- #define b11101000 0xE8
- #define b11101001 0xE9
- #define b11101010 0xEA
- #define b11101011 0xEB
- #define b11101100 0xEC
- #define b11101101 0xED
- #define b11101110 0xEE
- #define b11101111 0xEF
- #define b11110000 0xF0
- #define b11110001 0xF1
- #define b11110010 0xF2
- #define b11110011 0xF3
- #define b11110100 0xF4
- #define b11110101 0xF5
- #define b11110110 0xF6
- #define b11110111 0xF7
- #define b11111000 0xF8
- #define b11111001 0xF9
- #define b11111010 0xFA

- #define b11111011 0xFB
- #define b11111100 0xFC
- #define b11111101 0xFD
- #define b11111110 0xFE
- #define b11111111 0xFF

### 4.1.1 Detailed Description

Provides preprocessor macros to directly code binary integral representations.

**Copyright**

### 4.1.2 Macro Definition Documentation

#### 4.1.2.1 b0

```
#define b0 0x0
```

Definition at line 52 of file binary.h.

#### 4.1.2.2 b00

```
#define b00 0x0
```

Definition at line 56 of file binary.h.

#### 4.1.2.3 b000

```
#define b000 0x0
```

Definition at line 62 of file binary.h.

#### 4.1.2.4 b0000

```
#define b0000 0x0
```

Definition at line 72 of file binary.h.

#### 4.1.2.5 b00000

```
#define b00000 0x0
```

Definition at line 90 of file binary.h.

#### 4.1.2.6 b000000

```
#define b000000 0x0
```

Definition at line 125 of file binary.h.

#### 4.1.2.7 b0000000

```
#define b0000000 0x0
```

Definition at line 192 of file binary.h.

#### 4.1.2.8 b00000000

```
#define b00000000 0x00
```

Definition at line 322 of file binary.h.

#### 4.1.2.9 b00000001

```
#define b00000001 0x01
```

Definition at line 323 of file binary.h.

#### 4.1.2.10 b0000001

```
#define b0000001 0x1
```

Definition at line 193 of file binary.h.

#### 4.1.2.11 b00000010

```
#define b00000010 0x02
```

Definition at line 324 of file binary.h.

#### 4.1.2.12 b00000011

```
#define b00000011 0x03
```

Definition at line 325 of file binary.h.

#### 4.1.2.13 b000001

```
#define b000001 0x1
```

Definition at line 126 of file binary.h.

#### 4.1.2.14 b0000010

```
#define b0000010 0x2
```

Definition at line 194 of file binary.h.

#### 4.1.2.15 b00000100

```
#define b00000100 0x04
```

Definition at line 326 of file binary.h.

#### 4.1.2.16 b00000101

```
#define b00000101 0x05
```

Definition at line 327 of file binary.h.

#### 4.1.2.17 b0000011

```
#define b0000011 0x3
```

Definition at line 195 of file binary.h.

#### 4.1.2.18 b00000110

```
#define b00000110 0x06
```

Definition at line 328 of file binary.h.

#### 4.1.2.19 b00000111

```
#define b00000111 0x07
```

Definition at line 329 of file binary.h.

#### 4.1.2.20 b00001

```
#define b00001 0x1
```

Definition at line 91 of file binary.h.

#### 4.1.2.21 b000010

```
#define b000010 0x2
```

Definition at line 127 of file binary.h.

#### 4.1.2.22 b0000100

```
#define b0000100 0x4
```

Definition at line 196 of file binary.h.

#### 4.1.2.23 b00001000

```
#define b00001000 0x08
```

Definition at line 330 of file binary.h.

#### 4.1.2.24 b00001001

```
#define b00001001 0x09
```

Definition at line 331 of file binary.h.

#### 4.1.2.25 b0000101

```
#define b0000101 0x5
```

Definition at line 197 of file binary.h.

#### 4.1.2.26 b00001010

```
#define b00001010 0x0A
```

Definition at line 332 of file binary.h.

#### 4.1.2.27 b00001011

```
#define b00001011 0x0B
```

Definition at line 333 of file binary.h.

#### 4.1.2.28 b000011

```
#define b000011 0x3
```

Definition at line 128 of file binary.h.

#### 4.1.2.29 b0000110

```
#define b0000110 0x6
```

Definition at line 198 of file binary.h.

#### 4.1.2.30 b00001100

```
#define b00001100 0x0C
```

Definition at line 334 of file binary.h.

#### 4.1.2.31 b00001101

```
#define b00001101 0x0D
```

Definition at line 335 of file binary.h.

#### 4.1.2.32 b0000111

```
#define b0000111 0x7
```

Definition at line 199 of file binary.h.

#### 4.1.2.33 b00001110

```
#define b00001110 0x0E
```

Definition at line 336 of file binary.h.

#### 4.1.2.34 b00001111

```
#define b00001111 0x0F
```

Definition at line 337 of file binary.h.

#### 4.1.2.35 b0001

```
#define b0001 0x1
```

Definition at line 73 of file binary.h.

#### 4.1.2.36 b00010

```
#define b00010 0x2
```

Definition at line 92 of file binary.h.

#### 4.1.2.37 b000100

```
#define b000100 0x4
```

Definition at line 129 of file binary.h.

#### 4.1.2.38 b0001000

```
#define b0001000 0x8
```

Definition at line 200 of file binary.h.

#### 4.1.2.39 b00010000

```
#define b00010000 0x10
```

Definition at line 338 of file binary.h.

#### 4.1.2.40 b00010001

```
#define b00010001 0x11
```

Definition at line 339 of file binary.h.

#### 4.1.2.41 b0001001

```
#define b0001001 0x9
```

Definition at line 201 of file binary.h.

#### 4.1.2.42 b00010010

```
#define b00010010 0x12
```

Definition at line 340 of file binary.h.

#### 4.1.2.43 b00010011

```
#define b00010011 0x13
```

Definition at line 341 of file binary.h.

#### 4.1.2.44 b000101

```
#define b000101 0x5
```

Definition at line 130 of file binary.h.

#### 4.1.2.45 b0001010

```
#define b0001010 0xA
```

Definition at line 202 of file binary.h.

#### 4.1.2.46 b00010100

```
#define b00010100 0x14
```

Definition at line 342 of file binary.h.

#### 4.1.2.47 b00010101

```
#define b00010101 0x15
```

Definition at line 343 of file binary.h.

#### 4.1.2.48 b0001011

```
#define b0001011 0xB
```

Definition at line 203 of file binary.h.

#### 4.1.2.49 b00010110

```
#define b00010110 0x16
```

Definition at line 344 of file binary.h.

#### 4.1.2.50 b00010111

```
#define b00010111 0x17
```

Definition at line 345 of file binary.h.

#### 4.1.2.51 b00011

```
#define b00011 0x3
```

Definition at line 93 of file binary.h.

#### 4.1.2.52 b000110

```
#define b000110 0x6
```

Definition at line 131 of file binary.h.

#### 4.1.2.53 b0001100

```
#define b0001100 0xC
```

Definition at line 204 of file binary.h.

#### 4.1.2.54 b00011000

```
#define b00011000 0x18
```

Definition at line 346 of file binary.h.

#### 4.1.2.55 b00011001

```
#define b00011001 0x19
```

Definition at line 347 of file binary.h.

#### 4.1.2.56 b0001101

```
#define b0001101 0xD
```

Definition at line 205 of file binary.h.

#### 4.1.2.57 b00011010

```
#define b00011010 0x1A
```

Definition at line 348 of file binary.h.

#### 4.1.2.58 b00011011

```
#define b00011011 0x1B
```

Definition at line 349 of file binary.h.

#### 4.1.2.59 b000111

```
#define b000111 0x7
```

Definition at line 132 of file binary.h.

#### 4.1.2.60 b0001110

```
#define b0001110 0xE
```

Definition at line 206 of file binary.h.

#### 4.1.2.61 b00011100

```
#define b00011100 0x1C
```

Definition at line 350 of file binary.h.

#### 4.1.2.62 b00011101

```
#define b00011101 0x1D
```

Definition at line 351 of file binary.h.

#### 4.1.2.63 b0001111

```
#define b0001111 0xF
```

Definition at line 207 of file binary.h.

#### 4.1.2.64 b00011110

```
#define b00011110 0x1E
```

Definition at line 352 of file binary.h.

#### 4.1.2.65 b00011111

```
#define b00011111 0x1F
```

Definition at line 353 of file binary.h.

#### 4.1.2.66 b001

```
#define b001 0x1
```

Definition at line 63 of file binary.h.

#### 4.1.2.67 b0010

```
#define b0010 0x2
```

Definition at line 74 of file binary.h.

#### 4.1.2.68 b00100

```
#define b00100 0x4
```

Definition at line 94 of file binary.h.

#### 4.1.2.69 b001000

```
#define b001000 0x8
```

Definition at line 133 of file binary.h.

#### 4.1.2.70 b0010000

```
#define b0010000 0x10
```

Definition at line 208 of file binary.h.

#### 4.1.2.71 b00100000

```
#define b00100000 0x20
```

Definition at line 354 of file binary.h.

#### 4.1.2.72 b00100001

```
#define b00100001 0x21
```

Definition at line 355 of file binary.h.

#### 4.1.2.73 b0010001

```
#define b0010001 0x11
```

Definition at line 209 of file binary.h.

#### 4.1.2.74 b00100010

```
#define b00100010 0x22
```

Definition at line 356 of file binary.h.

#### 4.1.2.75 b00100011

```
#define b00100011 0x23
```

Definition at line 357 of file binary.h.

#### 4.1.2.76 b001001

```
#define b001001 0x9
```

Definition at line 134 of file binary.h.

#### 4.1.2.77 b0010010

```
#define b0010010 0x12
```

Definition at line 210 of file binary.h.

#### 4.1.2.78 b00100100

```
#define b00100100 0x24
```

Definition at line 358 of file binary.h.

#### 4.1.2.79 b00100101

```
#define b00100101 0x25
```

Definition at line 359 of file binary.h.

#### 4.1.2.80 b0010011

```
#define b0010011 0x13
```

Definition at line 211 of file binary.h.

#### 4.1.2.81 b00100110

```
#define b00100110 0x26
```

Definition at line 360 of file binary.h.

#### 4.1.2.82 b00100111

```
#define b00100111 0x27
```

Definition at line 361 of file binary.h.

#### 4.1.2.83 b00101

```
#define b00101 0x5
```

Definition at line 95 of file binary.h.

#### 4.1.2.84 b001010

```
#define b001010 0xA
```

Definition at line 135 of file binary.h.

#### 4.1.2.85 b0010100

```
#define b0010100 0x14
```

Definition at line 212 of file binary.h.

#### 4.1.2.86 b00101000

```
#define b00101000 0x28
```

Definition at line 362 of file binary.h.

#### 4.1.2.87 b00101001

```
#define b00101001 0x29
```

Definition at line 363 of file binary.h.

#### 4.1.2.88 b0010101

```
#define b0010101 0x15
```

Definition at line 213 of file binary.h.

#### 4.1.2.89 b00101010

```
#define b00101010 0x2A
```

Definition at line 364 of file binary.h.

#### 4.1.2.90 b00101011

```
#define b00101011 0x2B
```

Definition at line 365 of file binary.h.

#### 4.1.2.91 b001011

```
#define b001011 0xB
```

Definition at line 136 of file binary.h.

#### 4.1.2.92 b0010110

```
#define b0010110 0x16
```

Definition at line 214 of file binary.h.

#### 4.1.2.93 b00101100

```
#define b00101100 0x2C
```

Definition at line 366 of file binary.h.

#### 4.1.2.94 b00101101

```
#define b00101101 0x2D
```

Definition at line 367 of file binary.h.

#### 4.1.2.95 b0010111

```
#define b0010111 0x17
```

Definition at line 215 of file binary.h.

#### 4.1.2.96 b00101110

```
#define b00101110 0x2E
```

Definition at line 368 of file binary.h.

#### 4.1.2.97 b00101111

```
#define b00101111 0x2F
```

Definition at line 369 of file binary.h.

#### 4.1.2.98 b0011

```
#define b0011 0x3
```

Definition at line 75 of file binary.h.

#### 4.1.2.99 b00110

```
#define b00110 0x6
```

Definition at line 96 of file binary.h.

#### 4.1.2.100 b001100

```
#define b001100 0xC
```

Definition at line 137 of file binary.h.

#### 4.1.2.101 b0011000

```
#define b0011000 0x18
```

Definition at line 216 of file binary.h.

#### 4.1.2.102 b00110000

```
#define b00110000 0x30
```

Definition at line 370 of file binary.h.

#### 4.1.2.103 b00110001

```
#define b00110001 0x31
```

Definition at line 371 of file binary.h.

#### 4.1.2.104 b0011001

```
#define b0011001 0x19
```

Definition at line 217 of file binary.h.

#### 4.1.2.105 b00110010

```
#define b00110010 0x32
```

Definition at line 372 of file binary.h.

#### 4.1.2.106 b00110011

```
#define b00110011 0x33
```

Definition at line 373 of file binary.h.

#### 4.1.2.107 b001101

```
#define b001101 0xD
```

Definition at line 138 of file binary.h.

#### 4.1.2.108 b0011010

```
#define b0011010 0x1A
```

Definition at line 218 of file binary.h.

#### 4.1.2.109 b00110100

```
#define b00110100 0x34
```

Definition at line 374 of file binary.h.

#### 4.1.2.110 b00110101

```
#define b00110101 0x35
```

Definition at line 375 of file binary.h.

#### 4.1.2.111 b0011011

```
#define b0011011 0x1B
```

Definition at line 219 of file binary.h.

#### 4.1.2.112 b00110110

```
#define b00110110 0x36
```

Definition at line 376 of file binary.h.

#### 4.1.2.113 b00110111

```
#define b00110111 0x37
```

Definition at line 377 of file binary.h.

#### 4.1.2.114 b00111

```
#define b00111 0x7
```

Definition at line 97 of file binary.h.

#### 4.1.2.115 b001110

```
#define b001110 0xE
```

Definition at line 139 of file binary.h.

#### 4.1.2.116 b0011100

```
#define b0011100 0x1C
```

Definition at line 220 of file binary.h.

#### 4.1.2.117 b00111000

```
#define b00111000 0x38
```

Definition at line 378 of file binary.h.

#### 4.1.2.118 b00111001

```
#define b00111001 0x39
```

Definition at line 379 of file binary.h.

#### 4.1.2.119 b0011101

```
#define b0011101 0x1D
```

Definition at line 221 of file binary.h.

#### 4.1.2.120 b00111010

```
#define b00111010 0x3A
```

Definition at line 380 of file binary.h.

#### 4.1.2.121 b00111011

```
#define b00111011 0x3B
```

Definition at line 381 of file binary.h.

#### 4.1.2.122 b001111

```
#define b001111 0xF
```

Definition at line 140 of file binary.h.

#### 4.1.2.123 b0011110

```
#define b0011110 0x1E
```

Definition at line 222 of file binary.h.

#### 4.1.2.124 b00111100

```
#define b00111100 0x3C
```

Definition at line 382 of file binary.h.

#### 4.1.2.125 b00111101

```
#define b00111101 0x3D
```

Definition at line 383 of file binary.h.

#### 4.1.2.126 b0011111

```
#define b0011111 0x1F
```

Definition at line 223 of file binary.h.

#### 4.1.2.127 b00111110

```
#define b00111110 0x3E
```

Definition at line 384 of file binary.h.

#### 4.1.2.128 b00111111

```
#define b00111111 0x3F
```

Definition at line 385 of file binary.h.

#### 4.1.2.129 b01

```
#define b01 0x1
```

Definition at line 57 of file binary.h.

#### 4.1.2.130 b010

```
#define b010 0x2
```

Definition at line 64 of file binary.h.

#### 4.1.2.131 b0100

```
#define b0100 0x4
```

Definition at line 76 of file binary.h.

#### 4.1.2.132 b01000

```
#define b01000 0x8
```

Definition at line 98 of file binary.h.

#### 4.1.2.133 b010000

```
#define b010000 0x10
```

Definition at line 141 of file binary.h.

#### 4.1.2.134 b0100000

```
#define b0100000 0x20
```

Definition at line 224 of file binary.h.

#### 4.1.2.135 b01000000

```
#define b01000000 0x40
```

Definition at line 386 of file binary.h.

#### 4.1.2.136 b01000001

```
#define b01000001 0x41
```

Definition at line 387 of file binary.h.

#### 4.1.2.137 b0100001

```
#define b0100001 0x21
```

Definition at line 225 of file binary.h.

#### 4.1.2.138 b01000010

```
#define b01000010 0x42
```

Definition at line 388 of file binary.h.

#### 4.1.2.139 b01000011

```
#define b01000011 0x43
```

Definition at line 389 of file binary.h.

#### 4.1.2.140 b010001

```
#define b010001 0x11
```

Definition at line 142 of file binary.h.

#### 4.1.2.141 b0100010

```
#define b0100010 0x22
```

Definition at line 226 of file binary.h.

#### 4.1.2.142 b01000100

```
#define b01000100 0x44
```

Definition at line 390 of file binary.h.

#### 4.1.2.143 b01000101

```
#define b01000101 0x45
```

Definition at line 391 of file binary.h.

#### 4.1.2.144 b0100011

```
#define b0100011 0x23
```

Definition at line 227 of file binary.h.

#### 4.1.2.145 b01000110

```
#define b01000110 0x46
```

Definition at line 392 of file binary.h.

#### 4.1.2.146 b01000111

```
#define b01000111 0x47
```

Definition at line 393 of file binary.h.

#### 4.1.2.147 b01001

```
#define b01001 0x9
```

Definition at line 99 of file binary.h.

#### 4.1.2.148 b010010

```
#define b010010 0x12
```

Definition at line 143 of file binary.h.

#### 4.1.2.149 b0100100

```
#define b0100100 0x24
```

Definition at line 228 of file binary.h.

#### 4.1.2.150 b01001000

```
#define b01001000 0x48
```

Definition at line 394 of file binary.h.

#### 4.1.2.151 b01001001

```
#define b01001001 0x49
```

Definition at line 395 of file binary.h.

#### 4.1.2.152 b0100101

```
#define b0100101 0x25
```

Definition at line 229 of file binary.h.

#### 4.1.2.153 b01001010

```
#define b01001010 0x4A
```

Definition at line 396 of file binary.h.

#### 4.1.2.154 b01001011

```
#define b01001011 0x4B
```

Definition at line 397 of file binary.h.

#### 4.1.2.155 b010011

```
#define b010011 0x13
```

Definition at line 144 of file binary.h.

#### 4.1.2.156 b0100110

```
#define b0100110 0x26
```

Definition at line 230 of file binary.h.

#### 4.1.2.157 b01001100

```
#define b01001100 0x4C
```

Definition at line 398 of file binary.h.

#### 4.1.2.158 b01001101

```
#define b01001101 0x4D
```

Definition at line 399 of file binary.h.

#### 4.1.2.159 b0100111

```
#define b0100111 0x27
```

Definition at line 231 of file binary.h.

#### 4.1.2.160 b01001110

```
#define b01001110 0x4E
```

Definition at line 400 of file binary.h.

#### 4.1.2.161 b01001111

```
#define b01001111 0x4F
```

Definition at line 401 of file binary.h.

#### 4.1.2.162 b0101

```
#define b0101 0x5
```

Definition at line 77 of file binary.h.

#### 4.1.2.163 b01010

```
#define b01010 0xA
```

Definition at line 100 of file binary.h.

#### 4.1.2.164 b010100

```
#define b010100 0x14
```

Definition at line 145 of file binary.h.

#### 4.1.2.165 b0101000

```
#define b0101000 0x28
```

Definition at line 232 of file binary.h.

#### 4.1.2.166 b01010000

```
#define b01010000 0x50
```

Definition at line 402 of file binary.h.

#### 4.1.2.167 b01010001

```
#define b01010001 0x51
```

Definition at line 403 of file binary.h.

#### 4.1.2.168 b0101001

```
#define b0101001 0x29
```

Definition at line 233 of file binary.h.

#### 4.1.2.169 b01010010

```
#define b01010010 0x52
```

Definition at line 404 of file binary.h.

#### 4.1.2.170 b01010011

```
#define b01010011 0x53
```

Definition at line 405 of file binary.h.

#### 4.1.2.171 b010101

```
#define b010101 0x15
```

Definition at line 146 of file binary.h.

#### 4.1.2.172 b0101010

```
#define b0101010 0x2A
```

Definition at line 234 of file binary.h.

#### 4.1.2.173 b01010100

```
#define b01010100 0x54
```

Definition at line 406 of file binary.h.

#### 4.1.2.174 b01010101

```
#define b01010101 0x55
```

Definition at line 407 of file binary.h.

#### 4.1.2.175 b0101011

```
#define b0101011 0x2B
```

Definition at line 235 of file binary.h.

#### 4.1.2.176 b01010110

```
#define b01010110 0x56
```

Definition at line 408 of file binary.h.

#### 4.1.2.177 b01010111

```
#define b01010111 0x57
```

Definition at line 409 of file binary.h.

#### 4.1.2.178 b01011

```
#define b01011 0xB
```

Definition at line 101 of file binary.h.

#### 4.1.2.179 b010110

```
#define b010110 0x16
```

Definition at line 147 of file binary.h.

#### 4.1.2.180 b0101100

```
#define b0101100 0x2C
```

Definition at line 236 of file binary.h.

#### 4.1.2.181 b01011000

```
#define b01011000 0x58
```

Definition at line 410 of file binary.h.

#### 4.1.2.182 b01011001

```
#define b01011001 0x59
```

Definition at line 411 of file binary.h.

#### 4.1.2.183 b0101101

```
#define b0101101 0x2D
```

Definition at line 237 of file binary.h.

#### 4.1.2.184 b01011010

```
#define b01011010 0x5A
```

Definition at line 412 of file binary.h.

#### 4.1.2.185 b01011011

```
#define b01011011 0x5B
```

Definition at line 413 of file binary.h.

#### 4.1.2.186 b010111

```
#define b010111 0x17
```

Definition at line 148 of file binary.h.

#### 4.1.2.187 b0101110

```
#define b0101110 0x2E
```

Definition at line 238 of file binary.h.

#### 4.1.2.188 b01011100

```
#define b01011100 0x5C
```

Definition at line 414 of file binary.h.

#### 4.1.2.189 b01011101

```
#define b01011101 0x5D
```

Definition at line 415 of file binary.h.

#### 4.1.2.190 b0101111

```
#define b0101111 0x2F
```

Definition at line 239 of file binary.h.

#### 4.1.2.191 b01011110

```
#define b01011110 0x5E
```

Definition at line 416 of file binary.h.

#### 4.1.2.192 b01011111

```
#define b01011111 0x5F
```

Definition at line 417 of file binary.h.

#### 4.1.2.193 b011

```
#define b011 0x3
```

Definition at line 65 of file binary.h.

#### 4.1.2.194 b0110

```
#define b0110 0x6
```

Definition at line 78 of file binary.h.

#### 4.1.2.195 b01100

```
#define b01100 0xC
```

Definition at line 102 of file binary.h.

#### 4.1.2.196 b011000

```
#define b011000 0x18
```

Definition at line 149 of file binary.h.

#### 4.1.2.197 b0110000

```
#define b0110000 0x30
```

Definition at line 240 of file binary.h.

#### 4.1.2.198 b01100000

```
#define b01100000 0x60
```

Definition at line 418 of file binary.h.

#### 4.1.2.199 b01100001

```
#define b01100001 0x61
```

Definition at line 419 of file binary.h.

#### 4.1.2.200 b0110001

```
#define b0110001 0x31
```

Definition at line 241 of file binary.h.

#### 4.1.2.201 b01100010

```
#define b01100010 0x62
```

Definition at line 420 of file binary.h.

#### 4.1.2.202 b01100011

```
#define b01100011 0x63
```

Definition at line 421 of file binary.h.

#### 4.1.2.203 b011001

```
#define b011001 0x19
```

Definition at line 150 of file binary.h.

#### 4.1.2.204 b0110010

```
#define b0110010 0x32
```

Definition at line 242 of file binary.h.

#### 4.1.2.205 b01100100

```
#define b01100100 0x64
```

Definition at line 422 of file binary.h.

#### 4.1.2.206 b01100101

```
#define b01100101 0x65
```

Definition at line 423 of file binary.h.

#### 4.1.2.207 b0110011

```
#define b0110011 0x33
```

Definition at line 243 of file binary.h.

#### 4.1.2.208 b01100110

```
#define b01100110 0x66
```

Definition at line 424 of file binary.h.

#### 4.1.2.209 b01100111

```
#define b01100111 0x67
```

Definition at line 425 of file binary.h.

#### 4.1.2.210 b01101

```
#define b01101 0xD
```

Definition at line 103 of file binary.h.

#### 4.1.2.211 b011010

```
#define b011010 0x1A
```

Definition at line 151 of file binary.h.

#### 4.1.2.212 b0110100

```
#define b0110100 0x34
```

Definition at line 244 of file binary.h.

#### 4.1.2.213 b01101000

```
#define b01101000 0x68
```

Definition at line 426 of file binary.h.

#### 4.1.2.214 b01101001

```
#define b01101001 0x69
```

Definition at line 427 of file binary.h.

#### 4.1.2.215 b0110101

```
#define b0110101 0x35
```

Definition at line 245 of file binary.h.

#### 4.1.2.216 b01101010

```
#define b01101010 0x6A
```

Definition at line 428 of file binary.h.

#### 4.1.2.217 b01101011

```
#define b01101011 0x6B
```

Definition at line 429 of file binary.h.

#### 4.1.2.218 b011011

```
#define b011011 0x1B
```

Definition at line 152 of file binary.h.

#### 4.1.2.219 b0110110

```
#define b0110110 0x36
```

Definition at line 246 of file binary.h.

#### 4.1.2.220 b01101100

```
#define b01101100 0x6C
```

Definition at line 430 of file binary.h.

#### 4.1.2.221 b01101101

```
#define b01101101 0x6D
```

Definition at line 431 of file binary.h.

#### 4.1.2.222 b0110111

```
#define b0110111 0x37
```

Definition at line 247 of file binary.h.

#### 4.1.2.223 b01101110

```
#define b01101110 0x6E
```

Definition at line 432 of file binary.h.

#### 4.1.2.224 b01101111

```
#define b01101111 0x6F
```

Definition at line 433 of file binary.h.

#### 4.1.2.225 b0111

```
#define b0111 0x7
```

Definition at line 79 of file binary.h.

#### 4.1.2.226 b01110

```
#define b01110 0xE
```

Definition at line 104 of file binary.h.

#### 4.1.2.227 b011100

```
#define b011100 0x1C
```

Definition at line 153 of file binary.h.

#### 4.1.2.228 b0111000

```
#define b0111000 0x38
```

Definition at line 248 of file binary.h.

#### 4.1.2.229 b01110000

```
#define b01110000 0x70
```

Definition at line 434 of file binary.h.

#### 4.1.2.230 b01110001

```
#define b01110001 0x71
```

Definition at line 435 of file binary.h.

#### 4.1.2.231 b0111001

```
#define b0111001 0x39
```

Definition at line 249 of file binary.h.

#### 4.1.2.232 b01110010

```
#define b01110010 0x72
```

Definition at line 436 of file binary.h.

#### 4.1.2.233 b01110011

```
#define b01110011 0x73
```

Definition at line 437 of file binary.h.

#### 4.1.2.234 b011101

```
#define b011101 0x1D
```

Definition at line 154 of file binary.h.

#### 4.1.2.235 b0111010

```
#define b0111010 0x3A
```

Definition at line 250 of file binary.h.

#### 4.1.2.236 b01110100

```
#define b01110100 0x74
```

Definition at line 438 of file binary.h.

#### 4.1.2.237 b01110101

```
#define b01110101 0x75
```

Definition at line 439 of file binary.h.

#### 4.1.2.238 b0111011

```
#define b0111011 0x3B
```

Definition at line 251 of file binary.h.

#### 4.1.2.239 b01110110

```
#define b01110110 0x76
```

Definition at line 440 of file binary.h.

#### 4.1.2.240 b01110111

```
#define b01110111 0x77
```

Definition at line 441 of file binary.h.

#### 4.1.2.241 b01111

```
#define b01111 0xF
```

Definition at line 105 of file binary.h.

#### 4.1.2.242 b011110

```
#define b011110 0x1E
```

Definition at line 155 of file binary.h.

#### 4.1.2.243 b0111100

```
#define b0111100 0x3C
```

Definition at line 252 of file binary.h.

#### 4.1.2.244 b01111000

```
#define b01111000 0x78
```

Definition at line 442 of file binary.h.

#### 4.1.2.245 b01111001

```
#define b01111001 0x79
```

Definition at line 443 of file binary.h.

#### 4.1.2.246 b0111101

```
#define b0111101 0x3D
```

Definition at line 253 of file binary.h.

#### 4.1.2.247 b01111010

```
#define b01111010 0x7A
```

Definition at line 444 of file binary.h.

#### 4.1.2.248 b01111011

```
#define b01111011 0x7B
```

Definition at line 445 of file binary.h.

#### 4.1.2.249 b011111

```
#define b011111 0x1F
```

Definition at line 156 of file binary.h.

#### 4.1.2.250 b0111110

```
#define b0111110 0x3E
```

Definition at line 254 of file binary.h.

#### 4.1.2.251 b01111100

```
#define b01111100 0x7C
```

Definition at line 446 of file binary.h.

#### 4.1.2.252 b01111101

```
#define b01111101 0x7D
```

Definition at line 447 of file binary.h.

#### 4.1.2.253 b0111111

```
#define b0111111 0x3F
```

Definition at line 255 of file binary.h.

#### 4.1.2.254 b01111110

```
#define b01111110 0x7E
```

Definition at line 448 of file binary.h.

#### 4.1.2.255 b01111111

```
#define b01111111 0x7F
```

Definition at line 449 of file binary.h.

#### 4.1.2.256 b1

```
#define b1 0x1
```

Definition at line 53 of file binary.h.

#### 4.1.2.257 b10

```
#define b10 0x2
```

Definition at line 58 of file binary.h.

#### 4.1.2.258 b100

```
#define b100 0x4
```

Definition at line 66 of file binary.h.

#### 4.1.2.259 b1000

```
#define b1000 0x8
```

Definition at line 80 of file binary.h.

#### 4.1.2.260 b10000

```
#define b10000 0x10
```

Definition at line 106 of file binary.h.

#### 4.1.2.261 b100000

```
#define b100000 0x20
```

Definition at line 157 of file binary.h.

#### 4.1.2.262 b1000000

```
#define b1000000 0x40
```

Definition at line 256 of file binary.h.

#### 4.1.2.263 b10000000

```
#define b10000000 0x80
```

Definition at line 450 of file binary.h.

#### 4.1.2.264 b10000001

```
#define b10000001 0x81
```

Definition at line 451 of file binary.h.

#### 4.1.2.265 b1000001

```
#define b1000001 0x41
```

Definition at line 257 of file binary.h.

#### 4.1.2.266 b10000010

```
#define b10000010 0x82
```

Definition at line 452 of file binary.h.

#### 4.1.2.267 b10000011

```
#define b10000011 0x83
```

Definition at line 453 of file binary.h.

#### 4.1.2.268 b100001

```
#define b100001 0x21
```

Definition at line 158 of file binary.h.

#### 4.1.2.269 b1000010

```
#define b1000010 0x42
```

Definition at line 258 of file binary.h.

#### 4.1.2.270 b10000100

```
#define b10000100 0x84
```

Definition at line 454 of file binary.h.

#### 4.1.2.271 b10000101

```
#define b10000101 0x85
```

Definition at line 455 of file binary.h.

#### 4.1.2.272 b1000011

```
#define b1000011 0x43
```

Definition at line 259 of file binary.h.

#### 4.1.2.273 b10000110

```
#define b10000110 0x86
```

Definition at line 456 of file binary.h.

#### 4.1.2.274 b10000111

```
#define b10000111 0x87
```

Definition at line 457 of file binary.h.

#### 4.1.2.275 b10001

```
#define b10001 0x11
```

Definition at line 107 of file binary.h.

#### 4.1.2.276 b100010

```
#define b100010 0x22
```

Definition at line 159 of file binary.h.

#### 4.1.2.277 b1000100

```
#define b1000100 0x44
```

Definition at line 260 of file binary.h.

#### 4.1.2.278 b10001000

```
#define b10001000 0x88
```

Definition at line 458 of file binary.h.

#### 4.1.2.279 b10001001

```
#define b10001001 0x89
```

Definition at line 459 of file binary.h.

#### 4.1.2.280 b1000101

```
#define b1000101 0x45
```

Definition at line 261 of file binary.h.

#### 4.1.2.281 b10001010

```
#define b10001010 0x8A
```

Definition at line 460 of file binary.h.

#### 4.1.2.282 b10001011

```
#define b10001011 0x8B
```

Definition at line 461 of file binary.h.

#### 4.1.2.283 b100011

```
#define b100011 0x23
```

Definition at line 160 of file binary.h.

#### 4.1.2.284 b1000110

```
#define b1000110 0x46
```

Definition at line 262 of file binary.h.

#### 4.1.2.285 b10001100

```
#define b10001100 0x8C
```

Definition at line 462 of file binary.h.

#### 4.1.2.286 b10001101

```
#define b10001101 0x8D
```

Definition at line 463 of file binary.h.

#### 4.1.2.287 b1000111

```
#define b1000111 0x47
```

Definition at line 263 of file binary.h.

#### 4.1.2.288 b10001110

```
#define b10001110 0x8E
```

Definition at line 464 of file binary.h.

#### 4.1.2.289 b10001111

```
#define b10001111 0x8F
```

Definition at line 465 of file binary.h.

#### 4.1.2.290 b1001

```
#define b1001 0x9
```

Definition at line 81 of file binary.h.

#### 4.1.2.291 b10010

```
#define b10010 0x12
```

Definition at line 108 of file binary.h.

#### 4.1.2.292 b100100

```
#define b100100 0x24
```

Definition at line 161 of file binary.h.

#### 4.1.2.293 b1001000

```
#define b1001000 0x48
```

Definition at line 264 of file binary.h.

#### 4.1.2.294 b10010000

```
#define b10010000 0x90
```

Definition at line 466 of file binary.h.

#### 4.1.2.295 b10010001

```
#define b10010001 0x91
```

Definition at line 467 of file binary.h.

#### 4.1.2.296 b1001001

```
#define b1001001 0x49
```

Definition at line 265 of file binary.h.

#### 4.1.2.297 b10010010

```
#define b10010010 0x92
```

Definition at line 468 of file binary.h.

#### 4.1.2.298 b10010011

```
#define b10010011 0x93
```

Definition at line 469 of file binary.h.

#### 4.1.2.299 b100101

```
#define b100101 0x25
```

Definition at line 162 of file binary.h.

#### 4.1.2.300 b1001010

```
#define b1001010 0x4A
```

Definition at line 266 of file binary.h.

#### 4.1.2.301 b10010100

```
#define b10010100 0x94
```

Definition at line 470 of file binary.h.

#### 4.1.2.302 b10010101

```
#define b10010101 0x95
```

Definition at line 471 of file binary.h.

#### 4.1.2.303 b1001011

```
#define b1001011 0x4B
```

Definition at line 267 of file binary.h.

#### 4.1.2.304 b10010110

```
#define b10010110 0x96
```

Definition at line 472 of file binary.h.

#### 4.1.2.305 b10010111

```
#define b10010111 0x97
```

Definition at line 473 of file binary.h.

#### 4.1.2.306 b10011

```
#define b10011 0x13
```

Definition at line 109 of file binary.h.

#### 4.1.2.307 b100110

```
#define b100110 0x26
```

Definition at line 163 of file binary.h.

#### 4.1.2.308 b1001100

```
#define b1001100 0x4C
```

Definition at line 268 of file binary.h.

#### 4.1.2.309 b10011000

```
#define b10011000 0x98
```

Definition at line 474 of file binary.h.

#### 4.1.2.310 b10011001

```
#define b10011001 0x99
```

Definition at line 475 of file binary.h.

#### 4.1.2.311 b1001101

```
#define b1001101 0x4D
```

Definition at line 269 of file binary.h.

#### 4.1.2.312 b10011010

```
#define b10011010 0x9A
```

Definition at line 476 of file binary.h.

#### 4.1.2.313 b10011011

```
#define b10011011 0x9B
```

Definition at line 477 of file binary.h.

#### 4.1.2.314 b100111

```
#define b100111 0x27
```

Definition at line 164 of file binary.h.

#### 4.1.2.315 b1001110

```
#define b1001110 0x4E
```

Definition at line 270 of file binary.h.

#### 4.1.2.316 b10011100

```
#define b10011100 0x9C
```

Definition at line 478 of file binary.h.

#### 4.1.2.317 b10011101

```
#define b10011101 0x9D
```

Definition at line 479 of file binary.h.

#### 4.1.2.318 b1001111

```
#define b1001111 0x4F
```

Definition at line 271 of file binary.h.

#### 4.1.2.319 b10011110

```
#define b10011110 0x9E
```

Definition at line 480 of file binary.h.

#### 4.1.2.320 b10011111

```
#define b10011111 0x9F
```

Definition at line 481 of file binary.h.

#### 4.1.2.321 b101

```
#define b101 0x5
```

Definition at line 67 of file binary.h.

#### 4.1.2.322 b1010

```
#define b1010 0xA
```

Definition at line 82 of file binary.h.

#### 4.1.2.323 b10100

```
#define b10100 0x14
```

Definition at line 110 of file binary.h.

#### 4.1.2.324 b101000

```
#define b101000 0x28
```

Definition at line 165 of file binary.h.

#### 4.1.2.325 b1010000

```
#define b1010000 0x50
```

Definition at line 272 of file binary.h.

#### 4.1.2.326 b10100000

```
#define b10100000 0xA0
```

Definition at line 482 of file binary.h.

#### 4.1.2.327 b10100001

```
#define b10100001 0xA1
```

Definition at line 483 of file binary.h.

#### 4.1.2.328 b1010001

```
#define b1010001 0x51
```

Definition at line 273 of file binary.h.

#### 4.1.2.329 b10100010

```
#define b10100010 0xA2
```

Definition at line 484 of file binary.h.

#### 4.1.2.330 b10100011

```
#define b10100011 0xA3
```

Definition at line 485 of file binary.h.

#### 4.1.2.331 b101001

```
#define b101001 0x29
```

Definition at line 166 of file binary.h.

#### 4.1.2.332 b1010010

```
#define b1010010 0x52
```

Definition at line 274 of file binary.h.

#### 4.1.2.333 b10100100

```
#define b10100100 0xA4
```

Definition at line 486 of file binary.h.

#### 4.1.2.334 b10100101

```
#define b10100101 0xA5
```

Definition at line 487 of file binary.h.

#### 4.1.2.335 b1010011

```
#define b1010011 0x53
```

Definition at line 275 of file binary.h.

#### 4.1.2.336 b10100110

```
#define b10100110 0xA6
```

Definition at line 488 of file binary.h.

#### 4.1.2.337 b10100111

```
#define b10100111 0xA7
```

Definition at line 489 of file binary.h.

#### 4.1.2.338 b10101

```
#define b10101 0x15
```

Definition at line 111 of file binary.h.

#### 4.1.2.339 b101010

```
#define b101010 0x2A
```

Definition at line 167 of file binary.h.

#### 4.1.2.340 b1010100

```
#define b1010100 0x54
```

Definition at line 276 of file binary.h.

#### 4.1.2.341 b10101000

```
#define b10101000 0xA8
```

Definition at line 490 of file binary.h.

#### 4.1.2.342 b10101001

```
#define b10101001 0xA9
```

Definition at line 491 of file binary.h.

#### 4.1.2.343 b1010101

```
#define b1010101 0x55
```

Definition at line 277 of file binary.h.

#### 4.1.2.344 b10101010

```
#define b10101010 0xAA
```

Definition at line 492 of file binary.h.

#### 4.1.2.345 b10101011

```
#define b10101011 0xAB
```

Definition at line 493 of file binary.h.

#### 4.1.2.346 b101011

```
#define b101011 0x2B
```

Definition at line 168 of file binary.h.

#### 4.1.2.347 b1010110

```
#define b1010110 0x56
```

Definition at line 278 of file binary.h.

#### 4.1.2.348 b10101100

```
#define b10101100 0xAC
```

Definition at line 494 of file binary.h.

#### 4.1.2.349 b10101101

```
#define b10101101 0xAD
```

Definition at line 495 of file binary.h.

#### 4.1.2.350 b1010111

```
#define b1010111 0x57
```

Definition at line 279 of file binary.h.

#### 4.1.2.351 b10101110

```
#define b10101110 0xAE
```

Definition at line 496 of file binary.h.

#### 4.1.2.352 b10101111

```
#define b10101111 0xAF
```

Definition at line 497 of file binary.h.

#### 4.1.2.353 b1011

```
#define b1011 0xB
```

Definition at line 83 of file binary.h.

#### 4.1.2.354 b10110

```
#define b10110 0x16
```

Definition at line 112 of file binary.h.

#### 4.1.2.355 b101100

```
#define b101100 0x2C
```

Definition at line 169 of file binary.h.

#### 4.1.2.356 b1011000

```
#define b1011000 0x58
```

Definition at line 280 of file binary.h.

#### 4.1.2.357 b10110000

```
#define b10110000 0xB0
```

Definition at line 498 of file binary.h.

#### 4.1.2.358 b10110001

```
#define b10110001 0xB1
```

Definition at line 499 of file binary.h.

#### 4.1.2.359 b1011001

```
#define b1011001 0x59
```

Definition at line 281 of file binary.h.

#### 4.1.2.360 b10110010

```
#define b10110010 0xB2
```

Definition at line 500 of file binary.h.

#### 4.1.2.361 b10110011

```
#define b10110011 0xB3
```

Definition at line 501 of file binary.h.

#### 4.1.2.362 b101101

```
#define b101101 0x2D
```

Definition at line 170 of file binary.h.

#### 4.1.2.363 b1011010

```
#define b1011010 0x5A
```

Definition at line 282 of file binary.h.

#### 4.1.2.364 b10110100

```
#define b10110100 0xB4
```

Definition at line 502 of file binary.h.

#### 4.1.2.365 b10110101

```
#define b10110101 0xB5
```

Definition at line 503 of file binary.h.

#### 4.1.2.366 b1011011

```
#define b1011011 0x5B
```

Definition at line 283 of file binary.h.

#### 4.1.2.367 b10110110

```
#define b10110110 0xB6
```

Definition at line 504 of file binary.h.

#### 4.1.2.368 b10110111

```
#define b10110111 0xB7
```

Definition at line 505 of file binary.h.

#### 4.1.2.369 b10111

```
#define b10111 0x17
```

Definition at line 113 of file binary.h.

#### 4.1.2.370 b101110

```
#define b101110 0x2E
```

Definition at line 171 of file binary.h.

#### 4.1.2.371 b1011100

```
#define b1011100 0x5C
```

Definition at line 284 of file binary.h.

#### 4.1.2.372 b10111000

```
#define b10111000 0xB8
```

Definition at line 506 of file binary.h.

#### 4.1.2.373 b10111001

```
#define b10111001 0xB9
```

Definition at line 507 of file binary.h.

#### 4.1.2.374 b1011101

```
#define b1011101 0x5D
```

Definition at line 285 of file binary.h.

#### 4.1.2.375 b10111010

```
#define b10111010 0xBA
```

Definition at line 508 of file binary.h.

#### 4.1.2.376 b10111011

```
#define b10111011 0xBB
```

Definition at line 509 of file binary.h.

#### 4.1.2.377 b101111

```
#define b101111 0x2F
```

Definition at line 172 of file binary.h.

#### 4.1.2.378 b1011110

```
#define b1011110 0x5E
```

Definition at line 286 of file binary.h.

#### 4.1.2.379 b10111100

```
#define b10111100 0xBC
```

Definition at line 510 of file binary.h.

#### 4.1.2.380 b10111101

```
#define b10111101 0xBD
```

Definition at line 511 of file binary.h.

#### 4.1.2.381 b1011111

```
#define b1011111 0x5F
```

Definition at line 287 of file binary.h.

#### 4.1.2.382 b10111110

```
#define b10111110 0xBE
```

Definition at line 512 of file binary.h.

#### 4.1.2.383 b10111111

```
#define b10111111 0xBF
```

Definition at line 513 of file binary.h.

#### 4.1.2.384 b11

```
#define b11 0x3
```

Definition at line 59 of file binary.h.

#### 4.1.2.385 b110

```
#define b110 0x6
```

Definition at line 68 of file binary.h.

#### 4.1.2.386 b1100

```
#define b1100 0xC
```

Definition at line 84 of file binary.h.

#### 4.1.2.387 b11000

```
#define b11000 0x18
```

Definition at line 114 of file binary.h.

#### 4.1.2.388 b110000

```
#define b110000 0x30
```

Definition at line 173 of file binary.h.

#### 4.1.2.389 b1100000

```
#define b1100000 0x60
```

Definition at line 288 of file binary.h.

#### 4.1.2.390 b11000000

```
#define b11000000 0xC0
```

Definition at line 514 of file binary.h.

#### 4.1.2.391 b11000001

```
#define b11000001 0xC1
```

Definition at line 515 of file binary.h.

#### 4.1.2.392 b1100001

```
#define b1100001 0x61
```

Definition at line 289 of file binary.h.

#### 4.1.2.393 b11000010

```
#define b11000010 0xC2
```

Definition at line 516 of file binary.h.

#### 4.1.2.394 b11000011

```
#define b11000011 0xC3
```

Definition at line 517 of file binary.h.

#### 4.1.2.395 b110001

```
#define b110001 0x31
```

Definition at line 174 of file binary.h.

#### 4.1.2.396 b1100010

```
#define b1100010 0x62
```

Definition at line 290 of file binary.h.

#### 4.1.2.397 b11000100

```
#define b11000100 0xC4
```

Definition at line 518 of file binary.h.

#### 4.1.2.398 b11000101

```
#define b11000101 0xC5
```

Definition at line 519 of file binary.h.

#### 4.1.2.399 b1100011

```
#define b1100011 0x63
```

Definition at line 291 of file binary.h.

#### 4.1.2.400 b11000110

```
#define b11000110 0xC6
```

Definition at line 520 of file binary.h.

#### 4.1.2.401 b11000111

```
#define b11000111 0xC7
```

Definition at line 521 of file binary.h.

#### 4.1.2.402 b11001

```
#define b11001 0x19
```

Definition at line 115 of file binary.h.

#### 4.1.2.403 b110010

```
#define b110010 0x32
```

Definition at line 175 of file binary.h.

#### 4.1.2.404 b1100100

```
#define b1100100 0x64
```

Definition at line 292 of file binary.h.

#### 4.1.2.405 b11001000

```
#define b11001000 0xC8
```

Definition at line 522 of file binary.h.

#### 4.1.2.406 b11001001

```
#define b11001001 0xC9
```

Definition at line 523 of file binary.h.

#### 4.1.2.407 b1100101

```
#define b1100101 0x65
```

Definition at line 293 of file binary.h.

#### 4.1.2.408 b11001010

```
#define b11001010 0xCA
```

Definition at line 524 of file binary.h.

#### 4.1.2.409 b11001011

```
#define b11001011 0xCB
```

Definition at line 525 of file binary.h.

#### 4.1.2.410 b110011

```
#define b110011 0x33
```

Definition at line 176 of file binary.h.

#### 4.1.2.411 b1100110

```
#define b1100110 0x66
```

Definition at line 294 of file binary.h.

#### 4.1.2.412 b11001100

```
#define b11001100 0xCC
```

Definition at line 526 of file binary.h.

#### 4.1.2.413 b11001101

```
#define b11001101 0xCD
```

Definition at line 527 of file binary.h.

#### 4.1.2.414 b1100111

```
#define b1100111 0x67
```

Definition at line 295 of file binary.h.

#### 4.1.2.415 b11001110

```
#define b11001110 0xCE
```

Definition at line 528 of file binary.h.

#### 4.1.2.416 b11001111

```
#define b11001111 0xCF
```

Definition at line 529 of file binary.h.

#### 4.1.2.417 b1101

```
#define b1101 0xD
```

Definition at line 85 of file binary.h.

#### 4.1.2.418 b11010

```
#define b11010 0x1A
```

Definition at line 116 of file binary.h.

#### 4.1.2.419 b110100

```
#define b110100 0x34
```

Definition at line 177 of file binary.h.

#### 4.1.2.420 b1101000

```
#define b1101000 0x68
```

Definition at line 296 of file binary.h.

#### 4.1.2.421 b11010000

```
#define b11010000 0xD0
```

Definition at line 530 of file binary.h.

#### 4.1.2.422 b11010001

```
#define b11010001 0xD1
```

Definition at line 531 of file binary.h.

### 4.1.2.423 b1101001

```
#define b1101001 0x69
```

Definition at line 297 of file binary.h.

### 4.1.2.424 b11010010

```
#define b11010010 0xD2
```

Definition at line 532 of file binary.h.

### 4.1.2.425 b11010011

```
#define b11010011 0xD3
```

Definition at line 533 of file binary.h.

### 4.1.2.426 b110101

```
#define b110101 0x35
```

Definition at line 178 of file binary.h.

### 4.1.2.427 b1101010

```
#define b1101010 0x6A
```

Definition at line 298 of file binary.h.

### 4.1.2.428 b11010100

```
#define b11010100 0xD4
```

Definition at line 534 of file binary.h.

#### 4.1.2.429 b11010101

```
#define b11010101 0xD5
```

Definition at line 535 of file binary.h.

#### 4.1.2.430 b1101011

```
#define b1101011 0x6B
```

Definition at line 299 of file binary.h.

#### 4.1.2.431 b11010110

```
#define b11010110 0xD6
```

Definition at line 536 of file binary.h.

#### 4.1.2.432 b11010111

```
#define b11010111 0xD7
```

Definition at line 537 of file binary.h.

#### 4.1.2.433 b11011

```
#define b11011 0x1B
```

Definition at line 117 of file binary.h.

#### 4.1.2.434 b110110

```
#define b110110 0x36
```

Definition at line 179 of file binary.h.

#### 4.1.2.435 b1101100

```
#define b1101100 0x6C
```

Definition at line 300 of file binary.h.

#### 4.1.2.436 b11011000

```
#define b11011000 0xD8
```

Definition at line 538 of file binary.h.

#### 4.1.2.437 b11011001

```
#define b11011001 0xD9
```

Definition at line 539 of file binary.h.

#### 4.1.2.438 b1101101

```
#define b1101101 0x6D
```

Definition at line 301 of file binary.h.

#### 4.1.2.439 b11011010

```
#define b11011010 0xDA
```

Definition at line 540 of file binary.h.

#### 4.1.2.440 b11011011

```
#define b11011011 0xDB
```

Definition at line 541 of file binary.h.

#### 4.1.2.441 b110111

```
#define b110111 0x37
```

Definition at line 180 of file binary.h.

#### 4.1.2.442 b1101110

```
#define b1101110 0x6E
```

Definition at line 302 of file binary.h.

#### 4.1.2.443 b11011100

```
#define b11011100 0xDC
```

Definition at line 542 of file binary.h.

#### 4.1.2.444 b11011101

```
#define b11011101 0xDD
```

Definition at line 543 of file binary.h.

#### 4.1.2.445 b1101111

```
#define b1101111 0x6F
```

Definition at line 303 of file binary.h.

#### 4.1.2.446 b11011110

```
#define b11011110 0xDE
```

Definition at line 544 of file binary.h.

#### 4.1.2.447 b11011111

```
#define b11011111 0xDF
```

Definition at line 545 of file binary.h.

#### 4.1.2.448 b111

```
#define b111 0x7
```

Definition at line 69 of file binary.h.

#### 4.1.2.449 b1110

```
#define b1110 0xE
```

Definition at line 86 of file binary.h.

#### 4.1.2.450 b11100

```
#define b11100 0x1C
```

Definition at line 118 of file binary.h.

#### 4.1.2.451 b111000

```
#define b111000 0x38
```

Definition at line 181 of file binary.h.

#### 4.1.2.452 b1110000

```
#define b1110000 0x70
```

Definition at line 304 of file binary.h.

#### 4.1.2.453 b11100000

```
#define b11100000 0xE0
```

Definition at line 546 of file binary.h.

#### 4.1.2.454 b11100001

```
#define b11100001 0xE1
```

Definition at line 547 of file binary.h.

#### 4.1.2.455 b1110001

```
#define b1110001 0x71
```

Definition at line 305 of file binary.h.

#### 4.1.2.456 b11100010

```
#define b11100010 0xE2
```

Definition at line 548 of file binary.h.

#### 4.1.2.457 b11100011

```
#define b11100011 0xE3
```

Definition at line 549 of file binary.h.

#### 4.1.2.458 b111001

```
#define b111001 0x39
```

Definition at line 182 of file binary.h.

### 4.1.2.459 b1110010

```
#define b1110010 0x72
```

Definition at line 306 of file binary.h.

### 4.1.2.460 b11100100

```
#define b11100100 0xE4
```

Definition at line 550 of file binary.h.

### 4.1.2.461 b11100101

```
#define b11100101 0xE5
```

Definition at line 551 of file binary.h.

### 4.1.2.462 b1110011

```
#define b1110011 0x73
```

Definition at line 307 of file binary.h.

### 4.1.2.463 b11100110

```
#define b11100110 0xE6
```

Definition at line 552 of file binary.h.

### 4.1.2.464 b11100111

```
#define b11100111 0xE7
```

Definition at line 553 of file binary.h.

#### 4.1.2.465 b11101

```
#define b11101 0x1D
```

Definition at line 119 of file binary.h.

#### 4.1.2.466 b111010

```
#define b111010 0x3A
```

Definition at line 183 of file binary.h.

#### 4.1.2.467 b1110100

```
#define b1110100 0x74
```

Definition at line 308 of file binary.h.

#### 4.1.2.468 b11101000

```
#define b11101000 0xE8
```

Definition at line 554 of file binary.h.

#### 4.1.2.469 b11101001

```
#define b11101001 0xE9
```

Definition at line 555 of file binary.h.

#### 4.1.2.470 b1110101

```
#define b1110101 0x75
```

Definition at line 309 of file binary.h.

#### 4.1.2.471 b11101010

```
#define b11101010 0xEA
```

Definition at line 556 of file binary.h.

#### 4.1.2.472 b11101011

```
#define b11101011 0xEB
```

Definition at line 557 of file binary.h.

#### 4.1.2.473 b111011

```
#define b111011 0x3B
```

Definition at line 184 of file binary.h.

#### 4.1.2.474 b1110110

```
#define b1110110 0x76
```

Definition at line 310 of file binary.h.

#### 4.1.2.475 b11101100

```
#define b11101100 0xEC
```

Definition at line 558 of file binary.h.

#### 4.1.2.476 b11101101

```
#define b11101101 0xED
```

Definition at line 559 of file binary.h.

#### 4.1.2.477 b1110111

```
#define b1110111 0x77
```

Definition at line 311 of file binary.h.

#### 4.1.2.478 b11101110

```
#define b11101110 0xEE
```

Definition at line 560 of file binary.h.

#### 4.1.2.479 b11101111

```
#define b11101111 0xEF
```

Definition at line 561 of file binary.h.

#### 4.1.2.480 b1111

```
#define b1111 0xF
```

Definition at line 87 of file binary.h.

#### 4.1.2.481 b11110

```
#define b11110 0x1E
```

Definition at line 120 of file binary.h.

#### 4.1.2.482 b111100

```
#define b111100 0x3C
```

Definition at line 185 of file binary.h.

#### 4.1.2.483 b1111000

```
#define b1111000 0x78
```

Definition at line 312 of file binary.h.

#### 4.1.2.484 b11110000

```
#define b11110000 0xF0
```

Definition at line 562 of file binary.h.

#### 4.1.2.485 b11110001

```
#define b11110001 0xF1
```

Definition at line 563 of file binary.h.

#### 4.1.2.486 b1111001

```
#define b1111001 0x79
```

Definition at line 313 of file binary.h.

#### 4.1.2.487 b11110010

```
#define b11110010 0xF2
```

Definition at line 564 of file binary.h.

#### 4.1.2.488 b11110011

```
#define b11110011 0xF3
```

Definition at line 565 of file binary.h.

#### 4.1.2.489 b111101

```
#define b111101 0x3D
```

Definition at line 186 of file binary.h.

#### 4.1.2.490 b1111010

```
#define b1111010 0x7A
```

Definition at line 314 of file binary.h.

#### 4.1.2.491 b11110100

```
#define b11110100 0xF4
```

Definition at line 566 of file binary.h.

#### 4.1.2.492 b11110101

```
#define b11110101 0xF5
```

Definition at line 567 of file binary.h.

#### 4.1.2.493 b1111011

```
#define b1111011 0x7B
```

Definition at line 315 of file binary.h.

#### 4.1.2.494 b11110110

```
#define b11110110 0xF6
```

Definition at line 568 of file binary.h.

#### 4.1.2.495 b11110111

```
#define b11110111 0xF7
```

Definition at line 569 of file binary.h.

#### 4.1.2.496 b11111

```
#define b11111 0x1F
```

Definition at line 121 of file binary.h.

#### 4.1.2.497 b111110

```
#define b111110 0x3E
```

Definition at line 187 of file binary.h.

#### 4.1.2.498 b1111100

```
#define b1111100 0x7C
```

Definition at line 316 of file binary.h.

#### 4.1.2.499 b11111000

```
#define b11111000 0xF8
```

Definition at line 570 of file binary.h.

#### 4.1.2.500 b11111001

```
#define b11111001 0xF9
```

Definition at line 571 of file binary.h.

#### 4.1.2.501 b1111101

```
#define b1111101 0x7D
```

Definition at line 317 of file binary.h.

#### 4.1.2.502 b11111010

```
#define b11111010 0xFA
```

Definition at line 572 of file binary.h.

#### 4.1.2.503 b11111011

```
#define b11111011 0xFB
```

Definition at line 573 of file binary.h.

#### 4.1.2.504 b111111

```
#define b111111 0x3F
```

Definition at line 188 of file binary.h.

#### 4.1.2.505 b1111110

```
#define b1111110 0x7E
```

Definition at line 318 of file binary.h.

#### 4.1.2.506 b11111100

```
#define b11111100 0xFC
```

Definition at line 574 of file binary.h.

#### 4.1.2.507 b11111101

```
#define b11111101 0xFD
```

Definition at line 575 of file binary.h.

#### 4.1.2.508 b1111111

```
#define b1111111 0x7F
```

Definition at line 319 of file binary.h.

#### 4.1.2.509 b11111110

```
#define b11111110 0xFE
```

Definition at line 576 of file binary.h.

#### 4.1.2.510 b11111111

```
#define b11111111 0xFF
```

Definition at line 577 of file binary.h.

#### 4.1.2.511 BINARY16

```
#define BINARY16(
              val1,
              val2 )   ((((uint16_t)BINARY8(val1))<<8)|((uint16_t)BINARY8(val2)))
```

Definition at line 44 of file binary.h.

#### 4.1.2.512 BINARY32

```
#define BINARY32(
              val1,
              val2,
              val3,
              val4 )   ((((uint32_t)BINARY16(val1,val2))<<16)|((uint32_t)BINARY16(val3,val4)))
```

Definition at line 47 of file binary.h.

#### 4.1.2.513 BINARY8

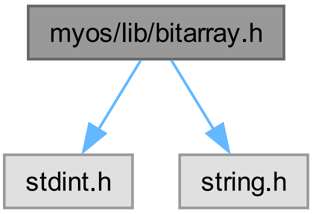
```
#define BINARY8(
              val )  (((uint8_t)(b##val))&0xFF)
```

Definition at line 42 of file binary.h.

## 4.2 myos/lib/bitarray.h File Reference

```
#include <stdint.h>
#include <string.h>
```

Include dependency graph for bitarray.h:



### Macros

- #define BITARRAY_RESET_STATE 0
- #define BITARRAY_SET_STATE 1
- #define BITARRAY(name, size) bitarray_t name##_bitarray[((size)+7)>>3]

  *Defines an bit array.*
- #define BITARRAY_INIT(name) memset(name##_bitarray,0x00,sizeof(name##_bitarray))
- #define BITARRAY_SIZE(name) ((sizeof(name##_bitarray)/sizeof(name##_bitarray[0]))<<3)

  *Size of array in bits.*
- #define BITARRAY_GET(name, bit) ((unsigned)((name##_bitarray[((unsigned)(bit))>>3] & ((1<<(((unsigned)(bit))&0x07)))) > 0))

  *Get a specific bit.*
- #define BITARRAY_SET(name, bit) name##_bitarray[((unsigned)(bit))>>3] |= (1<<(((unsigned)(bit))&0x07))

  *Set a specific bit to 1.*
- #define BITARRAY_RESET(name, bit) name##_bitarray[((unsigned)(bit))>>3] &= ~(1<<(((unsigned)(bit))&0x07))

  *Reset a specific bit to 0.*
- #define BITARRAY_TOGGLE(name, bit) name##_bitarray[((unsigned)(bit))>>3] ^= (1<<(((unsigned)(bit))&0x07))

  *Toggle a specific bit.*
- #define BITARRAY_SET_VALUE(name, bit, value) do{if((value) == 0){BITARRAY_RESET(name,bit);}else{BITARRAY_SET(nan

  *Toggle a specific bit.*

### Typedefs

- typedef uint8_t bitarray_t

## 4.2.1 Detailed Description

**Copyright**

https://opensource.org/license/mit/

## 4.2.2 Macro Definition Documentation

### 4.2.2.1 BITARRAY

```
#define BITARRAY(
              name,
              size )  bitarray_t name##_bitarray[((size)+7)>>3]
```

Defines an bit array.

**Parameters**

| | |
|---|---|
| *name* | Name of the bit array |
| *size* | Minimum number of bits required |

Definition at line 55 of file bitarray.h.

### 4.2.2.2 BITARRAY_GET

```
#define BITARRAY_GET(
              name,
              bit )   ((unsigned)((name##_bitarray[((unsigned)(bit))>>3] & ((1<<(((unsigned)(bit))&0x07))))
> 0))
```

Get a specific bit.

**Parameters**

| | |
|---|---|
| *name* | Name of the bit array |
| *bit* | Specific bit index |

**Returns**

Value of the specific bit. Either 0 or 1.

Definition at line 81 of file bitarray.h.

#### 4.2.2.3 BITARRAY_INIT

```
#define BITARRAY_INIT(
              name )   memset(name##_bitarray,0x00,sizeof(name##_bitarray))
```

Definition at line 58 of file bitarray.h.

#### 4.2.2.4 BITARRAY_RESET

```
#define BITARRAY_RESET(
              name,
              bit )   name##_bitarray[((unsigned)(bit))>>3] &= ~(1<<(((unsigned)(bit))&0x07))
```

Reset a specific bit to 0.

**Parameters**

| | |
|---|---|
| *name* | Name of the bit array |
| *bit* | Specific bit index |

Definition at line 102 of file bitarray.h.

#### 4.2.2.5 BITARRAY_RESET_STATE

```
#define BITARRAY_RESET_STATE 0
```

Definition at line 43 of file bitarray.h.

#### 4.2.2.6 BITARRAY_SET

```
#define BITARRAY_SET(
              name,
              bit )   name##_bitarray[((unsigned)(bit))>>3] |= (1<<(((unsigned)(bit))&0x07))
```

Set a specific bit to 1.

**Parameters**

| | |
|---|---|
| *name* | Name of the bit array |
| *bit* | Specific bit index |

Definition at line 93 of file bitarray.h.

#### 4.2.2.7 BITARRAY_SET_STATE

```
#define BITARRAY_SET_STATE 1
```

Definition at line 44 of file bitarray.h.

#### 4.2.2.8 BITARRAY_SET_VALUE

```
#define BITARRAY_SET_VALUE(
              name,
              bit,
              value )   do{if((value) == 0){BITARRAY_RESET(name,bit);}else{BITARRAY_SET(name,bit);}}while(0)
```

Toggle a specific bit.

**Parameters**

| | |
|---|---|
| *name* | Name of the bit array |
| *bit* | Specific bit index |

Definition at line 120 of file bitarray.h.

#### 4.2.2.9 BITARRAY_SIZE

```
#define BITARRAY_SIZE(
              name )   ((sizeof(name##_bitarray)/sizeof(name##_bitarray[0]))<<3)
```

Size of array in bits.

Returns the size of a bit array in bits. The size is expressed as a multiple of 8.

**Parameters**

| | |
|---|---|
| *name* | Name of the bit array |

Definition at line 71 of file bitarray.h.

#### 4.2.2.10 BITARRAY_TOGGLE

```
#define BITARRAY_TOGGLE(
                name,
                bit )   name##_bitarray[((unsigned)(bit))>>3] ^= (1<<(((unsigned)(bit))&0x07))
```

Toggle a specific bit.

**Parameters**

| | |
|---|---|
| *name* | Name of the bit array |
| *bit* | Specific bit index |

Definition at line 111 of file bitarray.h.

### 4.2.3 Typedef Documentation

#### 4.2.3.1 bitarray_t

```
typedef uint8_t bitarray_t
```

Definition at line 46 of file bitarray.h.

## 4.3 myos/lib/bits.h File Reference

Provides preprocessor macros for bit manipulations.

### Macros

- #define BITS(x, m) ((x)&(m))

  *Read out bits from bit field by using a bit mask.*
- #define BITS_INVERT(x) (∼x)

  *Toggle all bits of integer.*
- #define BITS_TEST(x, m) (((x)&(m))==(m)))

  *Test if all of the bits defined by bit mask are set in the bit field.*
- #define BITS_SET(x, m) ((x)|=(m))

  *Set bits defined by bit mask to logical 1 in bit field.*
- #define BITS_CLEAR(x, m) ((x)&=∼(m))

  *Set bits defined by bit mask to logical 0 in bit field.*
- #define BITS_TOGGLE(x, m) ((x)^=(m))

  *Toggle bits in bit field as defined by bit mask.*

### 4.3.1 Detailed Description

Provides preprocessor macros for bit manipulations.

**Copyright**

Bit field

```
A bit field is termed an unsigned integer in which single bits or
group of bits are stringed together. It represents a kind of compound
data type on bit level, in contrast to primitive data types, where the
value is build by interpreting all positions of the primitive data type together.

A bit field is always embedded in the data type of the binary number (in contrast to bit arrays).


Bit mask

Accessing a single bit, either in reading or writing manner,
is supported by hardware in a similar way like accessing a
byte or word - it will do with only one command for most machines.

But support through compiler is often similar to an access on several
bits, where a bit group has to be "removed" from the variable word before
comparison or manipulation. This is done by using a a so called bit mask
and some elementary boolean bit operations.

Bit masks are called bit fields that do not represent any information themselves,
but are used to read out or manipulate bit fields.


Source: https://de.wikipedia.org/wiki/Bitfeld
Source: https://de.wikipedia.org/wiki/Bitkette
```

### 4.3.2 Macro Definition Documentation

#### 4.3.2.1 BITS

```
#define BITS(
              x,
              m ) ((x)&(m))
```

Read out bits from bit field by using a bit mask.

To read out one or more specific bits of a bit field, it is logically AND-ed with a bit mask.

```
Example

1-Bit:

    01001011 bit field
AND 00001000 bit mask
-------------
=   00001000 result

0-Bit:

    01001011 bit field
AND 00000100 bit mask
-------------
=   00000000 result
```

Definition at line 89 of file bits.h.

#### 4.3.2.2 BITS_CLEAR

```
#define BITS_CLEAR(
              x,
              m ) ((x)&=~(m))
```

Set bits defined by bit mask to logical 0 in bit field.

To clear one or more specific bits of a bit field, it is logically NAND-ed with a bit mask defining the bits to be cleared.

Set bits to „0“:

#### 4.3.2.3 NOT 00001000 bit mask

= 11110111 inverted bit mask

#### 4.3.2.4 AND 01001011 bit field

= 01000011 result

Definition at line 132 of file bits.h.

#### 4.3.2.5 BITS_INVERT

```
#define BITS_INVERT(
              x ) (∼x)
```

Toggle all bits of integer.

Definition at line 94 of file bits.h.

#### 4.3.2.6 BITS_SET

```
#define BITS_SET(
              x,
              m ) ((x)|=(m))
```

Set bits defined by bit mask to logical 1 in bit field.

To set one or more specific bits of a bit field, it is logically OR-ed with a bit mask defining the bits to be set.

Example

Set bits to „1“:

```
01001011 bit field
```

#### 4.3.2.7 OR 00000100 bit mask

= 01001111 result

Definition at line 116 of file bits.h.

#### 4.3.2.8 BITS_TEST

```
#define BITS_TEST(
              x,
              m ) (((x)&(m))==(m)))
```

Test if all of the bits defined by bit mask are set in the bit field.

Definition at line 99 of file bits.h.

#### 4.3.2.9 BITS_TOGGLE

```
#define BITS_TOGGLE(
             x,
             m ) ((x)^=(m))
```

Toggle bits in bit field as defined by bit mask.

To toggle one or more specific bits of a bit field, it is logically XOR-ed with a bit mask defining the bits to be toggled.

```
01001011 bit field
```

#### 4.3.2.10 XOR 00000110 bit mask

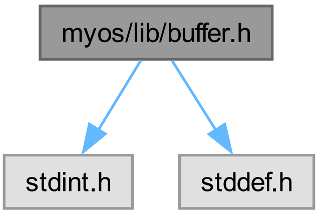
= 01001101 result

Definition at line 144 of file bits.h.

## 4.4 myos/lib/buffer.h File Reference

```
#include <stdint.h>
#include <stddef.h>
```

Include dependency graph for buffer.h:



### Macros

- #define BUFFER_TYPEDEF(name, type, size)
- #define BUFFER_T(name) name##_buffer_t
- #define BUFFER_ITEMS(buffer) ((buffer).items)
- #define BUFFER_RAW(buffer) ((uint8_t*)BUFFER_ITEMS(buffer))
- #define BUFFER_SIZEOF(buffer) (sizeof(BUFFER_ITEMS(buffer)))
- #define BUFFER_SIZE(buffer) (BUFFER_SIZEOF(buffer)/sizeof(BUFFER_ITEMS(buffer)[0]))
- #define BUFFER_COUNT(buffer) ((buffer).count)
- #define BUFFER_INIT(buffer) do{BUFFER_COUNT(buffer)=0;}while(0)
- #define BUFFER_FULL(buffer) (BUFFER_COUNT(buffer)>=BUFFER_SIZE(buffer))
- #define BUFFER_EMPTY(buffer) (!BUFFER_COUNT(buffer))
- #define BUFFER_NEXT(buffer) (++BUFFER_COUNT(buffer))
- #define BUFFER_VAL(buffer) (BUFFER_ITEMS(buffer)[BUFFER_COUNT(buffer)])
- #define BUFFER_PTR(buffer) (&BUFFER_VAL(buffer))
- #define BUFFER_APPEND(buffer, item) do{BUFFER_VAL(buffer) = item; BUFFER_NEXT(buffer);}while(0)

## 4.4.1 Detailed Description

**Copyright**

## 4.4.2 Macro Definition Documentation

### 4.4.2.1 BUFFER_APPEND

```
#define BUFFER_APPEND(
              buffer,
              item )   do{BUFFER_VAL(buffer) = item; BUFFER_NEXT(buffer);}while(0)
```

Append an item to the buffer by value. Automatically increases the buffer count. It does not check for buffer overflow. In case of buffer overflow the behaviour is undefined. Use BUFFER_FULL to check if buffer is full before using this macro.

Definition at line 146 of file buffer.h.

### 4.4.2.2 BUFFER_COUNT

```
#define BUFFER_COUNT(
              buffer )   ((buffer).count)
```

Returns the current number of items used in the buffer.

Definition at line 95 of file buffer.h.

### 4.4.2.3 BUFFER_EMPTY

```
#define BUFFER_EMPTY(
              buffer )   (!BUFFER_COUNT(buffer))
```

Checks if the buffer has no items stored. Returns 1 if buffer is empty, otherwise 0.

Definition at line 116 of file buffer.h.

### 4.4.2.4 BUFFER_FULL

```
#define BUFFER_FULL(
              buffer )   (BUFFER_COUNT(buffer)>=BUFFER_SIZE(buffer))
```

Checks if no further buffer items are available. Returns 1 if buffer is full, otherwise 0.

Definition at line 109 of file buffer.h.

### 4.4.2.5 BUFFER_INIT

```
#define BUFFER_INIT(
              buffer )   do{BUFFER_COUNT(buffer)=0;}while(0)
```

Initializes the buffer.

Definition at line 101 of file buffer.h.

### 4.4.2.6 BUFFER_ITEMS

```
#define BUFFER_ITEMS(
              buffer )   ((buffer).items)
```

Returns a pointer to the buffer items. Type of returned pointer is of buffer item type.

Definition at line 69 of file buffer.h.

### 4.4.2.7 BUFFER_NEXT

```
#define BUFFER_NEXT(
              buffer )   (++BUFFER_COUNT(buffer))
```

Moves to next free item in the buffer. It does not check for buffer overflow. In case of buffer overflow the behaviour is undefined. Use BUFFER_FULL to check if buffer is full before using this macro.

Definition at line 124 of file buffer.h.

#### 4.4.2.8 BUFFER_PTR

```
#define BUFFER_PTR(
             buffer )   (&BUFFER_VAL(buffer))
```

Returns a pointer to the current free item.

Definition at line 136 of file buffer.h.

#### 4.4.2.9 BUFFER_RAW

```
#define BUFFER_RAW(
             buffer )   ((uint8_t*)BUFFER_ITEMS(buffer))
```

Returns an pointer to the buffer items. Type of pointer is void.

Definition at line 76 of file buffer.h.

#### 4.4.2.10 BUFFER_SIZE

```
#define BUFFER_SIZE(
             buffer )   (BUFFER_SIZEOF(buffer)/sizeof(BUFFER_ITEMS(buffer)[0]))
```

Returns the maximum number of items the buffer can hold.

Definition at line 89 of file buffer.h.

#### 4.4.2.11 BUFFER_SIZEOF

```
#define BUFFER_SIZEOF(
             buffer )   (sizeof(BUFFER_ITEMS(buffer)))
```

Returns the size of the buffer in bytes.

Definition at line 82 of file buffer.h.

#### 4.4.2.12 BUFFER_T

```
#define BUFFER_T(
             name )   name##_buffer_t
```

Defines a buffer of type *name*. Can also be combined with different storage classifiers. (static, extern, volatile ...)

Definition at line 61 of file buffer.h.

#### 4.4.2.13 BUFFER_TYPEDEF

```
#define BUFFER_TYPEDEF(
              name,
              type,
              size )
```

**Value:**

```
   typedef struct {                  \
          size_t count;              \
          type items [size];         \
   } name##_buffer_t
```

Declares a buffer type. The argumnents are *name* for a unique identification of the buffer type, the *type* of the items and the number of items of *type* the buffer can hold (*size*).

Definition at line 50 of file buffer.h.

#### 4.4.2.14 BUFFER_VAL

```
#define BUFFER_VAL(
              buffer )   (BUFFER_ITEMS(buffer)[BUFFER_COUNT(buffer)])
```

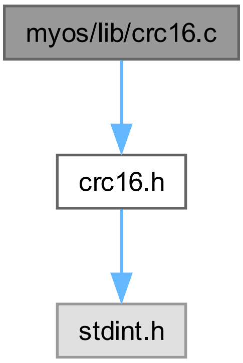
Returns the current free item by value.

Definition at line 130 of file buffer.h.

## 4.5 myos/lib/crc16.c File Reference

```
#include "crc16.h"
```

Include dependency graph for crc16.c:



## Functions

- uint16_t crc16_acc (uint16_t seed, uint16_t polynom, uint8_t byte)

### 4.5.1 Function Documentation

#### 4.5.1.1 crc16_acc()

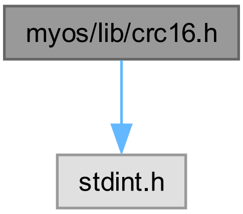
```
uint16_t crc16_acc (
             uint16_t seed,
             uint16_t polynom,
             uint8_t byte )
```

Definition at line 5 of file crc16.c.

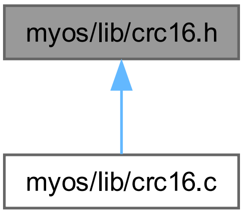
## 4.6 myos/lib/crc16.h File Reference

```
#include <stdint.h>
```

Include dependency graph for crc16.h:



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



## Macros

- #define CRC16_DNP 0x3D65
- #define CRC16_CCITT 0x1021
- #define CRC16_IBM 0x8005
- #define CRC16_T10_DIF 0x8BB7
- #define CRC16_DECT 0x0589
- #define CRC16_ARINC 0xA02B

## Functions

- uint16_t crc16_acc (uint16_t seed, uint16_t polynom, uint8_t byte)

## 4.6.1 Macro Definition Documentation

### 4.6.1.1 CRC16_ARINC

```
#define CRC16_ARINC 0xA02B
```

Definition at line 12 of file crc16.h.

### 4.6.1.2 CRC16_CCITT

```
#define CRC16_CCITT 0x1021
```

Definition at line 8 of file crc16.h.

### 4.6.1.3 CRC16_DECT

```
#define CRC16_DECT 0x0589
```

Definition at line 11 of file crc16.h.

### 4.6.1.4 CRC16_DNP

```
#define CRC16_DNP 0x3D65
```

Definition at line 7 of file crc16.h.

#### 4.6.1.5 CRC16_IBM

```
#define CRC16_IBM 0x8005
```

Definition at line 9 of file crc16.h.

#### 4.6.1.6 CRC16_T10_DIF

```
#define CRC16_T10_DIF 0x8BB7
```

Definition at line 10 of file crc16.h.

### 4.6.2 Function Documentation

#### 4.6.2.1 crc16_acc()

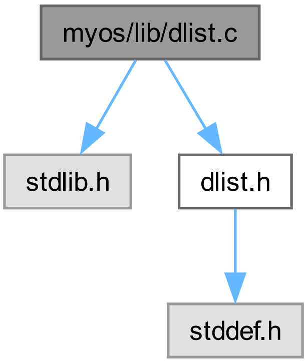
```
uint16_t crc16_acc (
             uint16_t seed,
             uint16_t polynom,
             uint8_t byte )
```

Definition at line 5 of file crc16.c.

## 4.7 myos/lib/dlist.c File Reference

```
#include <stdlib.h>
#include "dlist.h"
```

Include dependency graph for dlist.c:



## Functions

- dlist_node_t ∗ dlist_find (dlist_t ∗dlist, void ∗node)
- size_t dlist_size (dlist_t ∗dlist)

### 4.7.1 Function Documentation

#### 4.7.1.1 dlist_find()

```
dlist_node_t∗ dlist_find (
            dlist_t ∗ dlist,
            void ∗ node )
```

Definition at line 61 of file dlist.c.

#### 4.7.1.2 dlist_size()

```
size_t dlist_size (
            dlist_t ∗ dlist )
```

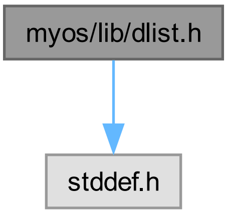
Definition at line 77 of file dlist.c.

## 4.8 myos/lib/dlist.h File Reference

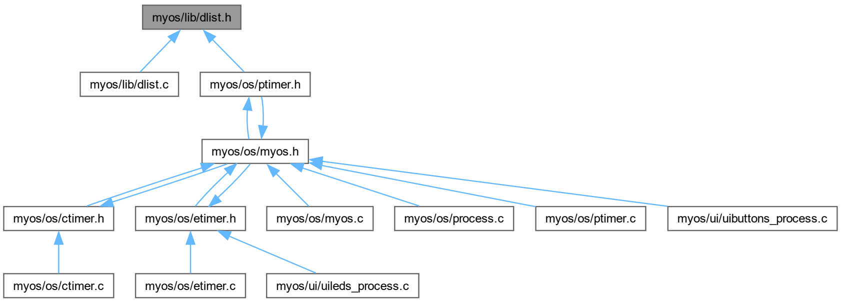
Circular doubly linked list.

```
#include <stddef.h>
```

Include dependency graph for dlist.h:



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



## Data Structures

- struct dlist_node_t

## Macros

- #define DLIST_NODE_TYPE dlist_node_t dlist_node
- #define dlist_next(dlist, existing_node) (((dlist_node_t∗)(existing_node))->next)

  *Get next list node.*
- #define dlist_prev(dlist, existing_node) (((dlist_node_t∗)(existing_node))->prev)

  *Get previous list node.*
- #define dlist_init(dlist)

  *Initialize list.*
- #define dlist_begin(dlist) ((dlist)->next)

  *Return iterator to beginning.*
- #define dlist_front(dlist) (dlist_begin(dlist))
- #define dlist_end(dlist) (dlist)

  *Return iterator to end.*
- #define dlist_back(dlist) (dlist_prev(dlist,dlist))
- #define dlist_foreach(dlist, iterator) for(iterator = (void∗)dlist_begin(dlist); ((dlist_node_t∗)iterator) != dlist_end(dlist); iterator=(void∗)dlist_next(dlist,iterator))

  *Iterate through whole list.*
- #define dlist_empty(dlist) ((dlist)->next==(dlist))

  *Test whether container is empty.*
- #define dlist_push_front(dlist, node_to_add)

  *Insert element at beginning.*
- #define dlist_pop_front(dlist)

  *Delete first node.*
- #define dlist_push_back(dlist, node_to_add)

  *Insert element at end.*
- #define dlist_pop_back(dlist)
- #define dlist_erase(dlist, existing_node)

  *Removes a specific node from list.*
- #define dlist_insert_after(dlist, existing_node, node_to_insert)

  *Insert elements after position.*
- #define dlist_insert_before(dlist, existing_node, node_to_insert)

  *Insert element before position.*

## Typedefs

- typedef struct dlist_node_t dlist_node_t
- typedef dlist_node_t dlist_t

## Functions

- size_t dlist_size (dlist_t ∗dlist)
- dlist_node_t ∗ dlist_find (dlist_t ∗dlist, void ∗node)

### 4.8.1 Detailed Description

Circular doubly linked list.

**Copyright**

**Authors**

marco@bacchi.at

In computer science, a linked list is a linear collection of data elements, whose order is not given by their physical placement in memory. Instead, each element points to the next. It is a data structure consisting of a collection of nodes which together represent a sequence. In its most basic form, each node contains: data, and a reference (in other words, a link) to the next node in the sequence. This structure allows for efficient insertion or removal of elements from any position in the sequence during iteration. [ https://en.wikipedia.org/wiki/↩Linked_list]

In a 'doubly linked list', each node contains, besides the next-node link, a second link field pointing to the 'previous' node in the sequence. The two links may be called 'forward('s') and 'backwards', or 'next' and 'prev'('previous'). [ https://en.wikipedia.org/wiki/Linked_list#Doubly_linked_list]

In the last node of a list, the link field often contains a null reference, a special value is used to indicate the lack of further nodes. A less common convention is to make it point to the first node of the list; in that case, the list is said to be 'circular' or 'circularly linked'; otherwise, it is said to be 'open' or 'linear'. It is a list where the last pointer points to the first node. [ https://en.wikipedia.org/wiki/Linked_list#Circular_linked_list]

## 4.8.2 Macro Definition Documentation

### 4.8.2.1 dlist_back

```
#define dlist_back(
              dlist ) (dlist_prev(dlist,dlist))
```

Definition at line 136 of file dlist.h.

### 4.8.2.2 dlist_begin

```
#define dlist_begin(
              dlist ) ((dlist)->next)
```

Return iterator to beginning.

Returns an iterator pointing to the first element in the list container. If the container is empty, the returned iterator value shall not be dereferenced.

**Parameters**

| | |
|---|---|
| *list* | Listsize_t dlist_size(dlist_t ∗dlist); |

**Returns**

Pointer to the first node or on the list itself.

Definition at line 112 of file dlist.h.

### 4.8.2.3 dlist_empty

```
#define dlist_empty(
              dlist ) ((dlist)->next==(dlist))
```

Test whether container is empty.

Returns whether the list container is empty or not.

**Parameters**

| | |
|---|---|
| *list* | List |

**Returns**

0 if it is not empty or 1 if it is empty

Definition at line 157 of file dlist.h.

### 4.8.2.4 dlist_end

```
#define dlist_end(
              dlist ) (dlist)
```

Return iterator to end.

Returns an iterator referring to the past-the-end element in the list container. It does not point to any element, and thus shall not be dereferenced. This function is often used in combination with list_begin to specify a range including all the elements in the container. As the Successor of the last element is the list itself, it returns the list itself.

**Parameters**

| | |
|---|---|
| *list* | List |

**Returns**

Iterator referring to the past-the-end element

Definition at line 127 of file dlist.h.

### 4.8.2.5 dlist_erase

```
#define dlist_erase(
              dlist,
              existing_node )
```

**Value:**

```
    do{ \
        ((dlist_node_t*)existing_node)->prev->next = ((dlist_node_t*)existing_node)->next; \
        ((dlist_node_t*)existing_node)->next->prev = ((dlist_node_t*)existing_node)->prev; \
    } while(0)
```

Removes a specific node from list.

Removes a specific node from list. Node has to be a member of the list, otherwise erasing the node results in undefined behavior

**Parameters**

| | |
|---|---|
| *list* | List |
| *nodep* | Node to remove from list |

Definition at line 221 of file dlist.h.

#### 4.8.2.6 dlist_foreach

```
#define dlist_foreach(
              dlist,
              iterator )   for(iterator = (void*)dlist_begin(dlist); ((dlist_node_t*)iterator)
!= dlist_end(dlist); iterator=(void*)dlist_next(dlist,iterator))
```

Iterate through whole list.

Iterates from first to last element in the list container.

**Parameters**

| | |
|---|---|
| *slistptr* | List |
| *iterator* | |

Definition at line 146 of file dlist.h.

#### 4.8.2.7 dlist_front

```
#define dlist_front(
              dlist )  (dlist_begin(dlist))
```

Definition at line 113 of file dlist.h.

#### 4.8.2.8 dlist_init

```
#define dlist_init(
              dlist )
```

**Value:**

```
    do{ \
        (dlist)->next = (dlist); \
        (dlist)->prev = (dlist); \
    }while(0)
```

Initialize list.

An empty list only contains the list instance pointing to itself. It is important not to forget to initialize the list, otherwise there may be undefined behavior with list functions and function like macros.

**Parameters**

| | |
|---|---|
| *list* | List to initialize |

Definition at line 97 of file dlist.h.

#### 4.8.2.9 dlist_insert_after

```
#define dlist_insert_after(
              dlist,
              existing_node,
              node_to_insert )
```

**Value:**

```
    do{ \
        ((dlist_node_t*)node_to_insert)->next = ((dlist_node_t*)existing_node)->next; \
        ((dlist_node_t*)node_to_insert)->prev = existing_node; \
        ((dlist_node_t*)existing_node)->next->prev = node_to_insert; \
        ((dlist_node_t*)existing_node)->next = node_to_insert; \
    } while(0)
```

Insert elements after position.

The container is extended by inserting a new element after the element at the specified position.

**Parameters**

| | |
|---|---|
| *list* | List |
| *existing_node* | Node to insert new node after |
| *node_to_insert* | Node to insert |

Definition at line 235 of file dlist.h.

#### 4.8.2.10 dlist_insert_before

```
#define dlist_insert_before(
              dlist,
              existing_node,
              node_to_insert )
```

**Value:**

```
    do{ \
        ((dlist_node_t*)node_to_insert)->next = existing_node; \
        ((dlist_node_t*)node_to_insert)->prev = ((dlist_node_t*)existing_node)->prev; \
        ((dlist_node_t*)existing_node)->prev->next = node_to_insert; \
        ((dlist_node_t*)existing_node)->prev = node_to_insert; \
    } while(0)
```

Insert element before position.

The container is extended by inserting a new element before the element at the specified position.

**Parameters**

| | |
|---|---|
| *list* | List |
| *existing_node* | Node to insert new node before |
| *node_to_insert* | Node to insert |

Definition at line 250 of file dlist.h.

### 4.8.2.11 dlist_next

```
#define dlist_next(
              dlist,
              existing_node )   (((dlist_node_t*)(existing_node))->next)
```

Get next list node.

**Parameters**

| | |
|---|---|
| *list* | List |
| *node* | Current node |

**Returns**

Successor of current node

Definition at line 75 of file dlist.h.

### 4.8.2.12 DLIST_NODE_TYPE

```
#define DLIST_NODE_TYPE dlist_node_t dlist_node
```

Add DLIST_NODE_TYPE as first member of a structure to make it a list node

Definition at line 64 of file dlist.h.

### 4.8.2.13 dlist_pop_back

```
#define dlist_pop_back(
              dlist )
```

**Value:**

```
    do { \
        (dlist)->prev = (dlist)->prev->prev; \
        (dlist)->prev->next = (dlist); \
    }while(0)
```

Definition at line 206 of file dlist.h.

### 4.8.2.14 dlist_pop_front

```
#define dlist_pop_front(
              dlist )
```

**Value:**

```
    do{ \
        (dlist)->next = (dlist)->next->next; \
        (dlist)->next->prev = (dlist); \
    }while(0);
```

Delete first node.

Removes the first node of the list container.

Definition at line 178 of file dlist.h.

### 4.8.2.15 dlist_prev

```
#define dlist_prev(
              dlist,
              existing_node )   (((dlist_node_t*)(existing_node))->prev)
```

Get previous list node.

**Parameters**

| | |
|---|---|
| *list* | List |
| *node* | Current node |

**Returns**

Precessor of current node

Definition at line 86 of file dlist.h.

### 4.8.2.16 dlist_push_back

```
#define dlist_push_back(
              dlist,
              node_to_add )
```

**Value:**

```
    do { \
        ((dlist_node_t*)(node_to_add))->next = (dlist); \
        ((dlist_node_t*)(node_to_add))->prev = (dlist)->prev; \
        (dlist)->prev->next = ((dlist_node_t*)(node_to_add)); \
        (dlist)->prev = ((dlist_node_t*)(node_to_add)); \
    }while(0)
```

Insert element at end.

Inserts a new node at the end of the list, right after the current back node.

**Parameters**

| | |
|---|---|
| *list* | List |
| *node* | Node to add to the list |

Definition at line 192 of file dlist.h.

#### 4.8.2.17 dlist_push_front

```
#define dlist_push_front(
                dlist,
                node_to_add )
```

**Value:**

```
    do{ \
        ((dlist_node_t*)(node_to_add))->next = (dlist)->next; \
        ((dlist_node_t*)(node_to_add))->prev = (dlist); \
        (dlist)->next->prev = ((dlist_node_t*)(node_to_add)); \
        (dlist)->next = ((dlist_node_t*)(node_to_add)); \
    }while(0);
```

Insert element at beginning.

Inserts a new node at the beginning of the list, right before its current first element.

**Parameters**

| | |
|---|---|
| *list* | List \Param node_to_add Node to add to the list |

Definition at line 166 of file dlist.h.

### 4.8.3 Typedef Documentation

#### 4.8.3.1 dlist_node_t

```
typedef struct dlist_node_t dlist_node_t
```

Definition at line 1 of file dlist.h.

#### 4.8.3.2 dlist_t

```
typedef dlist_node_t dlist_t
```

List instance is also a member of the list

Definition at line 266 of file dlist.h.

### 4.8.4 Function Documentation

#### 4.8.4.1 dlist_find()

```
dlist_node_t* dlist_find (
             dlist_t * dlist,
             void * node )
```

Definition at line 61 of file dlist.c.

#### 4.8.4.2 dlist_size()

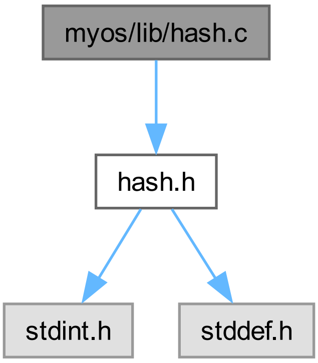
```
size_t dlist_size (
             dlist_t * dlist )
```

Definition at line 77 of file dlist.c.

## 4.9 myos/lib/hash.c File Reference

```
#include "hash.h"
```

Include dependency graph for hash.c:



### Functions

- uint32_t hash_sdbm (uint32_t seed, void *data, size_t size)

### 4.9.1 Function Documentation

#### 4.9.1.1 hash_sdbm()

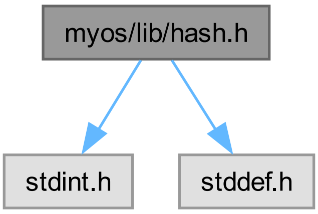
```
uint32_t hash_sdbm (
             uint32_t seed,
             void * data,
             size_t size )
```

Definition at line 37 of file hash.c.

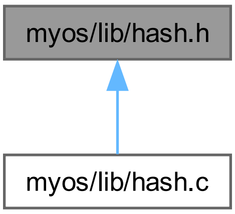
## 4.10 myos/lib/hash.h File Reference

```
#include <stdint.h>
#include <stddef.h>
```

Include dependency graph for hash.h:



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



## Macros

- #define hash_sdbm_acc(hash, byte)  (byte + (hash << 6) + (hash << 16) - hash)

## Functions

- uint32_t hash_sdbm (uint32_t seed, void ∗data, size_t size)

## 4.10.1 Macro Definition Documentation

### 4.10.1.1 hash_sdbm_acc

```
#define hash_sdbm_acc(
              hash,
              byte )   (byte + (hash << 6) + (hash << 16) - hash)
```

Definition at line 18 of file hash.h.

## 4.10.2 Function Documentation

### 4.10.2.1 hash_sdbm()

```
uint32_t hash_sdbm (
             uint32_t seed,
             void * data,
             size_t size )
```

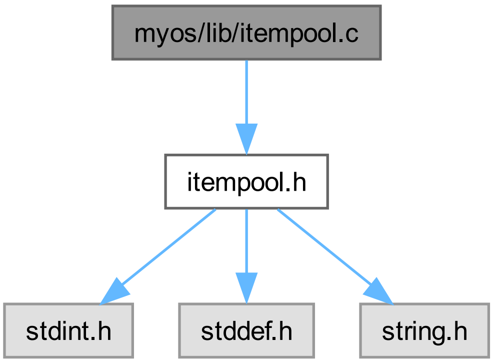
Definition at line 37 of file hash.c.

## 4.11 myos/lib/itempool.c File Reference

see itempool.h

```
#include "itempool.h"
```

Include dependency graph for itempool.c:



### Functions

- void ∗ itempool_alloc (uint8_t ∗items, uint8_t ∗status, size_t itemsize, size_t poolsize)
- void ∗ itempool_calloc (uint8_t ∗items, uint8_t ∗status, size_t itemsize, size_t poolsize)

### 4.11.1 Detailed Description

see itempool.h

see itempool.h

### 4.11.2 Function Documentation

#### 4.11.2.1 itempool_alloc()

```
void* itempool_alloc (
            uint8_t * items,
            uint8_t * status,
            size_t itemsize,
            size_t poolsize )
```

Definition at line 55 of file itempool.c.

Here is the caller graph for this function:

## 4.12 myos/lib/itempool.h File Reference

```
#include <stdint.h>
#include <stddef.h>
#include <string.h>
```

Include dependency graph for itempool.h: This graph shows which files directly or indirectly include this file:

### Macros

- #define ITEMPOOL_ITEM_FREE 0
- #define ITEMPOOL_ITEM_USED 1
- #define ITEMPOOL_TYPEDEF(name, type, size)
- #define ITEMPOOL_T(name)  name##_itempool_t
- #define ITEMPOOL_INIT(itempool)
- #define ITEMPOOL_SIZE(itempool)
- #define ITEMPOOL_ITEM_SIZE(itempool)  (sizeof(*ITEMPOOL_ITEMS(itempool)))
- #define ITEMPOOL_STATUS(itempool)  ((itempool).status)
- #define ITEMPOOL_ITEMS(itempool)  ((itempool).items)
- #define ITEMPOOL_ALLOC(itempool)
- #define ITEMPOOL_CALLOC(itempool)
- #define ITEMPOOL_FREE(itempool, itemptr)

### Functions

- void * itempool_alloc (uint8_t *items, uint8_t *status, size_t itemsize, size_t poolsize)
- void * itempool_calloc (uint8_t *items, uint8_t *status, size_t itemsize, size_t poolsize)

### 4.12.1 Detailed Description

Memory pools, also known as fixed-size block allocation, enable dynamic memory allocation similar to malloc or C++'s operator new. However, these implementations are prone to fragmentation due to variable block sizes and are therefore not recommended for use in real-time systems where performance is critical. To address this issue, a more efficient approach is to preallocate a fixed number of memory blocks of the same size, which is known as a memory pool. During runtime, the application can allocate, access, and free blocks represented by handles, resulting in improved performance.

### 4.12.2 Macro Definition Documentation

#### 4.12.2.1 ITEMPOOL_ALLOC

```
#define ITEMPOOL_ALLOC(
            itempool )
```

**Value:**

```
    itempool_alloc( \
        (uint8_t*)ITEMPOOL_ITEMS(itempool), \
        ITEMPOOL_STATUS(itempool), \
        ITEMPOOL_ITEM_SIZE(itempool), \
        ITEMPOOL_SIZE(itempool))
```

Definition at line 80 of file itempool.h.

### 4.12.2.2 ITEMPOOL_CALLOC

```
#define ITEMPOOL_CALLOC(
                itempool )
```

**Value:**

```
    itempool_calloc( \
        (uint8_t*)ITEMPOOL_ITEMS(itempool), \
        ITEMPOOL_STATUS(itempool), \
        ITEMPOOL_ITEM_SIZE(itempool), \
        ITEMPOOL_SIZE(itempool))
```

Definition at line 87 of file itempool.h.

### 4.12.2.3 ITEMPOOL_FREE

```
#define ITEMPOOL_FREE(
                itempool,
                itemptr )
```

**Value:**

```
        do{ITEMPOOL_STATUS(itempool) \
            [((uint8_t*)itemptr-(uint8_t*)ITEMPOOL_ITEMS(itempool))/ \
             ITEMPOOL_ITEM_SIZE(itempool)] \
             =ITEMPOOL_ITEM_FREE;}while(0)
```

Definition at line 102 of file itempool.h.

### 4.12.2.4 ITEMPOOL_INIT

```
#define ITEMPOOL_INIT(
                itempool )
```

**Value:**

```
        memset(ITEMPOOL_STATUS(itempool), \
               ITEMPOOL_ITEM_FREE, \
               ITEMPOOL_SIZE(itempool))
```

Definition at line 62 of file itempool.h.

### 4.12.2.5 ITEMPOOL_ITEM_FREE

```
#define ITEMPOOL_ITEM_FREE 0
```

Definition at line 50 of file itempool.h.

#### 4.12.2.6 ITEMPOOL_ITEM_SIZE

```
#define ITEMPOOL_ITEM_SIZE(
             itempool )   (sizeof(*ITEMPOOL_ITEMS(itempool)))
```

Definition at line 71 of file itempool.h.

#### 4.12.2.7 ITEMPOOL_ITEM_USED

```
#define ITEMPOOL_ITEM_USED 1
```

Definition at line 51 of file itempool.h.

#### 4.12.2.8 ITEMPOOL_ITEMS

```
#define ITEMPOOL_ITEMS(
             itempool )   ((itempool).items)
```

Definition at line 77 of file itempool.h.

#### 4.12.2.9 ITEMPOOL_SIZE

```
#define ITEMPOOL_SIZE(
             itempool )
```

**Value:**

```
(sizeof(ITEMPOOL_STATUS(itempool))/ \
 sizeof(*ITEMPOOL_STATUS(itempool)))
```

Definition at line 67 of file itempool.h.

#### 4.12.2.10 ITEMPOOL_STATUS

```
#define ITEMPOOL_STATUS(
             itempool )   ((itempool).status)
```

Definition at line 74 of file itempool.h.

#### 4.12.2.11 ITEMPOOL_T

```
#define ITEMPOOL_T(
             name )  name##_itempool_t
```

Definition at line 59 of file itempool.h.

#### 4.12.2.12 ITEMPOOL_TYPEDEF

```
#define ITEMPOOL_TYPEDEF(
             name,
             type,
             size )
```

**Value:**

```
    typedef struct { \
        uint8_t status[size]; \
        type items[size]; \
    }name##_itempool_t
```

Definition at line 53 of file itempool.h.

### 4.12.3 Function Documentation

#### 4.12.3.1 itempool_alloc()

```
void* itempool_alloc (
            uint8_t * items,
            uint8_t * status,
            size_t itemsize,
            size_t poolsize )
```

Definition at line 55 of file itempool.c.

Here is the caller graph for this function:

#### 4.12.3.2 itempool_calloc()

```
void* itempool_calloc (
            uint8_t * items,
            uint8_t * status,
            size_t itemsize,
            size_t poolsize )
```

Definition at line 73 of file itempool.c.

Here is the call graph for this function:

# 4.13 myos/lib/ringbuffer.h File Reference

`#include <stdlib.h>`

Include dependency graph for ringbuffer.h: This graph shows which files directly or indirectly include this file:

## Macros

- #define RINGBUFFER_TYPEDEF(name, type, size)
- #define RINGBUFFER_T(name) name##_ringbuffer_t
- #define RINGBUFFER_ITEMS(ringbuffer) ((ringbuffer).items)
- #define RINGBUFFER_RAW(ringbuffer) ((uint8_t∗)RINGBUFFER_ITEMS(ringbuffer))
- #define RINGBUFFER_SIZEOF(ringbuffer) (sizeof(RINGBUFFER_ITEMS(ringbuffer)))
- #define RINGBUFFER_SIZE(ringbuffer) (RINGBUFFER_SIZEOF(ringbuffer)/sizeof(RINGBUFFER_ITEMS(ringbuffer)[0]))
- #define RINGBUFFER_COUNT(ringbuffer) ((ringbuffer).count)
- #define RINGBUFFER_HEAD(ringbuffer) ((ringbuffer).head)
- #define RINGBUFFER_TAIL(ringbuffer) ((ringbuffer).tail)
- #define RINGBUFFER_INIT(ringbuffer)
- #define RINGBUFFER_TAIL_VAL(ringbuffer) (RINGBUFFER_ITEMS(ringbuffer)[RINGBUFFER_TAIL(ringbuffer)])
- #define RINGBUFFER_TAIL_PTR(ringbuffer) (&RINGBUFFER_TAIL_VAL(ringbuffer))
- #define RINGBUFFER_HEAD_VAL(ringbuffer) (RINGBUFFER_ITEMS(ringbuffer)[RINGBUFFER_HEAD(ringbuffer)])
- #define RINGBUFFER_HEAD_PTR(ringbuffer) (&RINGBUFFER_HEAD_VAL(ringbuffer))
- #define RINGBUFFER_PUSH(ringbuffer)
- #define RINGBUFFER_POP(ringbuffer)
- #define RINGBUFFER_FULL(ringbuffer) (RINGBUFFER_COUNT(ringbuffer)>=RINGBUFFER_SIZE(ringbuffer))
- #define RINGBUFFER_EMPTY(ringbuffer) (!RINGBUFFER_COUNT(ringbuffer))
- #define RINGBUFFER_READ(ringbufferptr, var)
- #define RINGBUFFER_WRITE(ringbufferptr, value)

### 4.13.1 Detailed Description

**Copyright**

`https://opensource.org/license/mit/`

First implementation was derived from buffer.h We decided to change it to a stand alone version to keep the memory model as flat as possible, also for weak compilers.

## 4.13.2 Macro Definition Documentation

### 4.13.2.1 RINGBUFFER_COUNT

```
#define RINGBUFFER_COUNT(
              ringbuffer )   ((ringbuffer).count)
```

Returns the current number of used items in the ringbuffer.

Definition at line 103 of file ringbuffer.h.

### 4.13.2.2 RINGBUFFER_EMPTY

```
#define RINGBUFFER_EMPTY(
              ringbuffer )   (!RINGBUFFER_COUNT(ringbuffer))
```

Checks if the ringbuffer has no items stored. Returns 1 if ringbuffer is empty, otherwise 0.

Definition at line 197 of file ringbuffer.h.

### 4.13.2.3 RINGBUFFER_FULL

```
#define RINGBUFFER_FULL(
              ringbuffer )   (RINGBUFFER_COUNT(ringbuffer)>=RINGBUFFER_SIZE(ringbuffer))
```

Checks if no further ringbuffer items are available. Returns 1 if ringbuffer is full, otherwise 0.

Definition at line 189 of file ringbuffer.h.

### 4.13.2.4 RINGBUFFER_HEAD

```
#define RINGBUFFER_HEAD(
              ringbuffer )   ((ringbuffer).head)
```

Returns the current head index.

Definition at line 109 of file ringbuffer.h.

#### 4.13.2.5 RINGBUFFER_HEAD_PTR

```
#define RINGBUFFER_HEAD_PTR(
              ringbuffer )   (&RINGBUFFER_HEAD_VAL(ringbuffer))
```

Returns a pointer to the current head item.

Definition at line 149 of file ringbuffer.h.

#### 4.13.2.6 RINGBUFFER_HEAD_VAL

```
#define RINGBUFFER_HEAD_VAL(
              ringbuffer )   (RINGBUFFER_ITEMS(ringbuffer)[RINGBUFFER_HEAD(ringbuffer)])
```

Returns a reference of the current head item.

Definition at line 143 of file ringbuffer.h.

#### 4.13.2.7 RINGBUFFER_INIT

```
#define RINGBUFFER_INIT(
              ringbuffer )
```

**Value:**

```
    do{                                 \
        RINGBUFFER_COUNT(ringbuffer)=0; \
        RINGBUFFER_HEAD(ringbuffer)=0;  \
        RINGBUFFER_TAIL(ringbuffer)=0;  \
    }while(0)
```

Initializes the ringbuffer.

Definition at line 121 of file ringbuffer.h.

#### 4.13.2.8 RINGBUFFER_ITEMS

```
#define RINGBUFFER_ITEMS(
              ringbuffer )   ((ringbuffer).items)
```

Returns a pointer to the ringbuffer items. Type of returned pointer is of ringbuffer item type.

Definition at line 72 of file ringbuffer.h.

### 4.13.2.9 RINGBUFFER_POP

```
#define RINGBUFFER_POP(
                ringbuffer )
```

**Value:**

```
    do {                                                                    \
        if(++RINGBUFFER_HEAD(ringbuffer) == RINGBUFFER_SIZE(ringbuffer))    \
        {                                                                   \
            RINGBUFFER_HEAD(ringbuffer) = 0;                                \
        }                                                                   \
                                                                            \
        RINGBUFFER_COUNT(ringbuffer)--;                                     \
    }while (0)
```

Pops the current head item in the ringbuffer, or in other words: increases the head index to address the next item in the ringbuffer. Automatically wraps around the head index in case of index overflow. Also decreases the number of items in the ringbuffer by one.

Definition at line 175 of file ringbuffer.h.

### 4.13.2.10 RINGBUFFER_PUSH

```
#define RINGBUFFER_PUSH(
                ringbuffer )
```

**Value:**

```
    do {                                                                    \
        if(++RINGBUFFER_TAIL(ringbuffer) == RINGBUFFER_SIZE(ringbuffer))    \
        {                                                                   \
            RINGBUFFER_TAIL(ringbuffer) = 0;                                \
        }                                                                   \
                                                                            \
        RINGBUFFER_COUNT(ringbuffer)++;                                     \
    }while (0)
```

Pushes the current tail item in the ringbuffer, or in other words: increases the tail index to address the next item in the ringbuffer. Automatically wraps around the tail index in case of index overflow. Also increases the number of items in the ringbuffer by one.

Definition at line 158 of file ringbuffer.h.

### 4.13.2.11 RINGBUFFER_RAW

```
#define RINGBUFFER_RAW(
                ringbuffer )   ((uint8_t*)RINGBUFFER_ITEMS(ringbuffer))
```

Returns an pointer to the ringbuffer items. Type of pointer is uint8_t (byte).

Definition at line 79 of file ringbuffer.h.

#### 4.13.2.12 RINGBUFFER_READ

```
#define RINGBUFFER_READ(
              ringbufferptr,
              var )
```

**Value:**

```
    do {                                        \
        var = RINGBUFFER_HEAD_VAL(ringbufferptr);   \
        RINGBUFFER_POP(ringbufferptr);              \
    }while(0)
```

Reads one item into /a var by value. Automatically pops the ringbuffer. Read oldest element. App must ensure !RINGBUFFER_EMPTY() first.

Definition at line 206 of file ringbuffer.h.

#### 4.13.2.13 RINGBUFFER_SIZE

```
#define RINGBUFFER_SIZE(
              ringbuffer )   (RINGBUFFER_SIZEOF(ringbuffer)/sizeof(RINGBUFFER_ITEMS(ringbuffer)[0]))
```

Returns the maximum number of items the ringbuffer can hold.

Definition at line 93 of file ringbuffer.h.

#### 4.13.2.14 RINGBUFFER_SIZEOF

```
#define RINGBUFFER_SIZEOF(
              ringbuffer )   (sizeof(RINGBUFFER_ITEMS(ringbuffer)))
```

Returns the size of the ringbuffer in bytes.

Definition at line 86 of file ringbuffer.h.

#### 4.13.2.15 RINGBUFFER_T

```
#define RINGBUFFER_T(
              name )   name##_ringbuffer_t
```

Defines a ringbuffer of type *name*. Can also be used with different storage classifiers. (static, extern, volatile ...)

Definition at line 65 of file ringbuffer.h.

### 4.13.2.16 RINGBUFFER_TAIL

```
#define RINGBUFFER_TAIL(
              ringbuffer )   ((ringbuffer).tail)
```

Returns the current tail index.

Definition at line 115 of file ringbuffer.h.

### 4.13.2.17 RINGBUFFER_TAIL_PTR

```
#define RINGBUFFER_TAIL_PTR(
              ringbuffer )   (&RINGBUFFER_TAIL_VAL(ringbuffer))
```

Returns a pointer to the current tail item.

Definition at line 137 of file ringbuffer.h.

### 4.13.2.18 RINGBUFFER_TAIL_VAL

```
#define RINGBUFFER_TAIL_VAL(
              ringbuffer )   (RINGBUFFER_ITEMS(ringbuffer)[RINGBUFFER_TAIL(ringbuffer)])
```

Returns a reference of the current tail item.

Definition at line 131 of file ringbuffer.h.

### 4.13.2.19 RINGBUFFER_TYPEDEF

```
#define RINGBUFFER_TYPEDEF(
              name,
              type,
              size )
```

**Value:**

```
    typedef struct {                        \
        size_t head;                        \
        size_t tail;                        \
        size_t count;                       \
        type items [size];                  \
    } name##_ringbuffer_t
```

Declares a ringbuffer type. The argumnents are *name* for a unique identification of the ringbuffer type, the *type* of the items and the number of items of *type* the ringbuffer can hold (*size*).

Definition at line 52 of file ringbuffer.h.

#### 4.13.2.20 RINGBUFFER_WRITE

```
#define RINGBUFFER_WRITE(
                ringbufferptr,
                value )
```

**Value:**

```
    do {                                                    \
        RINGBUFFER_TAIL_VAL(ringbufferptr) = value;         \
        RINGBUFFER_PUSH(ringbufferptr);                     \
    }while(0)
```

Writes one item into ringbuffer by value. Automatically pushes the ringbuffer. Overwrites the oldest element if ringbuffer is full. App can choose to avoid the overwrite by checking RINGBUFFER_FULL().

Definition at line 218 of file ringbuffer.h.

## 4.14 myos/lib/slist.c File Reference

Circular singly linked list.

```
#include <stdlib.h>
#include "slist.h"
```

Include dependency graph for slist.c:

### Functions

- slist_node_t ∗ slist_find (slist_t ∗slist, void ∗node)

  *Checks if a node is in the list.*
- slist_node_t ∗ slist_back (slist_t ∗slist)

  *Returns last node.*
- slist_node_t ∗ slist_prev_prev (slist_t ∗slist, void ∗node)
- slist_node_t ∗ slist_prev (slist_t ∗slist, void ∗node)

  *Get previous list node.*
- size_t slist_size (slist_t ∗slist)

  *Counts number of nodes nb list.*

### 4.14.1 Detailed Description

Circular singly linked list.

In computer science, a linked list is a linear collection of data elements, whose order is not given by their physical placement in memory. Instead, each element points to the next. It is a data structure consisting of a collection of nodes which together represent a sequence. In its most basic form, each node contains: data, and a reference (in other words, a link) to the next node in the sequence. This structure allows for efficient insertion or removal of elements from any position in the sequence during iteration. [ https://en.wikipedia.org/wiki/Linked_list]

Singly linked lists contain nodes which have a data field as well as 'next' field, which points to the next node in line of nodes. Operations that can be performed on singly linked lists include insertion, deletion and traversal. [ https://en.wikipedia.org/wiki/Linked_list#Singly_linked_list]

In the last node of a list, the link field often contains a null reference, a special value is used to indicate the lack of further nodes. A less common convention is to make it point to the first node of the list; in that case, the list is said to be 'circular' or 'circularly linked'; otherwise, it is said to be 'open' or 'linear'. It is a list where the last pointer points to the first node. [ https://en.wikipedia.org/wiki/Linked_list#Circular_linked_list]

## 4.14.2 Function Documentation

### 4.14.2.1 slist_back()

```
slist_node_t* slist_back (
            slist_t * slist )
```

Returns last node.

This function returns an iterator on the last node in list.

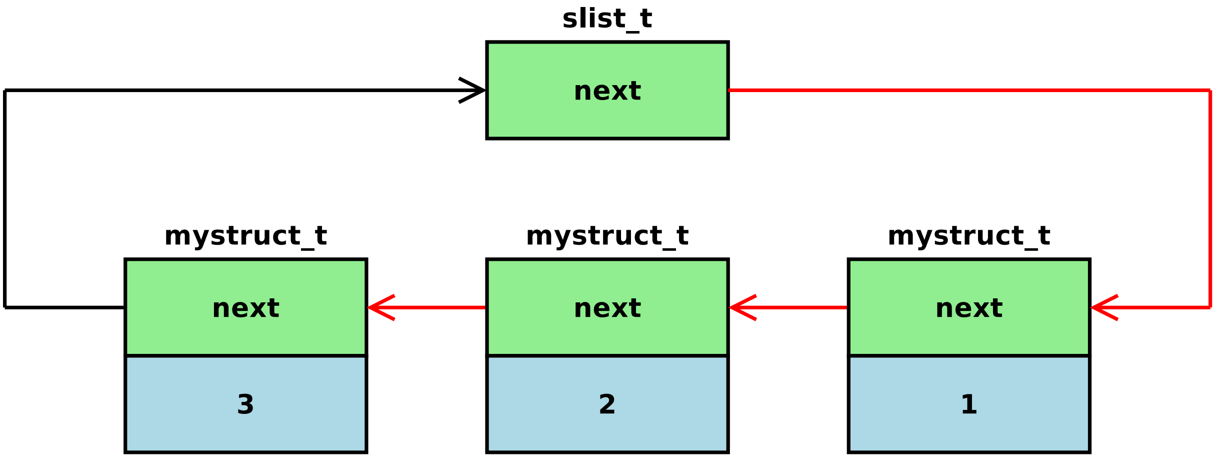


Figure 4.1 slist_back

**Parameters**

| | |
|---|---|
| *slist* | List |

**Returns**

Iterator on last node in list

Definition at line 71 of file slist.c.

### 4.14.2.2 slist_find()

```
slist_node_t* slist_find (
            slist_t * slist,
            void * node )
```

Checks if a node is in the list.

Iterates through list and checks if a specific node is in the list

**Parameters**

| | |
|---|---|
| *slist* | List |
| *node* | Node to search for |

**Returns**

Pointer to node if it exist, NULL otherwise

Definition at line 56 of file slist.c.

#### 4.14.2.3 slist_prev()

```
slist_node_t* slist_prev (
            slist_t * slist,
            void * existing_node )
```

Get previous list node.

**Parameters**

| | |
|---|---|
| *slist* | List |
| *node* | Current node |

**Returns**

Predecessor of current node

Definition at line 100 of file slist.c.

#### 4.14.2.4 slist_prev_prev()

```
slist_node_t* slist_prev_prev (
            slist_t * slist,
            void * node )
```

Definition at line 85 of file slist.c.

#### 4.14.2.5 slist_size()

```
size_t slist_size (
            slist_t * slist )
```

Counts number of nodes nb list.

Returns the number of elements in the list container. List node itself is not counted.

**Parameters**

| | |
|---|---|
| *slist* | List |

**Returns**

Returns the number of elements in the list container.

Definition at line 113 of file slist.c.

## 4.15 myos/lib/slist.h File Reference

Circular singly linked list.

```
#include <stddef.h>
#include <stdbool.h>
```

Include dependency graph for slist.h: This graph shows which files directly or indirectly include this file:

### Data Structures

- struct slist_node_t

### Macros

- #define SLIST_NODE_TYPE slist_node_t slist_node
- #define slist_next(slist, node) (((slist_node_t∗)(node))->next)

  *Get successor node.*
- #define slist_init(slist) do{slist_next(slist,slist) = (slist);}while(0)

  *Initialize list.*
- #define slist_begin(slist) ((slist)->next)

  *Return iterator to beginning.*
- #define slist_front(slist) (slist_begin(slist))

  *Return iterator to beginning.*
- #define slist_end(slist) (slist)

  *Return iterator to end.*
- #define slist_foreach(slist, iterator)

  *Iterate through whole list.*
- #define slist_empty(slist) ((slist)->next==(slist))

  *Test whether container is empty.*
- #define slist_push_front(slist, node_to_add)

  *Insert element at beginning.*
- #define slist_pop_front(slist) do{(slist)->next = (slist)->next->next;}while(0)

  *Delete first node.*
- #define slist_push_back(slist, node_to_add)

  *Insert element at end.*
- #define slist_insert_after(slistptr, posptr, nodeptr)

  *Insert elements after position.*
- #define slist_insert_before(slist, existing_node, node_to_insert)

  *Insert elements before position.*
- #define slist_clear(slistptr) slist_init(slistptr)

  *Clear list.*
- #define slist_erase(slistptr, nodeptr)

  *Removes a specific node from list.*
- #define slist_pop_back(slist)

  *Delete last element.*

## Typedefs

- typedef struct slist_node_t slist_node_t

  *Singly linked list node type.*
- typedef slist_node_t slist_t

## Functions

- slist_node_t ∗ slist_prev (slist_t ∗slist, void ∗existing_node)

  *Get previous list node.*
- size_t slist_size (slist_t ∗slist)

  *Counts number of nodes nb list.*
- slist_node_t ∗ slist_back (slist_t ∗slist)

  *Returns last node.*
- slist_node_t ∗ slist_prev_prev (slist_t ∗slist, void ∗node)
- slist_node_t ∗ slist_find (slist_t ∗slist, void ∗node)

  *Checks if a node is in the list.*

### 4.15.1 Detailed Description

Circular singly linked list.

**Copyright**

**License:**

In computer science, a linked list is a linear collection of data elements, whose order is not given by their physical placement in memory. Instead, each element points to the next. It is a data structure consisting of a collection of nodes which together represent a sequence. In its most basic form, each node contains: data, and a reference (in other words, a link) to the next node in the sequence. This structure allows for efficient insertion or removal of elements from any position in the sequence during iteration.

https://en.wikipedia.org/wiki/Linked_list

Singly linked lists contain nodes which have a data field as well as 'next' field, which points to the next node in line of nodes. Operations that can be performed on singly linked lists include insertion, deletion and traversal.

https://en.wikipedia.org/wiki/Linked_list#Singly_linked_list

In the last node of a list, the link field often contains a null reference, a special value is used to indicate the lack of further nodes. A less common convention is to make it point to the first node of the list; in that case, the list is said to be 'circular' or 'circularly linked'; otherwise, it is said to be 'open' or 'linear'. It is a list where the last pointer points to the first node.

https://en.wikipedia.org/wiki/Linked_list#Circular_linked_list

Comparison Singly Linked List vs. Arrays :

| Arrays | Singly Linked List |
|---|---|
| Arrays are stored in continuous location. | Linked lists are not stored in contiguous location. |
| Fixed size. | Dynamic in size. |
| Memory is allocated at compile time. | Memory is allocated at run time (not necessarily true). |
| Uses less memory than linked lists. | Uses more memory because it stores both data and the address of the next node. |
| Elements can be accessed easily. | Insertion and deletion operation is faster. |

Nomenclature (applies for slist and dlist) :

{html: width=640px, latex: width=5cm}

## 4.15.2 Macro Definition Documentation

### 4.15.2.1 slist_begin

```
#define slist_begin(
              slist ) ((slist)->next)
```

Return iterator to beginning.

Returns an iterator pointing to the first element in the list container. If the container is empty, the returned iterator value shall not be dereferenced.

**Parameters**

| | |
|---|---|
| *slist* | List |

**Returns**

Pointer to the first node or on the list itself.

Definition at line 120 of file slist.h.

### 4.15.2.2 slist_clear

```
#define slist_clear(
              slistptr )  slist_init(slistptr)
```

Clear list.

Removes all elements from the list container, and leaving the container with a size of 0.

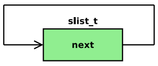


**Figure 4.2 slist_clear**

**Parameters**

| | |
|---|---|
| *slistptr* | List |

Definition at line 264 of file slist.h.

#### 4.15.2.3 slist_empty

```
#define slist_empty(
             slist ) ((slist)->next==(slist))
```

Test whether container is empty.

Returns whether the list container is empty or not.

**Parameters**

| | |
|---|---|
| *slistptr* | List |

**Returns**

0 if it is not empty or 1 if it is empty

Definition at line 171 of file slist.h.

#### 4.15.2.4 slist_end

```
#define slist_end(
             slist ) (slist)
```

Return iterator to end.

Returns an iterator referring to the past-the-end element in the list container. It does not point to any element, and thus shall not be dereferenced. This function is often used in combination with list_begin to specify a range including all the elements in the container. As the Successor of the last element is the list itself, it returns the list itself.

**Parameters**

| | |
|---|---|
| *slist* | List |

**Returns**

Iterator referring to the past-the-end element

Definition at line 147 of file slist.h.

#### 4.15.2.5 slist_erase

```
#define slist_erase(
              slistptr,
              nodeptr )
```

**Value:**

```
    do{                                                                             \
        slist_prev(slistptr,nodeptr)->next = ((slist_node_t*)(nodeptr))->next;      \
    }while(0)
```

Removes a specific node from list.

Removes a specific node from list. Node has to be a member of the list, otherwise erasing the node results in undefined behavior

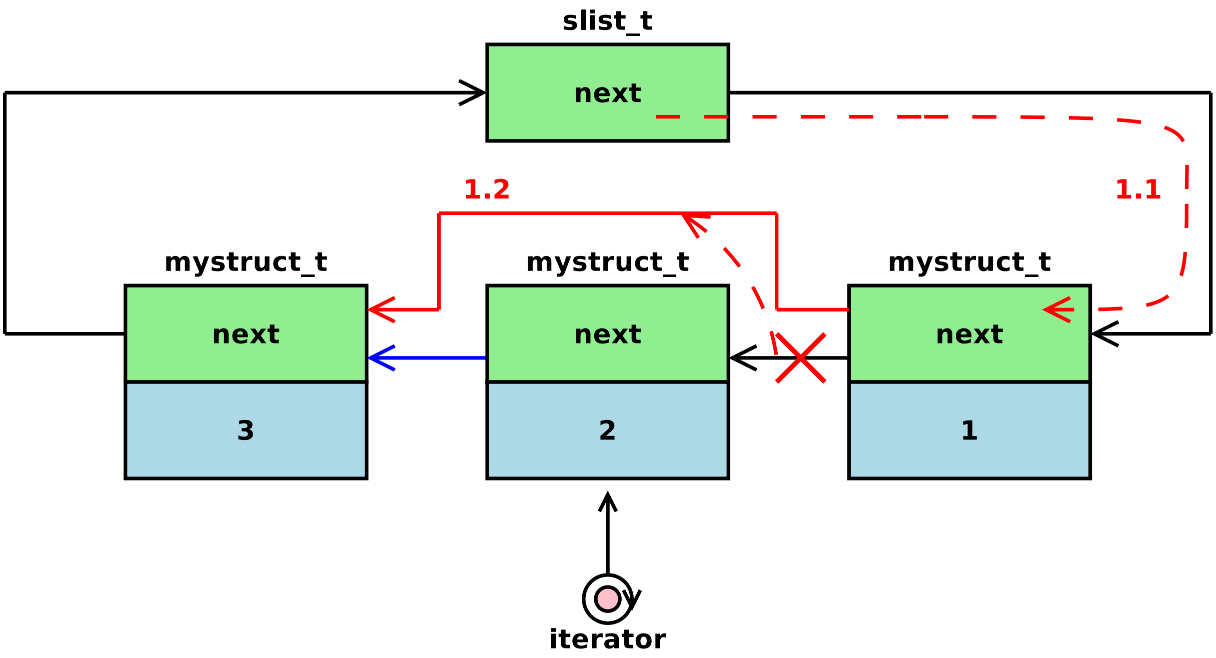


**Figure 4.3 slist_erase**

**Parameters**

| | |
|---|---|
| *slistptr* | List |
| *nodeptr* | Node to remove from list |

Definition at line 278 of file slist.h.

#### 4.15.2.6 slist_foreach

```
#define slist_foreach(
              slist,
              iterator )
```

**Value:**

```
    for(iterator = (void*)slist_begin(slist);          \
        ((slist_node_t*)iterator) != slist_end(slist);  \
        iterator=(void*)slist_next(slist,iterator))
```

Iterate through whole list.

Iterates from first to last element in the list container.

**Parameters**

| | |
|---|---|
| *slistptr* | List |
| *iterator* | |

Definition at line 157 of file slist.h.

### 4.15.2.7 slist_front

```
#define slist_front(
              slist ) (slist_begin(slist))
```

Return iterator to beginning.

Returns an iterator pointing to the first element in the list container. If the container is empty, the returned iterator value shall not be dereferenced.

**Parameters**

| | |
|---|---|
| *slist* | List |

**Returns**

Pointer to the first node or on the list itself.

Definition at line 131 of file slist.h.

### 4.15.2.8 slist_init

```
#define slist_init(
              slist )  do{slist_next(slist,slist) = (slist);}while(0)
```

Initialize list.

An empty list only contains the list instance pointing to itself. It is important not to forget to initialize the list, otherwise there may be undefined behavior with list functions and function like macros.

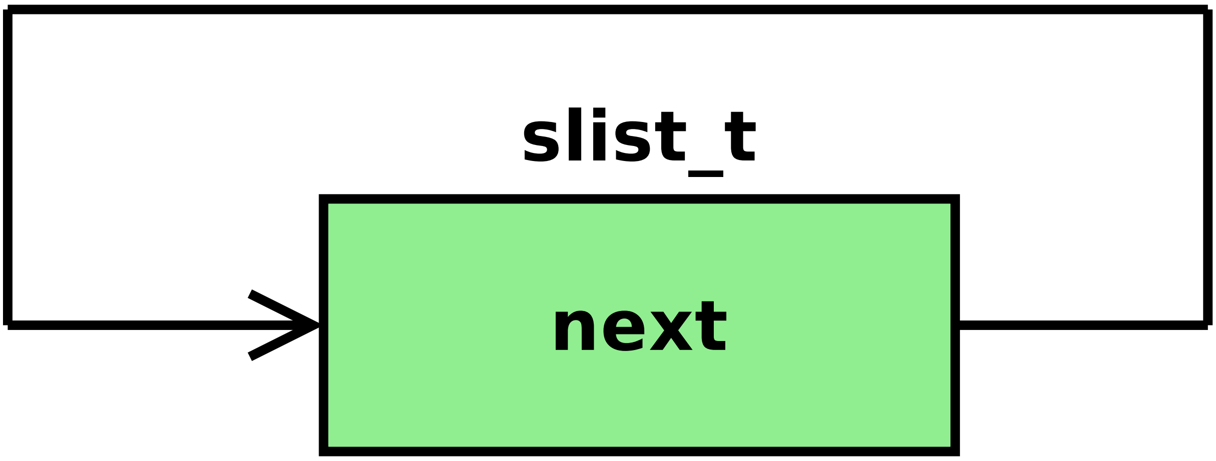


Figure 4.4 slist_init

**Parameters**

| | |
|---|---|
| *slist* | List to initialize |

Definition at line 108 of file slist.h.

### 4.15.2.9 slist_insert_after

```
#define slist_insert_after(
               slistptr,
               posptr,
               nodeptr )
```

**Value:**

```
    do{ \
        ((slist_node_t*)(nodeptr))->next = ((slist_node_t*)(posptr))->next; \
        ((slist_node_t*)(posptr))->next = ((slist_node_t*)(nodeptr)); \
    }while(0)
```

Insert elements after position.

The container is extended by inserting a new element after the element at the specified position.

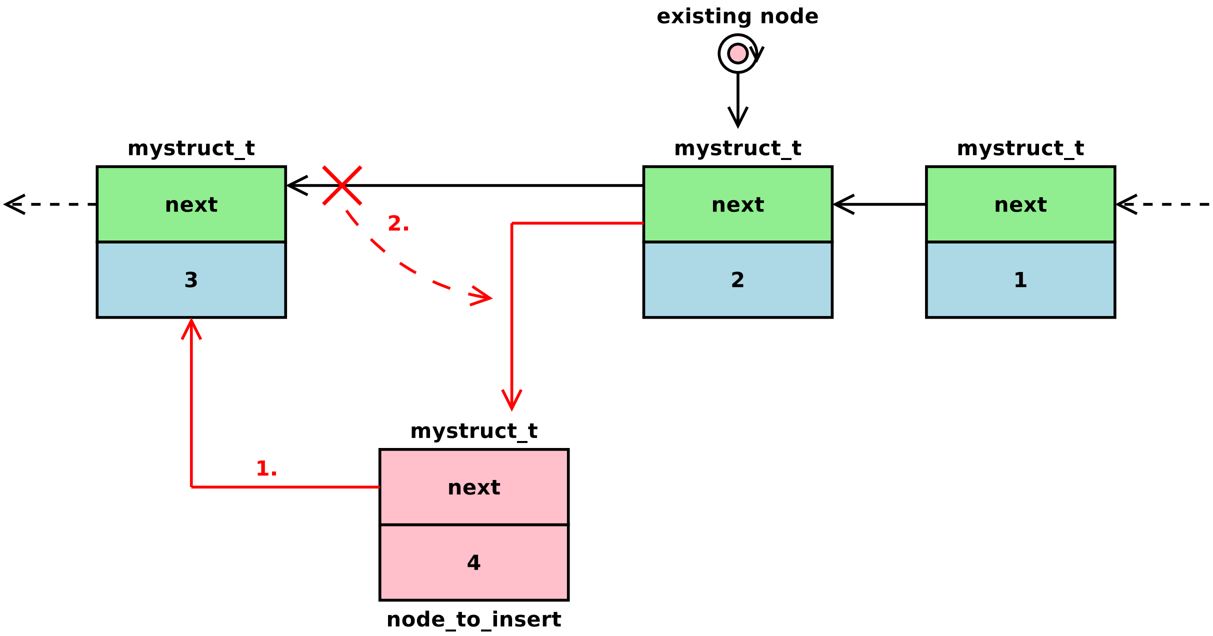


Figure 4.5 slist_insert_after

**Parameters**

| | |
|---|---|
| *slistptr* | List \Param nodeptr Node to add to the list |

Definition at line 231 of file slist.h.

### 4.15.2.10 slist_insert_before

```
#define slist_insert_before(
                slist,
                existing_node,
                node_to_insert )
```

**Value:**

```
    do{ \
        slist_prev(slist,existing_node)->next = (slist_node_t*)(node_to_insert); \
        ((slist_node_t*)node_to_insert))->next = existing_node; \
    }while(0)
```

Insert elements before position.

The container is extended by inserting a new element before the element at the specified position.

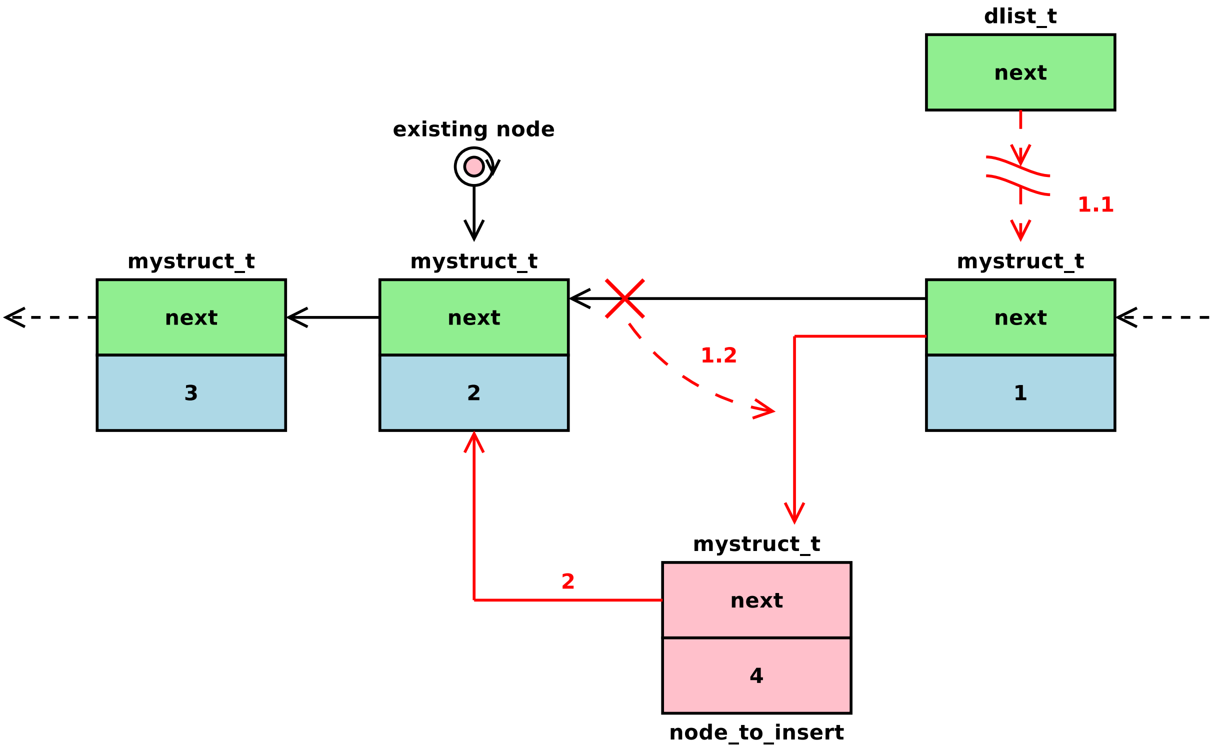


**Figure 4.6 slist_insert_before**

**Parameters**

| | |
|---|---|
| *slist* | List \Param node_to_insert Node to add to the list |

Definition at line 248 of file slist.h.

### 4.15.2.11 slist_next

```
#define slist_next(
                slist,
                node )   (((slist_node_t*)(node))->next)
```

Get successor node.

{html: width=20%}

**Parameters**

| | |
|---|---|
| *slist* | List |
| *node* | Current node |

**Returns**

Successor of current node

Definition at line 95 of file slist.h.

#### 4.15.2.12 SLIST_NODE_TYPE

```
#define SLIST_NODE_TYPE slist_node_t slist_node
```

Add SLIST_NODE_TYPE as first member of a structure to make it a list node

Definition at line 84 of file slist.h.

#### 4.15.2.13 slist_pop_back

```
#define slist_pop_back(
            slist )
```

**Value:**

```
do{                                                    \
    slist_prev_prev(slist,slist)->next = (slist);      \
}while(0)
```

Delete last element.

Removes the last element in the list container, effectively reducing the container size by one.

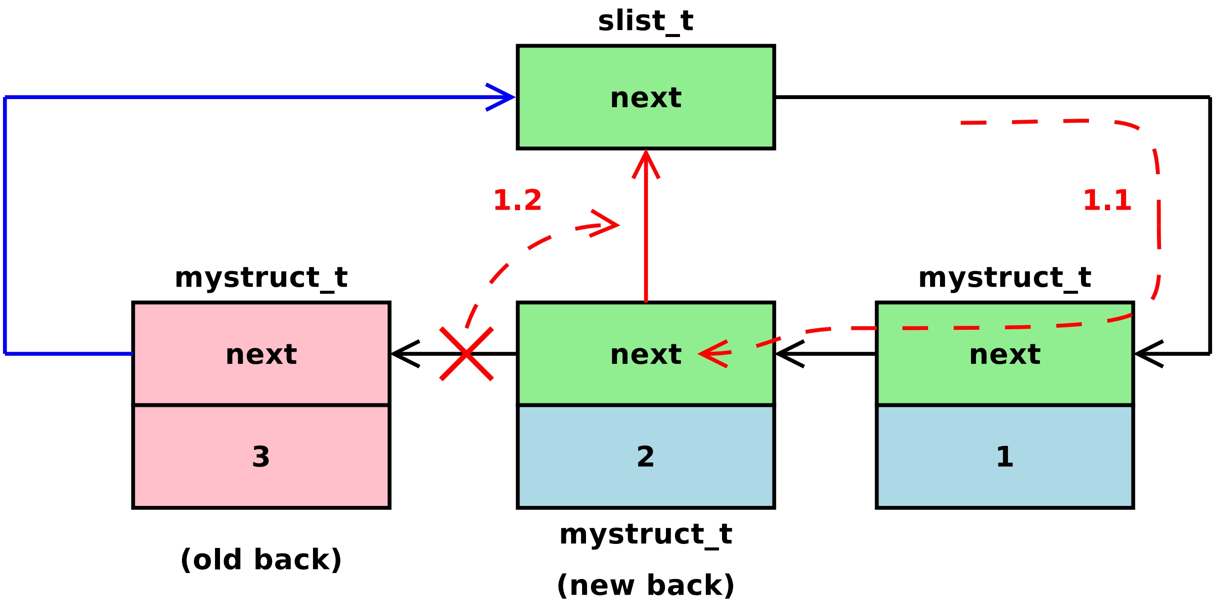


Figure 4.7 slist_pop_back

**Parameters**

| | |
|---|---|
| *slist* | List |

Definition at line 349 of file slist.h.

### 4.15.2.14 slist_pop_front

```
#define slist_pop_front(
              slist )  do{(slist)->next = (slist)->next->next;}while(0)
```

Delete first node.

Removes the first node of the list container.

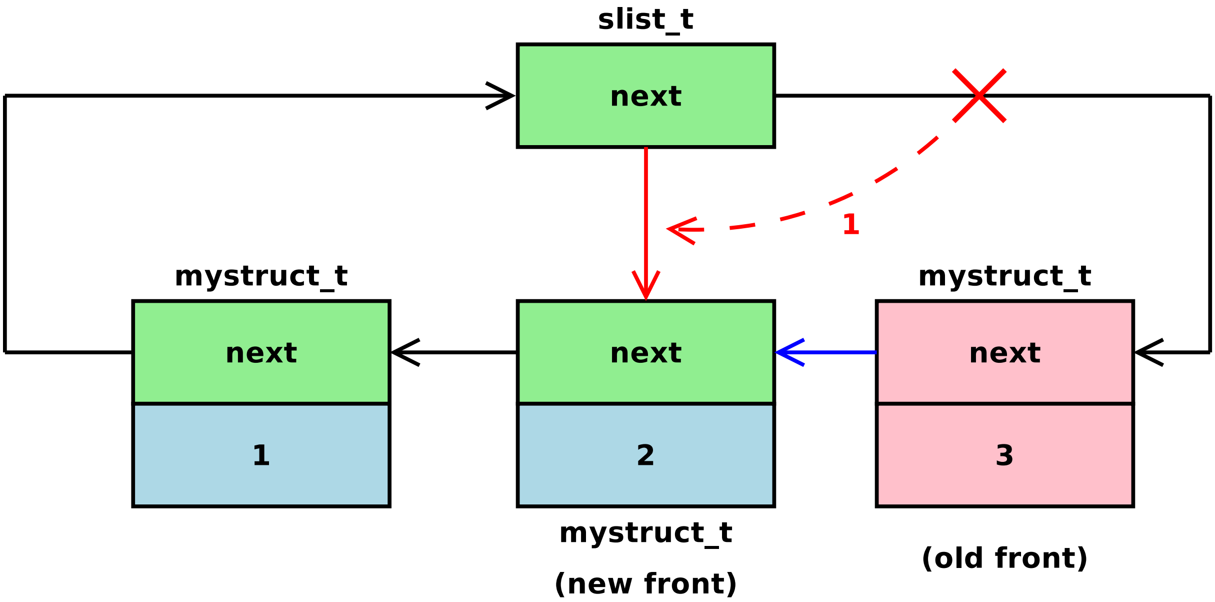


**Figure 4.8 slist_pop_front**

**Parameters**

| | |
|---|---|
| *slist* | List |

Definition at line 200 of file slist.h.

### 4.15.2.15 slist_push_back

```
#define slist_push_back(
              slist,
              node_to_add )
```

**Value:**

```
    do { \
        ((slist_node_t*)(node_to_add))->next = slist_back(slist); \
        ((slist_node_t*)(node_to_add))->next->next = ((slist_node_t*)(node_to_add)); \
        ((slist_node_t*)(node_to_add))->next = slist_end(slist);\
    }while(0)
```

Insert element at end.

Inserts a new node at the end of the list, right after the current back node.

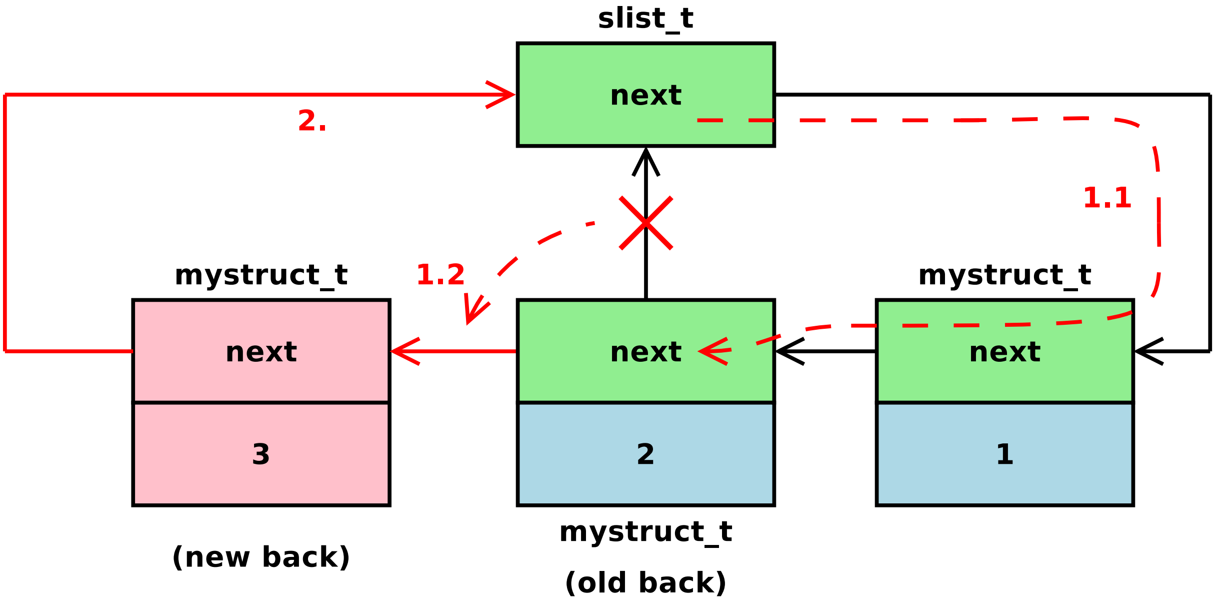


Figure 4.9 slist_push_back

**Parameters**

| | |
|---|---|
| *slistptr* | List |
| *nodeptr* | Node to add to the list |

Definition at line 212 of file slist.h.

#### 4.15.2.16 slist_push_front

```
#define slist_push_front(
                slist,
                node_to_add )
```

**Value:**

```
do{ \
    ((slist_node_t*)(node_to_add))->next = (slist)->next; \
    (slist)->next=((slist_node_t*)(node_to_add)); \
}while(0);
```

Insert element at beginning.

Inserts a new node at the beginning of the list, right before its current first element.

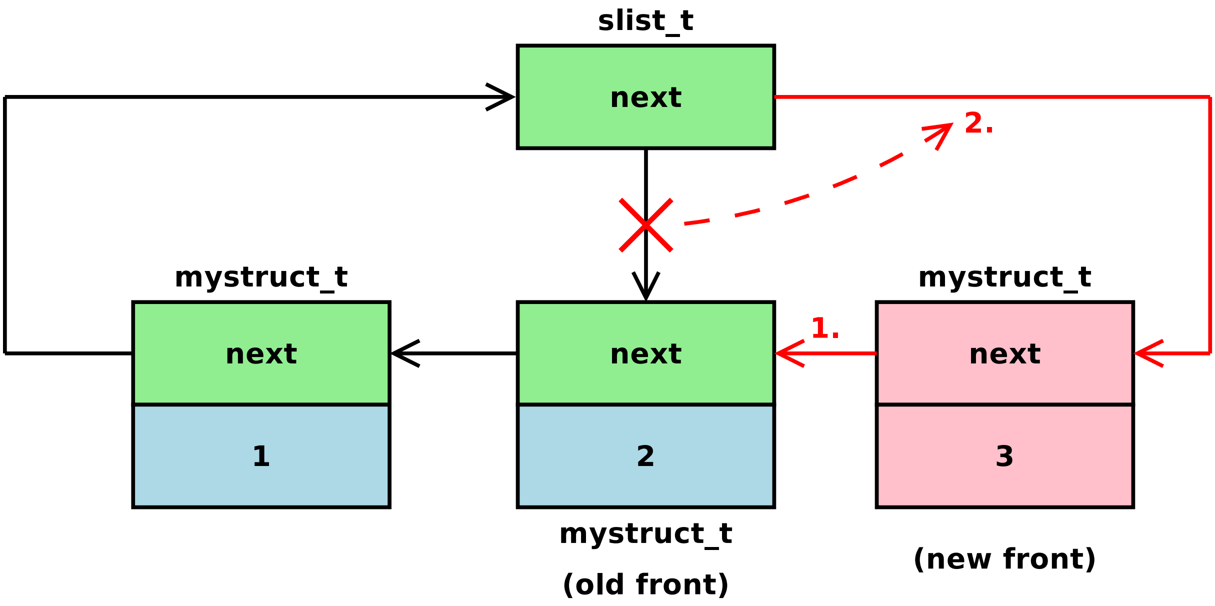


Figure 4.10 slist_push_front

**Parameters**

| | |
|---|---|
| *slist* | List |
| *nodeptr* | Node to add to the list |

Definition at line 185 of file slist.h.

## 4.15.3 Typedef Documentation

### 4.15.3.1 slist_node_t

```
typedef struct slist_node_t slist_node_t
```

Singly linked list node type.

This structure represents a node in a singly linked list, containing a pointer to the next node in the sequence.

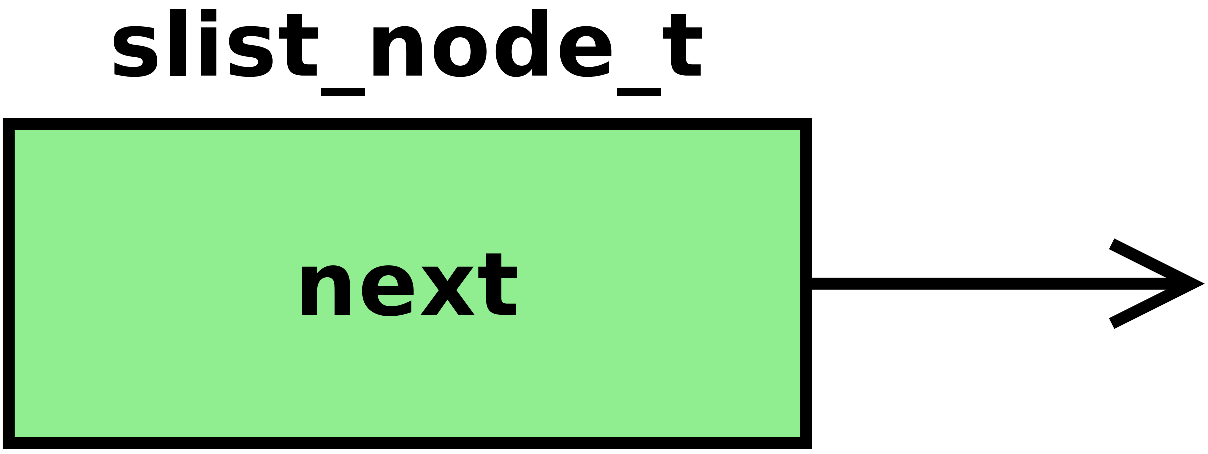


Figure 4.11 slist_node_t

Definition at line 1 of file slist.h.

#### 4.15.3.2 slist_t

```
typedef slist_node_t slist_t
```

List instance is also a member of the list

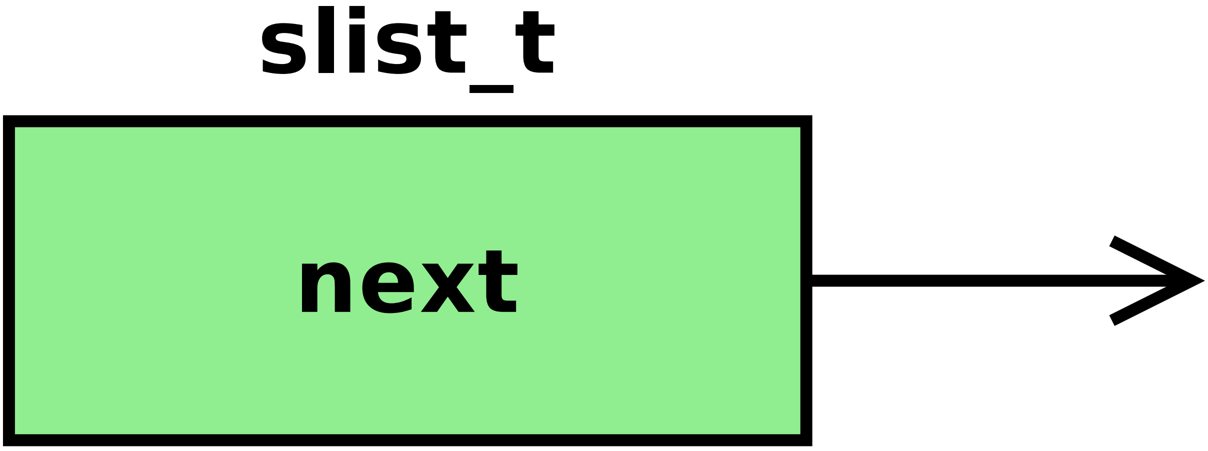


Figure 4.12 slist_t

List instance is also a member of the list

Definition at line 301 of file slist.h.

### 4.15.4 Function Documentation

#### 4.15.4.1 slist_back()

```
slist_node_t* slist_back (
            slist_t * slist )
```

Returns last node.

This function returns an iterator on the last node in list.

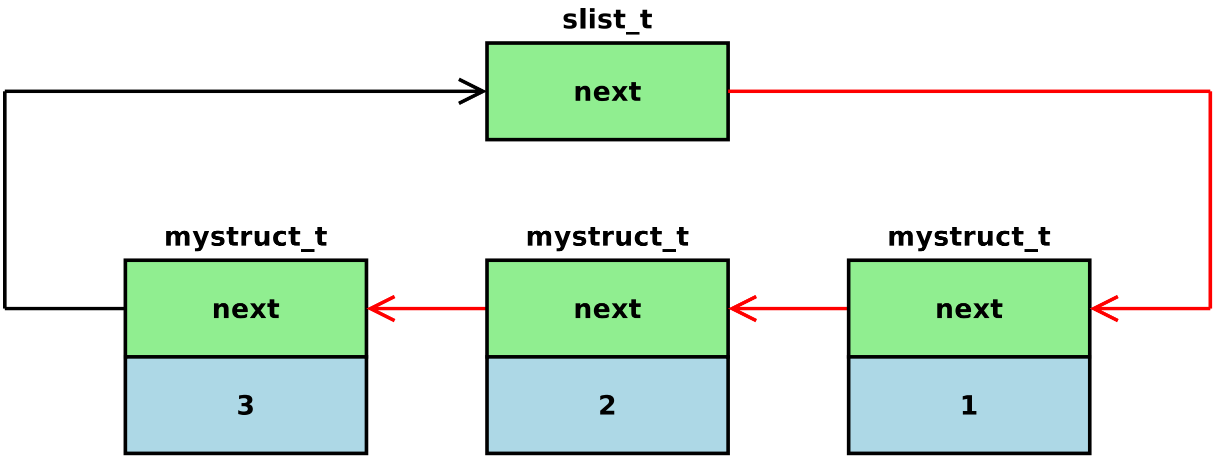


Figure 4.13 slist_back

**Parameters**

| | |
|---|---|
| *slist* | List |

**Returns**

Iterator on last node in list

Definition at line 71 of file slist.c.

### 4.15.4.2 slist_find()

```
slist_node_t* slist_find (
            slist_t * slist,
            void * node )
```

Checks if a node is in the list.

Iterates through list and checks if a specific node is in the list

**Parameters**

| | |
|---|---|
| *slist* | List |
| *node* | Node to search for |

**Returns**

Pointer to node if it exist, NULL otherwise

Definition at line 56 of file slist.c.

#### 4.15.4.3 slist_prev()

```
slist_node_t* slist_prev (
             slist_t * slist,
             void * existing_node )
```

Get previous list node.

**Parameters**

| | |
|---|---|
| *slist* | List |
| *node* | Current node |

**Returns**

Predecessor of current node

Definition at line 100 of file slist.c.

#### 4.15.4.4 slist_prev_prev()

```
slist_node_t* slist_prev_prev (
             slist_t * slist,
             void * node )
```

Definition at line 85 of file slist.c.

#### 4.15.4.5 slist_size()

```
size_t slist_size (
             slist_t * slist )
```

Counts number of nodes nb list.

Returns the number of elements in the list container. List node itself is not counted.

**Parameters**

| | |
|---|---|
| *slist* | List |

**Returns**

Returns the number of elements in the list container.

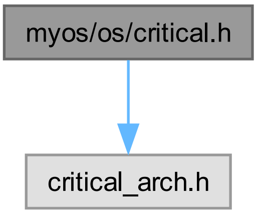
Definition at line 113 of file slist.c.

## 4.16 myos/os/critical.h File Reference

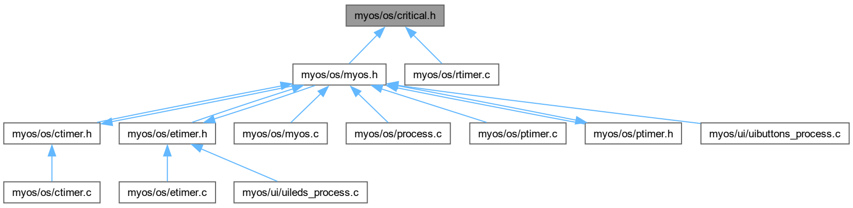
Critical section handling.

`#include "critical_arch.h"`

Include dependency graph for critical.h:



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



### Macros

- #define CRITICAL_SECTION_BEGIN CRITICAL_ARCH_SECTION_BEGIN

  *Begin of a critical section in code.*
- #define CRITICAL_SECTION_END CRITICAL_ARCH_SECTION_END

  *End of a critical section in code.*
- #define CRITICAL_STATEMENT(x)

  *Critical section for one statement.*

### 4.16.1 Detailed Description

Critical section handling.

**Copyright**

Copyright 2013-2023 Marco Bacchi `marco@bacchi.at`

**License:**

This file is released under the MIT License.
https://opensource.org/license/mit/

In concurrent programming, concurrent accesses to shared resources can lead to unexpected or erroneous behavior, so parts of the program where the shared resource is accessed need to be protected in ways that avoid the concurrent access.

One way to do so is known as a critical section or critical region. This protected section cannot be entered by more than one process or thread at a time; others are suspended until the first leaves the critical section. Typically, the critical section accesses a shared resource, such as a data structure, a peripheral device, or a network connection, that would not operate correctly in the context of multiple concurrent accesses.

The simplest method to prevent any change of processor control inside the critical section is implementing a semaphore. In uniprocessor systems, this can be done by disabling interrupts on entry into the critical section, avoiding system calls that can cause a context switch while inside the section, and restoring interrupts to their previous state on exit.

https://en.wikipedia.org/wiki/Critical_section

Critical sections should be kept as short as possible to minimize the time during which interrupts are disabled. This helps reduce the impact on the overall system responsiveness.

## 4.16.2 Macro Definition Documentation

### 4.16.2.1 CRITICAL_SECTION_BEGIN

```
#define CRITICAL_SECTION_BEGIN CRITICAL_ARCH_SECTION_BEGIN
```

Begin of a critical section in code.

Inside a critical section, it's common to disable interrupts temporarily to prevent other interrupt sources from preempting the current ISR. This ensures that the critical section is executed atomically without interruption.

This macro disables the interrupts before entering the critical section.

Definition at line 76 of file critical.h.

### 4.16.2.2 CRITICAL_SECTION_END

```
#define CRITICAL_SECTION_END CRITICAL_ARCH_SECTION_END
```

End of a critical section in code.

Once the critical section is completed, interrupts are re-enabled to allow the processor to respond to other interrupt sources.

This macro enables the interrupts before leaving the critical section.

Definition at line 85 of file critical.h.

### 4.16.2.3 CRITICAL_STATEMENT

```
#define CRITICAL_STATEMENT(
              x )
```

**Value:**

```
CRITICAL_SECTION_BEGIN();  \
x;                         \
CRITICAL_SECTION_END();
```

Critical section for one statement.

In computer programming, a statement is a syntactic unit of an imperative programming language that expresses some action to be carried out.

Simple statements are complete in themselves; these include assignments, subroutine calls, and a few statements which may significantly affect the program flow of control (e.g. goto, return, stop/halt).

https://en.wikipedia.org/wiki/Statement_(computer_science)

This macro disables interrupts before entering the critical section, executes the statement provided and enables interrupts again before leaving the critical section.

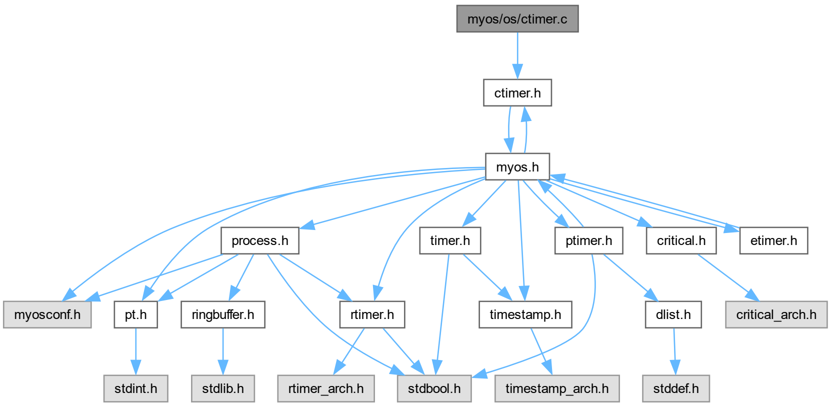
Usage example : CRITICAL_STATEMENT( a = b );

Definition at line 106 of file critical.h.

## 4.17 myos/os/ctimer.c File Reference

`#include "ctimer.h"`

Include dependency graph for ctimer.c:



## Functions

- void ctimer_start (ctimer_t ∗ctimer, timespan_t span, ctimer_callback_t callback, void ∗data)

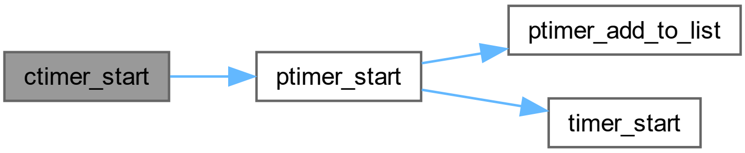
### 4.17.1 Function Documentation

#### 4.17.1.1 ctimer_start()

```
void ctimer_start (
            ctimer_t * ctimer,
            timespan_t span,
            ctimer_callback_t callback,
            void * data )
```

Definition at line 57 of file ctimer.c.

Here is the call graph for this function:

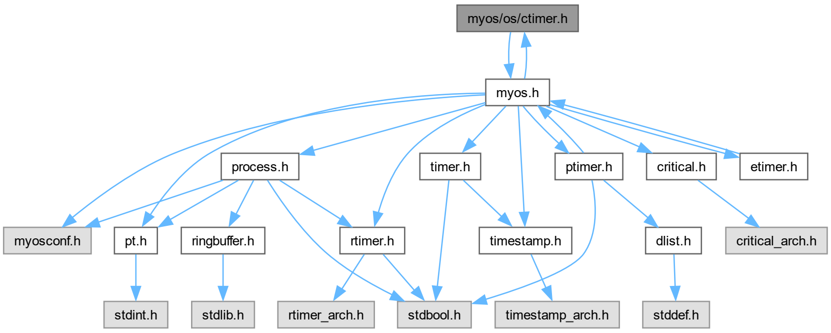


## 4.18 myos/os/ctimer.h File Reference

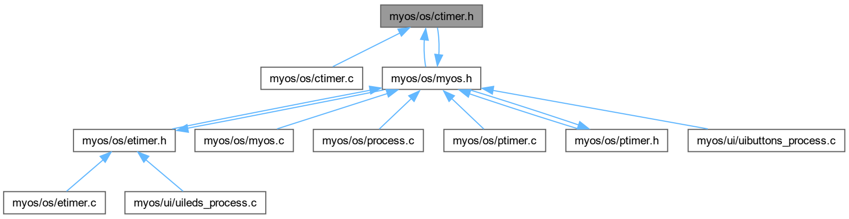
Callback timer implementation (ctimer)

```
#include "myos.h"
```

Include dependency graph for ctimer.h:



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



## Data Structures

- struct ctimer_t

## Macros

- #define ctimer_module_init() ptimer_module_init()
- #define ctimer_restart(ctimerptr) ptimer_restart((ptimer_t∗)ctimerptr)
- #define ctimer_reset(ctimerptr) ptimer_rreset((ptimer_t∗)ctimerptr)
- #define ctimer_stop(ctimerptr) ptimer_stop((ptimer_t∗)ctimerptr)
- #define ctimer_expired(ctimerptr) ptimer_expired((ptimer_t∗)ctimerptr)

## Typedefs

- typedef struct ctimer_t ctimer_t
- typedef void(∗ ctimer_callback_t) (ctimer_t ∗ctimer)

## Functions

- void ctimer_start (ctimer_t ∗ctimer, timespan_t span, ctimer_callback_t callback, void ∗data)

## 4.18.1 Detailed Description

Callback timer implementation (ctimer)

**Copyright**

Copyright (c) 2006, Swedish Institute of Computer Science.

**License:**

This file is released under the 3-Clause BSD License. `https://opensource.org/license/bsd-3-clause/`

In computer programming, a callback or callback function is any reference to executable code that is passed as an argument to another piece of code; that code is expected to call back (execute) the callback function as part of its job. This execution may be immediate as in a synchronous callback, or it might happen at a later point in time as in an asynchronous callback. They are also called blocking and non-blocking.

`https://en.wikipedia.org/wiki/Callback_(computer_programming)`

Callback timer callbacks are asynchronous callbacks.

The callback function gets invoked in the context of the process that started the callback timer.

## 4.18.2 Macro Definition Documentation

### 4.18.2.1 ctimer_expired

```
#define ctimer_expired(
             ctimerptr ) ptimer_expired((ptimer_t*)ctimerptr)
```

Definition at line 80 of file ctimer.h.

### 4.18.2.2 ctimer_module_init

```
#define ctimer_module_init( ) ptimer_module_init()
```

Definition at line 76 of file ctimer.h.

### 4.18.2.3 ctimer_reset

```
#define ctimer_reset(
             ctimerptr ) ptimer_rreset((ptimer_t*)ctimerptr)
```

Definition at line 78 of file ctimer.h.

### 4.18.2.4 ctimer_restart

```
#define ctimer_restart(
             ctimerptr ) ptimer_restart((ptimer_t*)ctimerptr)
```

Definition at line 77 of file ctimer.h.

### 4.18.2.5 ctimer_stop

```
#define ctimer_stop(
             ctimerptr ) ptimer_stop((ptimer_t*)ctimerptr)
```

Definition at line 79 of file ctimer.h.

## 4.18.3 Typedef Documentation

### 4.18.3.1 ctimer_callback_t

```
typedef void(* ctimer_callback_t) (ctimer_t *ctimer)
```

Definition at line 66 of file ctimer.h.

### 4.18.3.2 ctimer_t

```
typedef struct ctimer_t ctimer_t
```

Definition at line 1 of file ctimer.h.

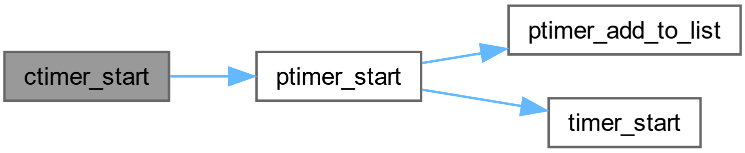
## 4.18.4 Function Documentation

### 4.18.4.1 ctimer_start()

```
void ctimer_start (
            ctimer_t * ctimer,
            timespan_t span,
            ctimer_callback_t callback,
            void * data )
```

Definition at line 57 of file ctimer.c.

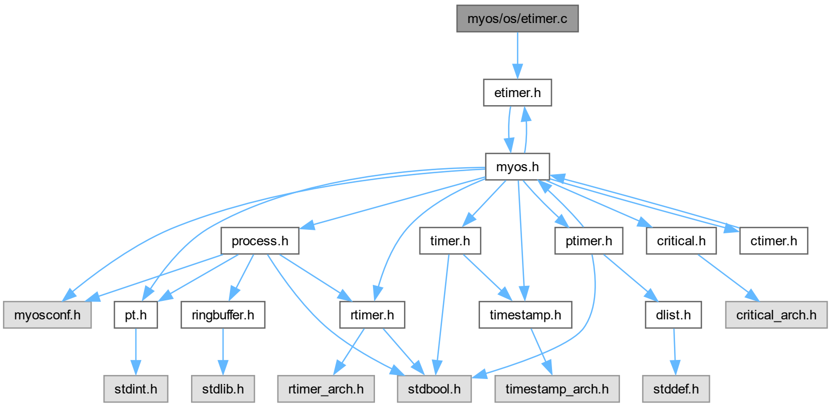
Here is the call graph for this function:



## 4.19 myos/os/etimer.c File Reference

```
#include "etimer.h"
```

Include dependency graph for etimer.c:



### Functions

- bool process_deliver_event (process_event_t ∗evt)
- void etimer_timeout_handler (ptimer_t ∗ptimer)
- void etimer_start (etimer_t ∗etimer, timespan_t span, process_t ∗to, process_event_id_t evtid, void ∗data)

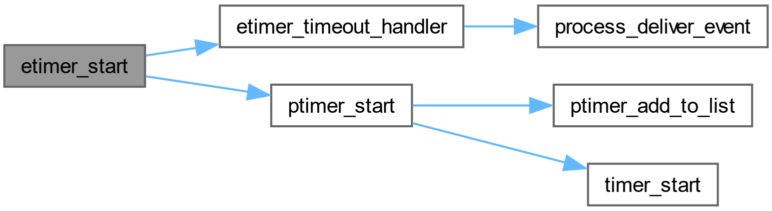
### 4.19.1 Function Documentation

#### 4.19.1.1 etimer_start()

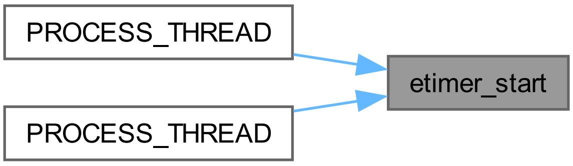
```
void etimer_start (
            etimer_t ∗ etimer,
            timespan_t span,
            process_t ∗ to,
            process_event_id_t evtid,
            void ∗ data )
```

Definition at line 53 of file etimer.c.

Here is the call graph for this function:



Here is the caller graph for this function:

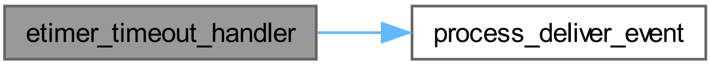


#### 4.19.1.2 etimer_timeout_handler()

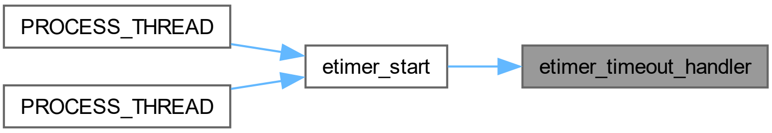
```
void etimer_timeout_handler (
            ptimer_t * ptimer )
```

Definition at line 42 of file etimer.c.

Here is the call graph for this function:



Here is the caller graph for this function:

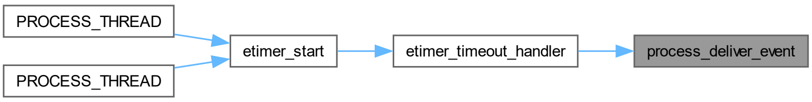


### 4.19.1.3 process_deliver_event()

```
bool process_deliver_event (
            process_event_t * evt )
```

Definition at line 102 of file process.c.

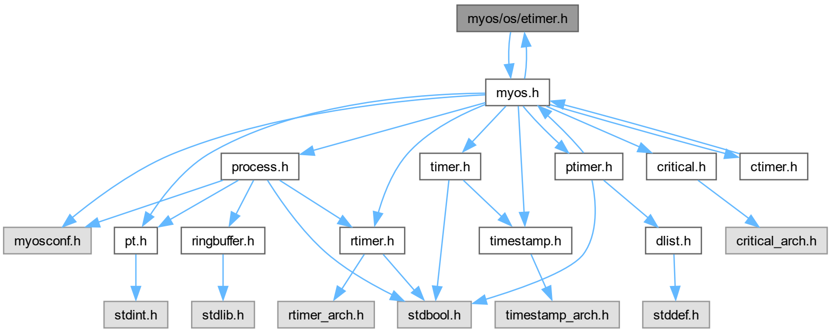
Here is the caller graph for this function:



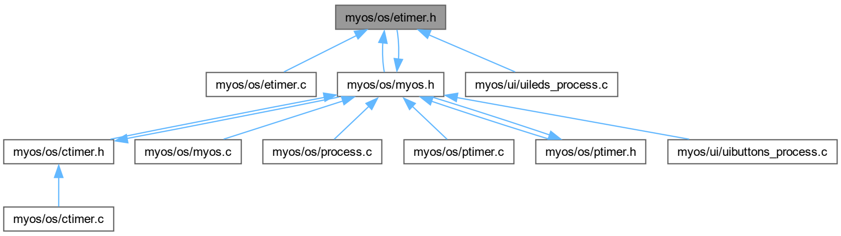
## 4.20 myos/os/etimer.h File Reference

```
#include "myos.h"
```

Include dependency graph for etimer.h:



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



## Data Structures

- struct etimer_t

## Macros

- #define etimer_module_init() ptimer_module_init()
- #define etimer_restart(etimerptr) ptimer_restart((ptimer_t∗)etimerptr)
- #define etimer_reset(etimerptr) ptimer_reset((ptimer_t∗)etimerptr)
- #define etimer_stop(etimerptr) ptimer_stop((ptimer_t∗)etimerptr)
- #define etimer_expired(etimerptr) ptimer_expired((ptimer_t∗)etimerptr)
- #define PROCESS_SLEEP(etimerptr, span)

## Functions

- void etimer_start (etimer_t ∗etimer, timespan_t span, process_t ∗to, process_event_id_t evtid, void ∗data)

## 4.20.1 Macro Definition Documentation

### 4.20.1.1 etimer_expired

```
#define etimer_expired(
             etimerptr ) ptimer_expired((ptimer_t*)etimerptr)
```

Definition at line 52 of file etimer.h.

### 4.20.1.2 etimer_module_init

```
#define etimer_module_init( ) ptimer_module_init()
```

Definition at line 47 of file etimer.h.

### 4.20.1.3 etimer_reset

```
#define etimer_reset(
             etimerptr ) ptimer_reset((ptimer_t*)etimerptr)
```

Definition at line 50 of file etimer.h.

#### 4.20.1.4 etimer_restart

```
#define etimer_restart(
            etimerptr ) ptimer_restart((ptimer_t*)etimerptr)
```

Definition at line 49 of file etimer.h.

#### 4.20.1.5 etimer_stop

```
#define etimer_stop(
            etimerptr ) ptimer_stop((ptimer_t*)etimerptr)
```

Definition at line 51 of file etimer.h.

#### 4.20.1.6 PROCESS_SLEEP

```
#define PROCESS_SLEEP(
            etimerptr,
            span )
```

**Value:**

```
do{ \
    etimer_start(etimerptr, span, PROCESS_THIS(), PROCESS_EVENT_CONTINUE, NULL); \
    PROCESS_WAIT_EVENT(PROCESS_EVENT_CONTINUE); \
}while(0)
```

Definition at line 56 of file etimer.h.

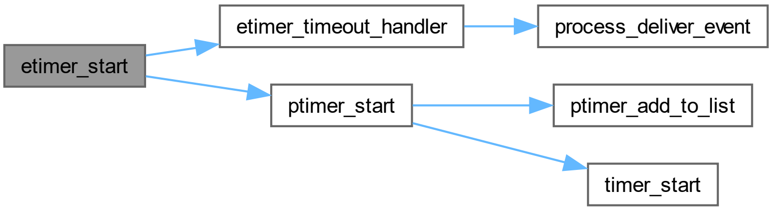
### 4.20.2 Function Documentation

#### 4.20.2.1 etimer_start()

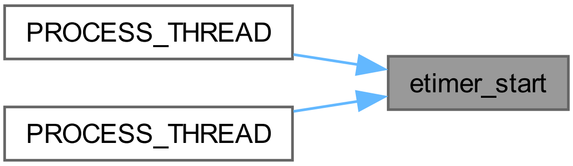
```
void etimer_start (
            etimer_t * etimer,
            timespan_t span,
            process_t * to,
            process_event_id_t evtid,
            void * data )
```

Definition at line 53 of file etimer.c.

Here is the call graph for this function:



Here is the caller graph for this function:



## 4.21 myos/os/myos.c File Reference

```
#include "myos.h"
```

Include dependency graph for myos.c:

### Functions

- void myos_init (void)

### 4.21.1 Function Documentation

#### 4.21.1.1 myos_init()

```
void myos_init (
            void  )
```

**Copyright**

Definition at line 74 of file myos.c.

Here is the call graph for this function:

## 4.22 myos/os/myos.h File Reference

```
#include "myosconf.h"
#include "critical.h"
#include "pt.h"
#include "process.h"
#include "timestamp.h"
#include "timer.h"
#include "ptimer.h"
#include "ctimer.h"
#include "etimer.h"
#include "rtimer.h"
```

Include dependency graph for myos.h: This graph shows which files directly or indirectly include this file:

### Functions

- void myos_init (void)

### 4.22.1 Function Documentation

#### 4.22.1.1 myos_init()

```
void myos_init (
             void  )
```

**Copyright**

Definition at line 74 of file myos.c.

Here is the call graph for this function:

## 4.23 myos/os/process.c File Reference

```
#include "myos.h"
#include <stdlib.h>
#include "debug.h"
```

Include dependency graph for process.c:

### Macros

- #define DBG(...)
- #define DBG_PROCESS(...)

### Functions

- RINGBUFFER_TYPEDEF (process_event_queue, process_event_t, 8)
- bool process_post (process_t ∗to, process_event_id_t evtid, void ∗data)
- bool process_deliver_event (process_event_t ∗evt)
- bool process_post_sync (process_t ∗to, process_event_id_t evtid, void ∗data)
- bool process_start (process_t ∗process, void ∗data)
- bool process_exit (process_t ∗process)
- void process_poll (process_t ∗process)
- int process_run (void)

## Variables

- process_t ∗ process_current = NULL

## 4.23.1 Macro Definition Documentation

### 4.23.1.1 DBG

```
#define DBG(
              ... )
```

Definition at line 46 of file process.c.

### 4.23.1.2 DBG_PROCESS

```
#define DBG_PROCESS(
              ... )
```

Definition at line 47 of file process.c.

## 4.23.2 Function Documentation

### 4.23.2.1 process_deliver_event()

```
bool process_deliver_event (
             process_event_t ∗ evt )
```

Definition at line 102 of file process.c.

Here is the caller graph for this function:

### 4.23.2.2 process_exit()

```
bool process_exit (
             process_t ∗ process )
```

Definition at line 174 of file process.c.

Here is the call graph for this function:

#### 4.23.2.3 process_poll()

```
void process_poll (
            process_t * process )
```

Definition at line 193 of file process.c.

#### 4.23.2.4 process_post()

```
bool process_post (
            process_t * to,
            process_event_id_t evtid,
            void * data )
```

Definition at line 70 of file process.c.

#### 4.23.2.5 process_post_sync()

```
bool process_post_sync (
            process_t * to,
            process_event_id_t evtid,
            void * data )
```

Definition at line 140 of file process.c.

Here is the call graph for this function: Here is the caller graph for this function:

#### 4.23.2.6 process_run()

```
int process_run (
            void  )  [inline]
```

Definition at line 201 of file process.c.

#### 4.23.2.7 process_start()

```
bool process_start (
            process_t * process,
            void * data )
```

Definition at line 153 of file process.c.

Here is the caller graph for this function:

#### 4.23.2.8 RINGBUFFER_TYPEDEF()

```
RINGBUFFER_TYPEDEF (
            process_event_queue ,
            process_event_t ,
            8 )
```

### 4.23.3 Variable Documentation

#### 4.23.3.1 process_current

```
process_t* process_current = NULL
```

Definition at line 50 of file process.c.

## 4.24 myos/os/process.h File Reference

```
#include "myosconf.h"
#include "pt.h"
#include "ringbuffer.h"
#include <stdbool.h>
#include "rtimer.h"
```

Include dependency graph for process.h: This graph shows which files directly or indirectly include this file:

### Data Structures

- struct process_t
- struct process_event_t

### Macros

- #define PROCESS_EVENT_QUEUE_SIZE 8
- #define PROCESS_EVENT_START 0
- #define PROCESS_EVENT_POLL 1
- #define PROCESS_EVENT_CONTINUE 2
- #define PROCESS_EVENT_TIMEOUT 3
- #define PROCESS_EVENT_EXIT 4
- #define PROCESS_BROADCAST NULL
- #define PROCESS_THIS() process_current
- #define PROCESS_CONTEXT_BEGIN(processptr)
- #define PROCESS_CONTEXT_END()
- #define PROCESS(name, threadname)
- #define PROCESS_EXTERN(name) extern process_t name
- #define PROCESS_DATA() (process->data)
- #define PROCESS_PT() (process->pt)
- #define PROCESS_EVENT_DATA() (evt->data)

- #define PROCESS_EVENT_ID() (evt->id)
- #define EXTERN_PROCESS(name) extern process_t name
- #define PROCESS_THREAD(name) int process_thread_##name(process_t ∗process, process_event_t ∗evt)
- #define PROCESS_RESPOND(evtid, dataptr) process_respond(evt,evtid,dataptr)
- #define PROCESS_INIT(processptr, threadname) process_init(processptr, process_thread_##threadname)
- #define PROCESS_IS_RUNNING(processptr) (PT_IS_RUNNING(&(processptr)->pt))
- #define PROCESS_BEGIN()
- #define PROCESS_END() PT_END(&PROCESS_PT())
- #define PROCESS_WAIT_EVENT(evtid) PT_YIELD_UNTIL(&PROCESS_PT(), PROCESS_EVENT_ID() == evtid)
- #define PROCESS_WAIT_EVENT_UNTIL(cond) PT_YIELD_UNTIL(&PROCESS_PT(), cond)
- #define PROCESS_WAIT_ANY_EVENT() PT_YIELD(&PROCESS_PT())
- #define PROCESS_SUSPEND()
- #define PROCESS_EXITHANDLER(handler)

## Typedefs

- typedef struct process_event_t process_event_t
- typedef struct process_t process_t
- typedef int(∗ process_thread_t) (process_t ∗process, process_event_t ∗evt)

## Functions

- void process_init (void)
- void process_init_process (process_t ∗process, process_thread_t thread)
- bool process_start (process_t ∗process, void ∗data)
- bool process_exit (process_t ∗process)
- bool process_post (process_t ∗to, process_event_id_t evtid, void ∗data)
- bool process_post_sync (process_t ∗to, process_event_id_t evtid, void ∗data)
- int process_run (void)
- void process_poll (process_t ∗process)

## Variables

- process_t ∗ process_current

## 4.24.1 Macro Definition Documentation

### 4.24.1.1 EXTERN_PROCESS

```
#define EXTERN_PROCESS(
              name ) extern process_t name
```

Definition at line 140 of file process.h.

#### 4.24.1.2 PROCESS

```
#define PROCESS(
              name,
              threadname )
```

**Value:**

```
   int process_thread_##threadname(process_t *process, process_event_t *evt);  \
   process_t name = {{0},process_thread_##threadname,0,{0},false}
```

Definition at line 127 of file process.h.

#### 4.24.1.3 PROCESS_BEGIN

```
#define PROCESS_BEGIN( )
```

**Value:**

```
   if(PROCESS_EVENT_ID() == PROCESS_EVENT_EXIT) {PT_EXIT(&PROCESS_PT());} \
   PT_BEGIN(&PROCESS_PT())
```

Definition at line 157 of file process.h.

#### 4.24.1.4 PROCESS_BROADCAST

```
#define PROCESS_BROADCAST NULL
```

Definition at line 82 of file process.h.

#### 4.24.1.5 PROCESS_CONTEXT_BEGIN

```
#define PROCESS_CONTEXT_BEGIN(
              processptr )
```

**Value:**

```
      do{ \
         process_t *process_backup; \
         process_backup = PROCESS_THIS(); \
         PROCESS_THIS() = processptr
```

Definition at line 116 of file process.h.

#### 4.24.1.6 PROCESS_CONTEXT_END

```
#define PROCESS_CONTEXT_END( )
```

**Value:**

```
        PROCESS_THIS() = process_backup; \
     }while(0)
```

Definition at line 122 of file process.h.

#### 4.24.1.7 PROCESS_DATA

```
#define PROCESS_DATA( ) (process->data)
```

Definition at line 134 of file process.h.

#### 4.24.1.8 PROCESS_END

```
#define PROCESS_END( ) PT_END(&PROCESS_PT())
```

Definition at line 163 of file process.h.

#### 4.24.1.9 PROCESS_EVENT_CONTINUE

```
#define PROCESS_EVENT_CONTINUE 2
```

Definition at line 78 of file process.h.

#### 4.24.1.10 PROCESS_EVENT_DATA

```
#define PROCESS_EVENT_DATA( ) (evt->data)
```

Definition at line 136 of file process.h.

#### 4.24.1.11 PROCESS_EVENT_EXIT

```
#define PROCESS_EVENT_EXIT 4
```

Definition at line 80 of file process.h.

### 4.24.1.12 PROCESS_EVENT_ID

```
#define PROCESS_EVENT_ID( ) (evt->id)
```

Definition at line 137 of file process.h.

### 4.24.1.13 PROCESS_EVENT_POLL

```
#define PROCESS_EVENT_POLL 1
```

Definition at line 77 of file process.h.

### 4.24.1.14 PROCESS_EVENT_QUEUE_SIZE

```
#define PROCESS_EVENT_QUEUE_SIZE 8
```

Definition at line 72 of file process.h.

### 4.24.1.15 PROCESS_EVENT_START

```
#define PROCESS_EVENT_START 0
```

Definition at line 76 of file process.h.

### 4.24.1.16 PROCESS_EVENT_TIMEOUT

```
#define PROCESS_EVENT_TIMEOUT 3
```

Definition at line 79 of file process.h.

### 4.24.1.17 PROCESS_EXITHANDLER

```
#define PROCESS_EXITHANDLER(
              handler )
```

Specify an action when a process exits.

**Note**

> This declaration must come immediately before the PROCESS_BEGIN() macro.

**Parameters**

| | |
|---|---|
| *handler* | The action to be performed. |

Definition at line 187 of file process.h.

### 4.24.1.18 PROCESS_EXTERN

```
#define PROCESS_EXTERN(
              name )   extern process_t name
```

Definition at line 131 of file process.h.

### 4.24.1.19 PROCESS_INIT

```
#define PROCESS_INIT(
              processptr,
              threadname )   process_init(processptr, process_thread_##threadname)
```

Definition at line 150 of file process.h.

### 4.24.1.20 PROCESS_IS_RUNNING

```
#define PROCESS_IS_RUNNING(
              processptr )   (PT_IS_RUNNING(&(processptr)->pt))
```

Definition at line 153 of file process.h.

### 4.24.1.21 PROCESS_PT

```
#define PROCESS_PT( ) (process->pt)
```

Definition at line 135 of file process.h.

#### 4.24.1.22 PROCESS_RESPOND

```
#define PROCESS_RESPOND(
              evtid,
              dataptr )   process_respond(evt,evtid,dataptr)
```

Definition at line 147 of file process.h.

#### 4.24.1.23 PROCESS_SUSPEND

```
#define PROCESS_SUSPEND( )
```

**Value:**

```
do{ \
   process_post(PROCESS_THIS(),PROCESS_EVENT_CONTINUE,NULL); \
   PROCESS_WAIT_EVENT(PROCESS_EVENT_CONTINUE); \
}while(0)
```

Definition at line 170 of file process.h.

#### 4.24.1.24 PROCESS_THIS

```
#define PROCESS_THIS( ) process_current
```

Definition at line 114 of file process.h.

#### 4.24.1.25 PROCESS_THREAD

```
#define PROCESS_THREAD(
              name ) int process_thread_##name(process_t *process, process_event_t *evt)
```

Definition at line 143 of file process.h.

#### 4.24.1.26 PROCESS_WAIT_ANY_EVENT

```
#define PROCESS_WAIT_ANY_EVENT( ) PT_YIELD(&PROCESS_PT())
```

Definition at line 166 of file process.h.

#### 4.24.1.27 PROCESS_WAIT_EVENT

```
#define PROCESS_WAIT_EVENT(
              evtid ) PT_YIELD_UNTIL(&PROCESS_PT(), PROCESS_EVENT_ID() == evtid)
```

Definition at line 164 of file process.h.

#### 4.24.1.28 PROCESS_WAIT_EVENT_UNTIL

```
#define PROCESS_WAIT_EVENT_UNTIL(
              cond ) PT_YIELD_UNTIL(&PROCESS_PT(), cond)
```

Definition at line 165 of file process.h.

### 4.24.2 Typedef Documentation

#### 4.24.2.1 process_event_t

```
typedef struct process_event_t process_event_t
```

Definition at line 1 of file process.h.

#### 4.24.2.2 process_t

```
typedef struct process_t process_t
```

Definition at line 1 of file process.h.

#### 4.24.2.3 process_thread_t

```
typedef int(* process_thread_t) (process_t *process, process_event_t *evt)
```

Definition at line 88 of file process.h.

### 4.24.3 Function Documentation

#### 4.24.3.1 process_exit()

```
bool process_exit (
            process_t * process )
```

Definition at line 174 of file process.c.

Here is the call graph for this function:

#### 4.24.3.2 process_init()

```
void process_init (
            void  )
```

Here is the caller graph for this function:

#### 4.24.3.3 process_init_process()

```
void process_init_process (
            process_t * process,
            process_thread_t thread )
```

#### 4.24.3.4 process_poll()

```
void process_poll (
            process_t * process )
```

Definition at line 193 of file process.c.

#### 4.24.3.5 process_post()

```
bool process_post (
            process_t * to,
            process_event_id_t evtid,
            void * data )
```

Definition at line 70 of file process.c.

#### 4.24.3.6 process_post_sync()

```
bool process_post_sync (
            process_t * to,
            process_event_id_t evtid,
            void * data )
```

Definition at line 140 of file process.c.

Here is the call graph for this function: Here is the caller graph for this function:

#### 4.24.3.7 process_run()

```
int process_run (
            void  )  [inline]
```

Definition at line 201 of file process.c.

#### 4.24.3.8 process_start()

```
bool process_start (
            process_t * process,
            void * data )
```

Definition at line 153 of file process.c.

Here is the caller graph for this function:

### 4.24.4 Variable Documentation

#### 4.24.4.1 process_current

```
process_t* process_current  [extern]
```

Definition at line 50 of file process.c.

## 4.25 myos/os/pt.h File Reference

Protothread definitions for myos.

```
#include <stdint.h>
```
Include dependency graph for pt.h: This graph shows which files directly or indirectly include this file:

### Data Structures

- struct pt_t

## Macros

- #define PT_STATE_WAITING 1
- #define PT_STATE_TERMINATED ((ptstate_t)(∼((ptstate_t)(0))))
- #define PT_IS_RUNNING(pt) ( ((pt)->lc > 0) && ((pt)->lc != LC_DEFAULT) )
- #define PT_INIT(pt)
- #define PT_THREAD(name_args)
- #define PT_BEGIN(pt)
- #define PT_END(pt)
- #define PT_WAIT_UNTIL(pt, condition)
- #define PT_WAIT_WHILE(pt, cond)
- #define PT_WAIT_THREAD(pt, thread)
- #define PT_SPAWN(pt, child, thread)
- #define PT_RESTART(pt)
- #define PT_EXIT(pt)
- #define PT_SCHEDULE(f)
- #define PT_YIELD(pt)
- #define PT_YIELD_UNTIL(pt, cond)

  *Yield from the protothread until a condition occurs.*

## Typedefs

- typedef uint8_t ptstate_t

- #define LC_INIT(s) s = 0;
- #define LC_DEFAULT ((lc_t)(∼((lc_t)(0))))
- #define LC_SET_DEFAULT(s) s = LC_DEFAULT
- #define LC_RESUME(s) switch(s) { case 0:
- #define LC_SET(s) s = __LINE__; case __LINE__:
- #define LC_SET_YIELD(s, retval) s = __LINE__; return retval; case __LINE__:
- #define LC_END(s) default:;}
- typedef uint16_t lc_t

## 4.25.1 Detailed Description

Protothread definitions for myos.

Based on Contiki 2.7 protothread implementation.

## 4.25.2 Macro Definition Documentation

### 4.25.2.1 LC_DEFAULT

```
#define LC_DEFAULT ((lc_t)(∼((lc_t)(0))))
```

Definition at line 56 of file pt.h.

#### 4.25.2.2 LC_END

```
#define LC_END(
              s ) default:;}
```

Definition at line 61 of file pt.h.

#### 4.25.2.3 LC_INIT

```
#define LC_INIT(
              s ) s = 0;
```

Definition at line 55 of file pt.h.

#### 4.25.2.4 LC_RESUME

```
#define LC_RESUME(
              s ) switch(s) { case 0:
```

Definition at line 58 of file pt.h.

#### 4.25.2.5 LC_SET

```
#define LC_SET(
              s ) s = __LINE__; case __LINE__:
```

Definition at line 59 of file pt.h.

#### 4.25.2.6 LC_SET_DEFAULT

```
#define LC_SET_DEFAULT(
              s ) s = LC_DEFAULT
```

Definition at line 57 of file pt.h.

#### 4.25.2.7 LC_SET_YIELD

```
#define LC_SET_YIELD(
              s,
              retval ) s = __LINE__; return retval; case __LINE__:
```

Definition at line 60 of file pt.h.

#### 4.25.2.8 PT_BEGIN

```
#define PT_BEGIN(
              pt )
```

Declare the start of a protothread inside the C function implementing the protothread.

This macro is used to declare the starting point of a protothread. It should be placed at the start of the function in which the protothread runs. All C statements above the PT_BEGIN() invokation will be executed each time the protothread is scheduled.

**Parameters**

| | |
|---|---|
| *pt* | A pointer to the protothread control structure. |

Definition at line 120 of file pt.h.

#### 4.25.2.9 PT_END

```
#define PT_END(
              pt )
```

Declare the end of a protothread.

This macro is used for declaring that a protothread ends. It must always be used together with a matching PT_BEGIN() macro.

**Parameters**

| | |
|---|---|
| *pt* | A pointer to the protothread control structure. |

Definition at line 132 of file pt.h.

#### 4.25.2.10 PT_EXIT

```
#define PT_EXIT(
              pt )
```

Exit the protothread.

This macro causes the protothread to exit. If the protothread was spawned by another protothread, the parent protothread will become unblocked and can continue to run.

**Parameters**

| | |
|---|---|
| *pt* | A pointer to the protothread control structure. |

Definition at line 234 of file pt.h.

### 4.25.2.11 PT_INIT

```
#define PT_INIT(
            pt )
```

Initialize a protothread.

Initializes a protothread. Initialization must be done prior to starting to execute the protothread.

**Parameters**

| | |
|---|---|
| *pt* | A pointer to the protothread control structure. |

**See also**

PT_SPAWN()

Definition at line 92 of file pt.h.

### 4.25.2.12 PT_IS_RUNNING

```
#define PT_IS_RUNNING(
            pt ) ( ((pt)->lc > 0) && ((pt)->lc != LC_DEFAULT) )
```

Definition at line 76 of file pt.h.

### 4.25.2.13 PT_RESTART

```
#define PT_RESTART(
            pt )
```

Restart the protothread.

This macro will block and cause the running protothread to restart its execution at the place of the PT_BEGIN() call.

**Parameters**

| | |
|---|---|
| *pt* | A pointer to the protothread control structure. |

Definition at line 217 of file pt.h.

### 4.25.2.14 PT_SCHEDULE

```
#define PT_SCHEDULE(
              f )
```

Schedule a protothread.

This function schedules a protothread. The return value of the function is non-zero if the protothread is running or zero if the protothread has exited.

**Parameters**

| | |
|---|---|
| *f* | The call to the C function implementing the protothread to be scheduled |

Definition at line 252 of file pt.h.

### 4.25.2.15 PT_SPAWN

```
#define PT_SPAWN(
              pt,
              child,
              thread )
```

Spawn a child protothread and wait until it exits.

This macro spawns a child protothread and waits until it exits. The macro can only be used within a protothread.

**Parameters**

| | |
|---|---|
| *pt* | A pointer to the protothread control structure. |
| *child* | A pointer to the child protothread's control structure. |
| *thread* | The child protothread with arguments |

Definition at line 200 of file pt.h.

### 4.25.2.16 PT_STATE_TERMINATED

```
#define PT_STATE_TERMINATED ((ptstate_t)(∼((ptstate_t)(0))))
```

Definition at line 73 of file pt.h.

#### 4.25.2.17 PT_STATE_WAITING

```
#define PT_STATE_WAITING 1
```

Definition at line 72 of file pt.h.

#### 4.25.2.18 PT_THREAD

```
#define PT_THREAD(
              name_args )
```

Declaration of a protothread.

This macro is used to declare a protothread. All protothreads must be declared with this macro.

**Parameters**

| | |
|---|---|
| *name_args* | The name and arguments of the C function implementing the protothread. |

Definition at line 105 of file pt.h.

#### 4.25.2.19 PT_WAIT_THREAD

```
#define PT_WAIT_THREAD(
              pt,
              thread )
```

Block and wait until a child protothread completes.

This macro schedules a child protothread. The current protothread will block until the child protothread completes.

**Note**

> The child protothread must be manually initialized with the PT_INIT() function before this function is used.

**Parameters**

| | |
|---|---|
| *pt* | A pointer to the protothread control structure. |
| *thread* | The child protothread with arguments |

**See also**

PT_SPAWN()

Definition at line 186 of file pt.h.

#### 4.25.2.20 PT_WAIT_UNTIL

```
#define PT_WAIT_UNTIL(
              pt,
              condition )
```

Block and wait until condition is true.

This macro blocks the protothread until the specified condition is true.

**Parameters**

| | |
|---|---|
| *pt* | A pointer to the protothread control structure. |
| *condition* | The condition. |

Definition at line 148 of file pt.h.

#### 4.25.2.21 PT_WAIT_WHILE

```
#define PT_WAIT_WHILE(
              pt,
              cond )
```

Block and wait while condition is true.

This function blocks and waits while condition is true. See PT_WAIT_UNTIL().

**Parameters**

| | |
|---|---|
| *pt* | A pointer to the protothread control structure. |
| *cond* | The condition. |

Definition at line 167 of file pt.h.

#### 4.25.2.22 PT_YIELD

```
#define PT_YIELD(
              pt )
```

Yield from the current protothread.

This function will yield the protothread, thereby allowing other processing to take place in the system.

**Parameters**

| | |
|---|---|
| *pt* | A pointer to the protothread control structure. |

Definition at line 264 of file pt.h.

#### 4.25.2.23 PT_YIELD_UNTIL

```
#define PT_YIELD_UNTIL(
              pt,
              cond )
```

Yield from the protothread until a condition occurs.

**Parameters**

| | |
|---|---|
| *pt* | A pointer to the protothread control structure. |
| *cond* | The condition.<br>`This function will yield the protothread, until the specified condition evaluates to true.` |

Definition at line 281 of file pt.h.

### 4.25.3 Typedef Documentation

#### 4.25.3.1 lc_t

```
typedef uint16_t lc_t
```

Definitions for local continuations

Definition at line 54 of file pt.h.

#### 4.25.3.2 ptstate_t

```
typedef uint8_t ptstate_t
```

Definition at line 69 of file pt.h.

## 4.26 myos/os/ptimer.c File Reference

```
#include "myos.h"
```

Include dependency graph for ptimer.c:

### Functions

- void ptimer_add_to_list (ptimer_t ∗ptimer)
- void ptimer_remove_from_list (ptimer_t ∗ptimer)
- int process_thread_ptimer_process (process_t ∗process, process_event_t ∗evt)
- void ptimer_start (ptimer_t ∗ptimer, timespan_t span, ptimer_handler_t handler)
- void ptimer_restart (ptimer_t ∗ptimer)
- void ptimer_reset (ptimer_t ∗ptimer)
- bool process_deliver_event (process_event_t ∗evt)
- void ptimer_processing (void)

### Variables

- timestamp_t ptimer_next_stop = 0
- bool ptimer_pending = false
- const process_event_t ptimer_poll_evt
- process_t ptimer_process = {{0},process_thread_ptimer_process ,0,{0},false}

### 4.26.1 Function Documentation

#### 4.26.1.1 process_deliver_event()

```
bool process_deliver_event (
            process_event_t ∗ evt )
```

Definition at line 102 of file process.c.

Here is the caller graph for this function:

#### 4.26.1.2 process_thread_ptimer_process()

```
int process_thread_ptimer_process (
            process_t ∗ process,
            process_event_t ∗ evt )
```

Definition at line 104 of file ptimer.c.

#### 4.26.1.3 ptimer_add_to_list()

```
void ptimer_add_to_list (
            ptimer_t * ptimer )
```

Definition at line 69 of file ptimer.c.

Here is the caller graph for this function:

#### 4.26.1.4 ptimer_processing()

```
void ptimer_processing (
            void  )
```

Definition at line 223 of file ptimer.c.

Here is the call graph for this function:

#### 4.26.1.5 ptimer_remove_from_list()

```
void ptimer_remove_from_list (
            ptimer_t * ptimer )
```

Definition at line 87 of file ptimer.c.

#### 4.26.1.6 ptimer_reset()

```
void ptimer_reset (
            ptimer_t * ptimer )
```

Definition at line 212 of file ptimer.c.

Here is the call graph for this function:

#### 4.26.1.7 ptimer_restart()

```
void ptimer_restart (
            ptimer_t * ptimer )
```

Definition at line 205 of file ptimer.c.

Here is the call graph for this function:

#### 4.26.1.8 ptimer_start()

```
void ptimer_start (
            ptimer_t * ptimer,
            timespan_t span,
            ptimer_handler_t handler )
```

Definition at line 198 of file ptimer.c.

Here is the call graph for this function: Here is the caller graph for this function:

### 4.26.2 Variable Documentation

#### 4.26.2.1 ptimer_next_stop

```
timestamp_t ptimer_next_stop = 0
```

Definition at line 36 of file ptimer.c.

#### 4.26.2.2 ptimer_pending

```
bool ptimer_pending = false
```

Definition at line 37 of file ptimer.c.

#### 4.26.2.3 ptimer_poll_evt

```
const process_event_t ptimer_poll_evt
```

**Initial value:**

```
= {
      .to = &ptimer_process,
      .id =  1 ,
      .data = NULL
}
```

Definition at line 44 of file ptimer.c.

#### 4.26.2.4 ptimer_process

```
process_t ptimer_process = {{0},process_thread_ptimer_process ,0,{0},false}
```

Definition at line 103 of file ptimer.c.

## 4.27 myos/os/ptimer.h File Reference

```
#include "myos.h"
#include <stdbool.h>
#include "dlist.h"
```

Include dependency graph for ptimer.h: This graph shows which files directly or indirectly include this file:

### Data Structures

- struct ptimer_t

### Macros

- #define PTLIST_NODE_TYPE DLIST_NODE_TYPE
- #define ptlist_init(listptr) dlist_init(listptr)
- #define ptlist_erase(listptr, nodeptr) dlist_erase(listptr,nodeptr)
- #define ptlist_next(listptr, nodeptr) dlist_next(listptr,nodeptr)
- #define ptlist_push_front(listptr, nodeptr) dlist_push_front(listptr,nodeptr)
- #define ptlist_prev(listptr, nodeptr) dlist_prev(listptr,nodeptr)
- #define ptlist_foreach(listptr, iterator) dlist_foreach(listptr,iterator)
- #define ptlist_find(listptr, nodeptr) dlist_findlistptr,nodeptr)
- #define ptlist_begin(listptr) dlist_begin(listptr)
- #define ptlist_end(listptr) dlist_end(listptr)
- #define ptlist_empty(listptr) dlist_empty(listptr)
- #define ptimer_module_init() process_start(&ptimer_process, NULL);
- #define ptimer_stop(ptimerptr) ptimer_remove_from_list(ptimerptr);
- #define ptimer_expired(ptimerptr) timer_expired(&(ptimerptr)->timer)

### Typedefs

- typedef dlist_t ptlist_t
- typedef dlist_node_t ptlist_node_t
- typedef struct ptimer_t ptimer_t
- typedef void(∗ ptimer_handler_t) (ptimer_t ∗data)

### Functions

- void ptimer_processing (void)
- void ptimer_start (ptimer_t ∗ptimer, timespan_t span, ptimer_handler_t handler)
- void ptimer_restart (ptimer_t ∗ptimer)
- void ptimer_reset (ptimer_t ∗ptimer)

### Variables

- process_t ptimer_process

## 4.27.1 Macro Definition Documentation

### 4.27.1.1 ptimer_expired

```
#define ptimer_expired(
              ptimerptr ) timer_expired(&(ptimerptr)->timer)
```

Definition at line 96 of file ptimer.h.

### 4.27.1.2 ptimer_module_init

```
#define ptimer_module_init( ) process_start(&ptimer_process, NULL);
```

Definition at line 90 of file ptimer.h.

### 4.27.1.3 ptimer_stop

```
#define ptimer_stop(
              ptimerptr ) ptimer_remove_from_list(ptimerptr);
```

Definition at line 95 of file ptimer.h.

### 4.27.1.4 ptlist_begin

```
#define ptlist_begin(
              listptr ) dlist_begin(listptr)
```

Definition at line 51 of file ptimer.h.

### 4.27.1.5 ptlist_empty

```
#define ptlist_empty(
              listptr ) dlist_empty(listptr)
```

Definition at line 53 of file ptimer.h.

### 4.27.1.6 ptlist_end

```
#define ptlist_end(
              listptr ) dlist_end(listptr)
```

Definition at line 52 of file ptimer.h.

### 4.27.1.7 ptlist_erase

```
#define ptlist_erase(
              listptr,
              nodeptr ) dlist_erase(listptr,nodeptr)
```

Definition at line 45 of file ptimer.h.

### 4.27.1.8 ptlist_find

```
#define ptlist_find(
              listptr,
              nodeptr ) dlist_findlistptr,nodeptr)
```

Definition at line 50 of file ptimer.h.

### 4.27.1.9 ptlist_foreach

```
#define ptlist_foreach(
              listptr,
              iterator ) dlist_foreach(listptr,iterator)
```

Definition at line 49 of file ptimer.h.

### 4.27.1.10 ptlist_init

```
#define ptlist_init(
              listptr ) dlist_init(listptr)
```

Definition at line 44 of file ptimer.h.

#### 4.27.1.11 ptlist_next

```
#define ptlist_next(
              listptr,
              nodeptr ) dlist_next(listptr,nodeptr)
```

Definition at line 46 of file ptimer.h.

#### 4.27.1.12 PTLIST_NODE_TYPE

```
#define PTLIST_NODE_TYPE DLIST_NODE_TYPE
```

Definition at line 43 of file ptimer.h.

#### 4.27.1.13 ptlist_prev

```
#define ptlist_prev(
              listptr,
              nodeptr ) dlist_prev(listptr,nodeptr)
```

Definition at line 48 of file ptimer.h.

#### 4.27.1.14 ptlist_push_front

```
#define ptlist_push_front(
              listptr,
              nodeptr ) dlist_push_front(listptr,nodeptr)
```

Definition at line 47 of file ptimer.h.

### 4.27.2 Typedef Documentation

#### 4.27.2.1 ptimer_handler_t

```
typedef void(* ptimer_handler_t) (ptimer_t *data)
```

Definition at line 80 of file ptimer.h.

#### 4.27.2.2 ptimer_t

```
typedef struct ptimer_t ptimer_t
```

Definition at line 76 of file ptimer.h.

#### 4.27.2.3 ptlist_node_t

```
typedef dlist_node_t ptlist_node_t
```

Definition at line 42 of file ptimer.h.

#### 4.27.2.4 ptlist_t

```
typedef dlist_t ptlist_t
```

**Copyright**

Definition at line 41 of file ptimer.h.

### 4.27.3 Function Documentation

#### 4.27.3.1 ptimer_processing()

```
void ptimer_processing (
            void  )
```

Definition at line 223 of file ptimer.c.

Here is the call graph for this function:

#### 4.27.3.2 ptimer_reset()

```
void ptimer_reset (
            ptimer_t * ptimer )
```

Definition at line 212 of file ptimer.c.

Here is the call graph for this function:

#### 4.27.3.3 ptimer_restart()

```
void ptimer_restart (
            ptimer_t * ptimer )
```

Definition at line 205 of file ptimer.c.

Here is the call graph for this function:

#### 4.27.3.4 ptimer_start()

```
void ptimer_start (
            ptimer_t * ptimer,
            timespan_t span,
            ptimer_handler_t handler )
```

Definition at line 198 of file ptimer.c.

Here is the call graph for this function: Here is the caller graph for this function:

### 4.27.4 Variable Documentation

#### 4.27.4.1 ptimer_process

```
process_t ptimer_process  [extern]
```

Definition at line 103 of file ptimer.c.

## 4.28 myos/os/rtimer.c File Reference

```
#include "rtimer.h"
#include <stdlib.h>
#include "critical.h"
```

Include dependency graph for rtimer.c:

### Functions

- bool rtimer_lock (void)
- void rtimer_release ()
- void rtimer_scheduler (void)
- rtimer_timespan_t rtimer_left (rtimer_t ∗rtimer)
- void rtimer_start (rtimer_t ∗rtimer, rtimer_timespan_t span, rtimer_callback_t callback, void ∗data)
- void rtimer_restart (rtimer_t ∗rtimer)
- void rtimer_reset (rtimer_t ∗rtimer)

### Variables

- rtimer_t ∗ rtimer_next = NULL
- bool rtimer_mutex = false

### 4.28.1 Function Documentation

#### 4.28.1.1 rtimer_left()

```
rtimer_timespan_t rtimer_left (
            rtimer_t ∗ rtimer )
```

Definition at line 77 of file rtimer.c.

#### 4.28.1.2 rtimer_lock()

```
bool rtimer_lock (
            void   )
```

Definition at line 42 of file rtimer.c.

#### 4.28.1.3 rtimer_release()

```
void rtimer_release ( )
```

Definition at line 59 of file rtimer.c.

Here is the caller graph for this function:

#### 4.28.1.4 rtimer_reset()

```
void rtimer_reset (
             rtimer_t * rtimer )
```

Definition at line 107 of file rtimer.c.

#### 4.28.1.5 rtimer_restart()

```
void rtimer_restart (
             rtimer_t * rtimer )
```

Definition at line 100 of file rtimer.c.

#### 4.28.1.6 rtimer_scheduler()

```
void rtimer_scheduler (
             void  )
```

Definition at line 66 of file rtimer.c.

Here is the call graph for this function:

#### 4.28.1.7 rtimer_start()

```
void rtimer_start (
             rtimer_t * rtimer,
             rtimer_timespan_t span,
             rtimer_callback_t callback,
             void * data )
```

Definition at line 90 of file rtimer.c.

### 4.28.2 Variable Documentation

#### 4.28.2.1 rtimer_mutex

```
bool rtimer_mutex = false
```

Definition at line 40 of file rtimer.c.

#### 4.28.2.2 rtimer_next

```
rtimer_t* rtimer_next = NULL
```

Definition at line 39 of file rtimer.c.

## 4.29 myos/os/rtimer.h File Reference

```
#include "rtimer_arch.h"
#include "stdbool.h"
```

Include dependency graph for rtimer.h: This graph shows which files directly or indirectly include this file:

### Data Structures

- struct rtimer_t

### Macros

- #define RTIMER_TICKS_PER_SEC RTIMER_ARCH_TICKS_PER_SEC
- #define rtimer_now() rtimer_arch_now()
- #define rtimer_init() rtimer_arch_init()
- #define RTIMER_TIMESTAMP_DIFF RTIMER_TIMESTAMP_ARCH_DIFF
- #define rtimer_timestamp_less_than(a, b) (RTIMER_TIMESTAMP_DIFF((a),(b)) < 0)
- #define PROCESS_RTIMER_OBTAIN()
- #define rtimer_expired(rtimerptr) (rtimer_left(rtimerptr) == 0)
- #define rtimer_timestamp_stop(rtimerptr) ((rtimerptr)->start+(rtimerptr)->span)
- #define rtimer_module_init rtimer_arch_module_init

### Typedefs

- typedef rtimer_arch_timestamp_t rtimer_timestamp_t
- typedef rtimer_timestamp_t rtimer_timespan_t
- typedef void(* rtimer_callback_t) (void *data)

### Functions

- void rtimer_start (rtimer_t *rtimer, rtimer_timespan_t span, rtimer_callback_t callback, void *data)
- void rtimer_reset (rtimer_t *rtimer)
- rtimer_timespan_t rtimer_left (rtimer_t *rtimer)
- bool rtimer_lock (void)

## 4.29.1 Macro Definition Documentation

### 4.29.1.1 PROCESS_RTIMER_OBTAIN

```
#define PROCESS_RTIMER_OBTAIN( )
```

**Value:**

```
do{ \
   while(!rtimer_lock()) \
   { \
      PROCESS_SUSPEND(); \
   } \
}while(0)
```

Definition at line 62 of file rtimer.h.

### 4.29.1.2 rtimer_expired

```
#define rtimer_expired(
                rtimerptr ) (rtimer_left(rtimerptr) == 0)
```

Definition at line 76 of file rtimer.h.

### 4.29.1.3 rtimer_init

```
#define rtimer_init( ) rtimer_arch_init()
```

Definition at line 48 of file rtimer.h.

### 4.29.1.4 rtimer_module_init

```
#define rtimer_module_init rtimer_arch_module_init
```

Definition at line 80 of file rtimer.h.

### 4.29.1.5 rtimer_now

```
#define rtimer_now( ) rtimer_arch_now()
```

Definition at line 47 of file rtimer.h.

#### 4.29.1.6 RTIMER_TICKS_PER_SEC

```
#define RTIMER_TICKS_PER_SEC RTIMER_ARCH_TICKS_PER_SEC
```

Definition at line 46 of file rtimer.h.

#### 4.29.1.7 RTIMER_TIMESTAMP_DIFF

```
#define RTIMER_TIMESTAMP_DIFF RTIMER_TIMESTAMP_ARCH_DIFF
```

Definition at line 49 of file rtimer.h.

#### 4.29.1.8 rtimer_timestamp_less_than

```
#define rtimer_timestamp_less_than(
              a,
              b ) (RTIMER_TIMESTAMP_DIFF((a),(b)) < 0)
```

Definition at line 50 of file rtimer.h.

#### 4.29.1.9 rtimer_timestamp_stop

```
#define rtimer_timestamp_stop(
              rtimerptr ) ((rtimerptr)->start+(rtimerptr)->span)
```

Definition at line 77 of file rtimer.h.

### 4.29.2 Typedef Documentation

#### 4.29.2.1 rtimer_callback_t

```
typedef void(* rtimer_callback_t) (void *data)
```

Definition at line 51 of file rtimer.h.

#### 4.29.2.2 rtimer_timespan_t

```
typedef rtimer_timestamp_t rtimer_timespan_t
```

Definition at line 44 of file rtimer.h.

#### 4.29.2.3 rtimer_timestamp_t

```
typedef rtimer_arch_timestamp_t rtimer_timestamp_t
```

Definition at line 43 of file rtimer.h.

### 4.29.3 Function Documentation

#### 4.29.3.1 rtimer_left()

```
rtimer_timespan_t rtimer_left (
             rtimer_t * rtimer )
```

Definition at line 77 of file rtimer.c.

#### 4.29.3.2 rtimer_lock()

```
bool rtimer_lock (
             void )
```

Definition at line 42 of file rtimer.c.

#### 4.29.3.3 rtimer_reset()

```
void rtimer_reset (
             rtimer_t * rtimer )
```

Definition at line 107 of file rtimer.c.

#### 4.29.3.4 rtimer_start()

```
void rtimer_start (
            rtimer_t * rtimer,
            rtimer_timespan_t span,
            rtimer_callback_t callback,
            void * data )
```

Definition at line 90 of file rtimer.c.

## 4.30 myos/os/timer.c File Reference

```
#include "timer.h"
```

Include dependency graph for timer.c:

### Functions

- void timer_start (myos_timer_t ∗timer, timespan_t span)

    *Starts a timer using a time span.*
- void timer_restart (myos_timer_t ∗timer)

    *Restarts a previously set timer.*
- void timer_reset (myos_timer_t ∗timer)

    *Resets a previously set timer.*

### 4.30.1 Function Documentation

#### 4.30.1.1 timer_reset()

```
void timer_reset (
            myos_timer_t * timer )
```

Resets a previously set timer.

Adds the currently set timespan to the timers start time.

**Parameters**

| | | |
|---|---|---|
| in | *timer* | Timer instance to reset |

Definition at line 72 of file timer.c.

Here is the caller graph for this function:

#### 4.30.1.2 timer_restart()

```
void timer_restart (
            myos_timer_t * timer )
```

Restarts a previously set timer.

Restarts a previously set timer. Be aware of that the timer must have been set properly before, otherwise behaviour is undefined.

**Parameters**

| | | |
|---|---|---|
| in | *timer* | Timer instance to restart |

Definition at line 65 of file timer.c.

Here is the caller graph for this function:

#### 4.30.1.3 timer_start()

```
void timer_start (
            myos_timer_t * timer,
            timespan_t span )
```

Starts a timer using a time span.

Timer will expire as soon as the time span elapsed. If the timer is running yet, the timer gets reinitialized with the new parameters provided. Any information about the previous timer setting gets lost.

**Parameters**

| | | |
|---|---|---|
| in | *timer* | Timer instance to set |
| in | *span* | Time span to wait to elapse |

Definition at line 59 of file timer.c.

Here is the caller graph for this function:

## 4.31 myos/os/timer.h File Reference

```
#include "timestamp.h"
#include <stdbool.h>
```

Include dependency graph for timer.h: This graph shows which files directly or indirectly include this file:

### Data Structures

- struct myos_timer_t

## Macros

- #define timer_t myos_timer_t
- #define timer_timestamp_stop(timerptr) ((timerptr)->start + (timerptr)->span)
- #define timer_expired(timerptr) timestamp_passed(timer_timestamp_stop(timerptr))

  *Check if timer expired.*
- #define timer_module_init timestamp_module_init

## Functions

- void timer_start (myos_timer_t ∗timer, timespan_t span)

  *Starts a timer using a time span.*
- void timer_reset (myos_timer_t ∗timer)

  *Resets a previously set timer.*
- void timer_restart (myos_timer_t ∗timer)

  *Restarts a previously set timer.*

## 4.31.1 Macro Definition Documentation

### 4.31.1.1 timer_expired

```
#define timer_expired(
              timerptr )  timestamp_passed(timer_timestamp_stop(timerptr))
```

Check if timer expired.

Checks if timer expired. Timer will expire as soon as the time span provided with timer_start is elapsed.

**Parameters**

| | | |
|---|---|---|
| in | *timer* | Timer instance to check |

**Returns**

true if timer expired, false otherwise

Definition at line 90 of file timer.h.

### 4.31.1.2 timer_module_init

```
#define timer_module_init timestamp_module_init
```

Definition at line 93 of file timer.h.

#### 4.31.1.3 timer_t

```
#define timer_t myos_timer_t
```

Definition at line 56 of file timer.h.

#### 4.31.1.4 timer_timestamp_stop

```
#define timer_timestamp_stop(
              timerptr )   ((timerptr)->start + (timerptr)->span)
```

Definition at line 79 of file timer.h.

### 4.31.2 Function Documentation

#### 4.31.2.1 timer_reset()

```
void timer_reset (
             myos_timer_t * timer )
```

Resets a previously set timer.

Adds the currently set timespan to the timers start time.

**Parameters**

| | | |
|---|---|---|
| in | *timer* | Timer instance to reset |

Definition at line 72 of file timer.c.

Here is the caller graph for this function:

#### 4.31.2.2 timer_restart()

```
void timer_restart (
             myos_timer_t * timer )
```

Restarts a previously set timer.

Restarts a previously set timer. Be aware of that the timer must have been set properly before, otherwise behaviour is undefined.

**Parameters**

| | | |
|---|---|---|
| in | *timer* | Timer instance to restart |

Definition at line 65 of file timer.c.

Here is the caller graph for this function:

#### 4.31.2.3 timer_start()

```
void timer_start (
            myos_timer_t * timer,
            timespan_t span )
```

Starts a timer using a time span.

Timer will expire as soon as the time span elapsed. If the timer is running yet, the timer gets reinitialized with the new parameters provided. Any information about the previous timer setting gets lost.

**Parameters**

| | | |
|---|---|---|
| in | *timer* | Timer instance to set |
| in | *span* | Time span to wait to elapse |

Definition at line 59 of file timer.c.

Here is the caller graph for this function:

## 4.32 myos/os/timestamp.h File Reference

Provides definitions to access and evaluate the systems time stamp counter.

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

### Macros

- #define TIMESTAMP_TICKS_PER_SEC TIMESTAMP_ARCH_TICKS_PER_SEC
- #define TIMESTAMP_DIFF TIMESTAMP_ARCH_DIFF
- #define timestamp_module_init timestamp_arch_module_init
- #define timestamp_now timestamp_arch_now
- #define timestamp_less_than(a, b) (TIMESTAMP_DIFF((a),(b)) < 0)
- #define timestamp_lessequal_than(a, b) (TIMESTAMP_DIFF((a),(b)) <= 0)
- #define timestamp_passed(timestamp) timestamp_lessequal_than(timestamp,timestamp_now())

  *Checks if a provided time stamp is reached.*
- #define timestamp_block_until(timestamp) while(!timestamp_passed(timestamp)){};

  *Blocks until the time stamp provided is in the past.*
- #define timestamp_block_for(timespan)

  *Blocks for some time.*

## Typedefs

- typedef timestamp_arch_t timestamp_t
- typedef timestamp_t timespan_t

## 4.32.1 Detailed Description

Provides definitions to access and evaluate the systems time stamp counter.

Provides the architecture independent part to access and evaluate the systems time stamp counter. For architecture dependent part see "timestamp_arch.h" of the corresponding architectural part.

Time stamps are the base for all non real time timers used by the os.

## 4.32.2 Macro Definition Documentation

### 4.32.2.1 timestamp_block_for

```
#define timestamp_block_for(
              timespan )
```

**Value:**

```
    do{                                                 \
        timestamp_t stop = timespan+timestamp_now();    \
        timestamp_block_until(stop);                    \
    }while(0)
```

Blocks for some time.

**Parameters**

| | | |
|---|---|---|
| in | *timespan* | Time to block |

Definition at line 89 of file timestamp.h.

### 4.32.2.2 timestamp_block_until

```
#define timestamp_block_until(
              timestamp )   while(!timestamp_passed(timestamp)){};
```

Blocks until the time stamp provided is in the past.

**Parameters**

| | | |
|---|---|---|
| in | *timestamp* | Time stamp to wait for |

Definition at line 82 of file timestamp.h.

### 4.32.2.3 TIMESTAMP_DIFF

```
#define TIMESTAMP_DIFF TIMESTAMP_ARCH_DIFF
```

Definition at line 54 of file timestamp.h.

### 4.32.2.4 timestamp_less_than

```
#define timestamp_less_than(
              a,
              b ) (TIMESTAMP_DIFF((a),(b)) < 0)
```

Definition at line 64 of file timestamp.h.

### 4.32.2.5 timestamp_lessequal_than

```
#define timestamp_lessequal_than(
              a,
              b ) (TIMESTAMP_DIFF((a),(b)) <= 0)
```

Definition at line 65 of file timestamp.h.

### 4.32.2.6 timestamp_module_init

```
#define timestamp_module_init timestamp_arch_module_init
```

Definition at line 59 of file timestamp.h.

### 4.32.2.7 timestamp_now

```
#define timestamp_now timestamp_arch_now
```

Definition at line 61 of file timestamp.h.

### 4.32.2.8 timestamp_passed

```
#define timestamp_passed(
              timestamp )  timestamp_lessequal_than(timestamp,timestamp_now())
```

Checks if a provided time stamp is reached.

Checks if the time stamp counter reached a provided time stamp yet.

**Parameters**

|  |  |  |
|---|---|---|
| `in` | *timestamp* | Time stamp to check |

**Returns**

1 if time stamp is in the past, otherwise 0

Definition at line 74 of file timestamp.h.

#### 4.32.2.9 TIMESTAMP_TICKS_PER_SEC

```
#define TIMESTAMP_TICKS_PER_SEC TIMESTAMP_ARCH_TICKS_PER_SEC
```

Definition at line 53 of file timestamp.h.

### 4.32.3 Typedef Documentation

#### 4.32.3.1 timespan_t

```
typedef timestamp_t timespan_t
```

Definition at line 57 of file timestamp.h.

#### 4.32.3.2 timestamp_t

```
typedef timestamp_arch_t timestamp_t
```

Definition at line 56 of file timestamp.h.

## 4.33 myos/ui/uibuttons.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include "uibuttons.h"
```

Include dependency graph for uibuttons.c:

## Macros

- #define debounce_timer hold_timer
- #define UIBUTTONS_TRANSITION_PRESSED() (button->prev_state == UIBUTTONS_STATE_RELEASED && curr_state == UIBUTTONS_STATE_PRESSED)
- #define UIBUTTONS_TRANSITION_RELEASED() (button->prev_state == UIBUTTONS_STATE_PRESSED && curr_state == UIBUTTONS_STATE_RELEASED)
- #define UIBUTTONS_TRANSITION_HELD() (button->prev_state == UIBUTTONS_STATE_PRESSED && curr_state == UIBUTTONS_STATE_PRESSED)
- #define UIBUTTONS_DEBOUNCING() (button->prev_state >= UIBUTTONS_STATE_RELEASED_DEBOUNCE)

## Functions

- void uibuttons_poll (uibutton_t ∗button)

## 4.33.1 Macro Definition Documentation

### 4.33.1.1 debounce_timer

```
#define debounce_timer hold_timer
```

**Copyright**

Definition at line 37 of file uibuttons.c.

#### 4.33.1.2 UIBUTTONS_DEBOUNCING

```
#define UIBUTTONS_DEBOUNCING( ) (button->prev_state >= UIBUTTONS_STATE_RELEASED_DEBOUNCE)
```

Definition at line 44 of file uibuttons.c.

#### 4.33.1.3 UIBUTTONS_TRANSITION_HELD

```
#define UIBUTTONS_TRANSITION_HELD( ) (button->prev_state == UIBUTTONS_STATE_PRESSED && curr_↩
state == UIBUTTONS_STATE_PRESSED)
```

Definition at line 41 of file uibuttons.c.

#### 4.33.1.4 UIBUTTONS_TRANSITION_PRESSED

```
#define UIBUTTONS_TRANSITION_PRESSED( ) (button->prev_state == UIBUTTONS_STATE_RELEASED &&
curr_state == UIBUTTONS_STATE_PRESSED)
```

Definition at line 39 of file uibuttons.c.

#### 4.33.1.5 UIBUTTONS_TRANSITION_RELEASED

```
#define UIBUTTONS_TRANSITION_RELEASED( ) (button->prev_state == UIBUTTONS_STATE_PRESSED &&
curr_state == UIBUTTONS_STATE_RELEASED)
```

Definition at line 40 of file uibuttons.c.

### 4.33.2 Function Documentation

#### 4.33.2.1 uibuttons_poll()

```
void uibuttons_poll (
             uibutton_t * button )
```

Definition at line 130 of file uibuttons.c.

## 4.34 myos/ui/uibuttons.h File Reference

```
#include <stdbool.h>
#include <stdint.h>
#include "uibuttons_conf.h"
```

Include dependency graph for uibuttons.h: This graph shows which files directly or indirectly include this file:

### Data Structures

- struct uibutton_t

### Macros

- #define UIBUTTONS_STATE_RELEASED 0
- #define UIBUTTONS_STATE_PRESSED 1
- #define UIBUTTONS_STATE_RELEASED_DEBOUNCE 2
- #define UIBUTTONS_STATE_PRESSED_DEBOUNCE 3
- #define UIBUTTONS_EVENT_BASE 0
- #define UIBUTTON(name)
- #define EXTERN_UIBUTTON(name)  extern uibutton_t name
- #define UIBUTTONS(...)
- #define UIBUTTONS_COUNT uibuttons_count
- #define UIBUTTONS_GET_ID(button)  ((int)(button-uibuttons))
- #define UIBUTTONS_POLL_SINGLE(id)  uibuttons_poll(uibuttons[id])
- #define UIBUTTONS_POLL_ALL()  do{int tmp;for(tmp=0;tmp<UIBUTTONS_COUNT;tmp++)UIBUTTONS_POLL_SINGLE(tmp
- #define UIBUTTONS_INIT_ALL()  do{int tmp;for(tmp=0;tmp<UIBUTTONS_COUNT;tmp++)uibuttons[tmp]->prev_state=uibuttons[tmp]->get();}while(0)

### Typedefs

- typedef bool(∗ uibuttons_get_t) ()

### Enumerations

- enum {
  UIBUTTONS_EVENT_RISING_EDGE = 0 , UIBUTTONS_EVENT_FALLING_EDGE , UIBUTTONS_EVENT_SHORT_PRESS
  , UIBUTTONS_EVENT_LONG_PRESS ,
  UIBUTTONS_EVENT_LONGER_PRESS , UIBUTTONS_EVENT_LONGEST_PRESS , UIBUTTONS_EVENT_REPEAT_PRES
  , UIBUTTONS_EVENT_SINGLE_CLICK ,
  UIBUTTONS_EVENT_DOUBLE_CLICK , UIBUTTONS_EVENT_TRIPLE_CLICK }

### Functions

- void uibuttons_poll (uibutton_t ∗button)

### 4.34.1 Macro Definition Documentation

#### 4.34.1.1 EXTERN_UIBUTTON

```
#define EXTERN_UIBUTTON(
              name )   extern uibutton_t name
```

Definition at line 86 of file uibuttons.h.

#### 4.34.1.2 UIBUTTON

```
#define UIBUTTON(
              name )
```

**Value:**

```
        bool uibuttons_get_##name(void); \
        uibutton_t name = {uibuttons_get_##name}; \
        bool uibuttons_get_##name(void)
```

Definition at line 81 of file uibuttons.h.

#### 4.34.1.3 UIBUTTONS

```
#define UIBUTTONS(
              ...  )
```

**Value:**

```
    const uibutton_t *uibuttons[] = { __VA_ARGS__ }; \
    const unsigned uibuttons_count = sizeof(uibuttons)/sizeof(uibuttons[0]);
```

Definition at line 89 of file uibuttons.h.

#### 4.34.1.4 UIBUTTONS_COUNT

```
#define UIBUTTONS_COUNT uibuttons_count
```

Definition at line 93 of file uibuttons.h.

#### 4.34.1.5 UIBUTTONS_EVENT_BASE

```
#define UIBUTTONS_EVENT_BASE 0
```

Definition at line 44 of file uibuttons.h.

#### 4.34.1.6 UIBUTTONS_GET_ID

```
#define UIBUTTONS_GET_ID(
            button )   ((int)(button-uibuttons))
```

Definition at line 95 of file uibuttons.h.

#### 4.34.1.7 UIBUTTONS_INIT_ALL

```
#define UIBUTTONS_INIT_ALL( )  do{int tmp;for(tmp=0;tmp<UIBUTTONS_COUNT;tmp++)uibuttons[tmp]->prev↩
_state=uibuttons[tmp]->get();}while(0)
```

Definition at line 104 of file uibuttons.h.

#### 4.34.1.8 UIBUTTONS_POLL_ALL

```
#define UIBUTTONS_POLL_ALL( )  do{int tmp;for(tmp=0;tmp<UIBUTTONS_COUNT;tmp++)UIBUTTONS_POLL_SINGLE(tmp);}whi
```

Definition at line 101 of file uibuttons.h.

#### 4.34.1.9 UIBUTTONS_POLL_SINGLE

```
#define UIBUTTONS_POLL_SINGLE(
            id )   uibuttons_poll(uibuttons[id])
```

Definition at line 98 of file uibuttons.h.

#### 4.34.1.10 UIBUTTONS_STATE_PRESSED

```
#define UIBUTTONS_STATE_PRESSED 1
```

Definition at line 40 of file uibuttons.h.

#### 4.34.1.11 UIBUTTONS_STATE_PRESSED_DEBOUNCE

```
#define UIBUTTONS_STATE_PRESSED_DEBOUNCE 3
```

Definition at line 42 of file uibuttons.h.

#### 4.34.1.12 UIBUTTONS_STATE_RELEASED

```
#define UIBUTTONS_STATE_RELEASED 0
```

**Copyright**

Definition at line 39 of file uibuttons.h.

#### 4.34.1.13 UIBUTTONS_STATE_RELEASED_DEBOUNCE

```
#define UIBUTTONS_STATE_RELEASED_DEBOUNCE 2
```

Definition at line 41 of file uibuttons.h.

### 4.34.2 Typedef Documentation

#### 4.34.2.1 uibuttons_get_t

```
typedef bool(* uibuttons_get_t) ()
```

Definition at line 60 of file uibuttons.h.

### 4.34.3 Enumeration Type Documentation

#### 4.34.3.1 anonymous enum

```
anonymous enum
```

**Enumerator**

| | |
|---|---|
| UIBUTTONS_EVENT_RISING_EDGE | |
| UIBUTTONS_EVENT_FALLING_EDGE | |
| UIBUTTONS_EVENT_SHORT_PRESS | |
| UIBUTTONS_EVENT_LONG_PRESS | |
| UIBUTTONS_EVENT_LONGER_PRESS | |
| UIBUTTONS_EVENT_LONGEST_PRESS | |
| UIBUTTONS_EVENT_REPEAT_PRESS | |
| UIBUTTONS_EVENT_SINGLE_CLICK | |
| UIBUTTONS_EVENT_DOUBLE_CLICK | |
| UIBUTTONS_EVENT_TRIPLE_CLICK | |

Definition at line 46 of file uibuttons.h.

### 4.34.4 Function Documentation

#### 4.34.4.1 uibuttons_poll()

```
void uibuttons_poll (
            uibutton_t * button )
```

Definition at line 130 of file uibuttons.c.

## 4.35 myos/ui/uibuttons_conf_template.h File Reference

### Macros

- #define UIBUTTONS_ENABLE_EDGES 1
- #define UIBUTTONS_ENABLE_SINGLE_PRESS 1
- #define UIBUTTONS_ENABLE_LONG_PRESS 1
- #define UIBUTTONS_ENABLE_REPEAT_PRESS 1
- #define UIBUTTONS_ENABLE_MULTI_CLICK 1
- #define UIBUTTONS_ENABLE_DEBOUNCING 1
- #define UIBUTTONS_DEBOUNCE_COUNT 1
- #define UIBUTTONS_LONG_PRESS_TIMEOUT 50
- #define UIBUTTONS_LONGER_PRESS_TIMEOUT 100
- #define UIBUTTONS_LONGEST_PRESS_TIMEOUT 150
- #define UIBUTTONS_REPEAT_DELAY 50
- #define UIBUTTONS_REPEAT_RATE 20
- #define UIBUTTONS_CLICK_TIMEOUT 10

### 4.35.1 Macro Definition Documentation

#### 4.35.1.1 UIBUTTONS_CLICK_TIMEOUT

```
#define UIBUTTONS_CLICK_TIMEOUT 10
```

Definition at line 46 of file uibuttons_conf_template.h.

#### 4.35.1.2 UIBUTTONS_DEBOUNCE_COUNT

```
#define UIBUTTONS_DEBOUNCE_COUNT 1
```

Definition at line 40 of file uibuttons_conf_template.h.

#### 4.35.1.3 UIBUTTONS_ENABLE_DEBOUNCING

```
#define UIBUTTONS_ENABLE_DEBOUNCING 1
```

Definition at line 38 of file uibuttons_conf_template.h.

#### 4.35.1.4 UIBUTTONS_ENABLE_EDGES

```
#define UIBUTTONS_ENABLE_EDGES 1
```

**Copyright**

Definition at line 33 of file uibuttons_conf_template.h.

#### 4.35.1.5 UIBUTTONS_ENABLE_LONG_PRESS

```
#define UIBUTTONS_ENABLE_LONG_PRESS 1
```

Definition at line 35 of file uibuttons_conf_template.h.

#### 4.35.1.6 UIBUTTONS_ENABLE_MULTI_CLICK

```
#define UIBUTTONS_ENABLE_MULTI_CLICK 1
```

Definition at line 37 of file uibuttons_conf_template.h.

#### 4.35.1.7 UIBUTTONS_ENABLE_REPEAT_PRESS

```
#define UIBUTTONS_ENABLE_REPEAT_PRESS 1
```

Definition at line 36 of file uibuttons_conf_template.h.

#### 4.35.1.8 UIBUTTONS_ENABLE_SINGLE_PRESS

```
#define UIBUTTONS_ENABLE_SINGLE_PRESS 1
```

Definition at line 34 of file uibuttons_conf_template.h.

#### 4.35.1.9 UIBUTTONS_LONG_PRESS_TIMEOUT

```
#define UIBUTTONS_LONG_PRESS_TIMEOUT 50
```

Definition at line 41 of file uibuttons_conf_template.h.

#### 4.35.1.10 UIBUTTONS_LONGER_PRESS_TIMEOUT

```
#define UIBUTTONS_LONGER_PRESS_TIMEOUT 100
```

Definition at line 42 of file uibuttons_conf_template.h.

#### 4.35.1.11 UIBUTTONS_LONGEST_PRESS_TIMEOUT

```
#define UIBUTTONS_LONGEST_PRESS_TIMEOUT 150
```

Definition at line 43 of file uibuttons_conf_template.h.

#### 4.35.1.12 UIBUTTONS_REPEAT_DELAY

```
#define UIBUTTONS_REPEAT_DELAY 50
```

Definition at line 44 of file uibuttons_conf_template.h.

#### 4.35.1.13 UIBUTTONS_REPEAT_RATE

```
#define UIBUTTONS_REPEAT_RATE 20
```

Definition at line 45 of file uibuttons_conf_template.h.

## 4.36 myos/ui/uibuttons_process.c File Reference

```
#include "myos.h"
#include "uibuttons_process.h"
```

Include dependency graph for uibuttons_process.c:

### Functions

- PROCESS (uibuttons_process, uibuttons_process)
- PROCESS_THREAD (uibuttons_process)
- void uibuttons_process_init (void)

### Variables

- uibutton_t ∗ uibuttons [ ]
- const int uibuttons_count

### 4.36.1 Function Documentation

#### 4.36.1.1 PROCESS()

```
PROCESS (
            uibuttons_process ,
            uibuttons_process  )
```

#### 4.36.1.2 PROCESS_THREAD()

```
PROCESS_THREAD (
            uibuttons_process  )
```

Definition at line 37 of file uibuttons_process.c.

Here is the call graph for this function:

#### 4.36.1.3 uibuttons_process_init()

```
void uibuttons_process_init (
            void  )
```

Definition at line 64 of file uibuttons_process.c.

Here is the call graph for this function:

### 4.36.2 Variable Documentation

#### 4.36.2.1 uibuttons

```
uibutton_t* uibuttons[]  [extern]
```

**Copyright**

#### 4.36.2.2 uibuttons_count

```
const int uibuttons_count  [extern]
```

## 4.37 myos/ui/uibuttons_process.h File Reference

```
#include "uibuttons.h"
```

Include dependency graph for uibuttons_process.h: This graph shows which files directly or indirectly include this file:

### Functions

- PROCESS_EXTERN (uibuttons_process)

### 4.37.1 Function Documentation

#### 4.37.1.1 PROCESS_EXTERN()

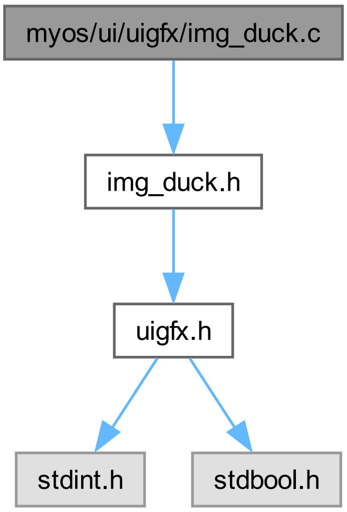
```
PROCESS_EXTERN (
            uibuttons_process  )
```

**Copyright**

## 4.38 myos/ui/uigfx/img_duck.c File Reference

```
#include "img_duck.h"
```

Include dependency graph for img_duck.c:



### Variables

- const uigfx_image_t img_duck = { image_data_duck, 384, 192, 1 }

### 4.38.1 Variable Documentation

#### 4.38.1.1 img_duck

```
const uigfx_image_t img_duck = { image_data_duck, 384, 192, 1 }
```

**Copyright**

https://opensource.org/license/mit/

Copyright 2013-2023 Marco Bacchi marco@bacchi.at

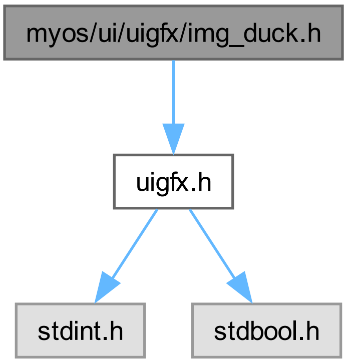
Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights

Definition at line 245 of file img_duck.c.

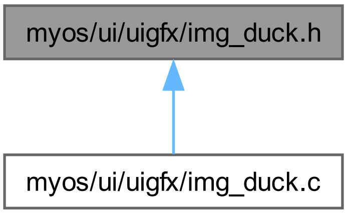
## 4.39 myos/ui/uigfx/img_duck.h File Reference

```
#include "uigfx.h"
```

Include dependency graph for img_duck.h:



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



## Variables

- const uigfx_image_t img_duck

## 4.39.1 Variable Documentation

### 4.39.1.1 img_duck

```
const uigfx_image_t img_duck [extern]
```

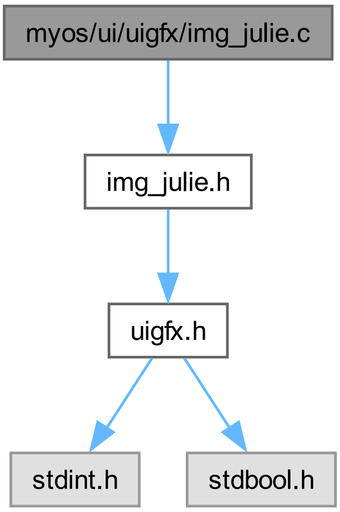
**Copyright**

Definition at line 245 of file img_duck.c.

## 4.40 myos/ui/uigfx/img_julie.c File Reference

```
#include "img_julie.h"
```

Include dependency graph for img_julie.c:



### Variables

- const uigfx_image_t img_julie = { image_data_julie, 128, 64, 1 }

### 4.40.1 Variable Documentation

#### 4.40.1.1 img_julie

```
const uigfx_image_t img_julie = { image_data_julie, 128, 64, 1 }
```

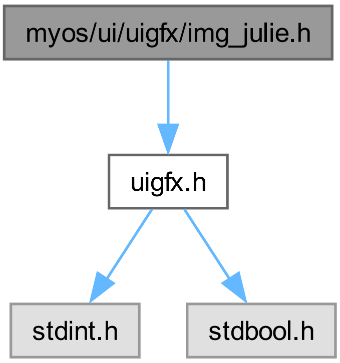
**Copyright**

Definition at line 115 of file img_julie.c.

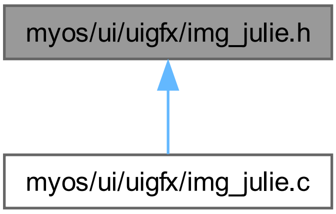
## 4.41 myos/ui/uigfx/img_julie.h File Reference

```
#include "uigfx.h"
```

Include dependency graph for img_julie.h:



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



## Variables

- const uigfx_image_t img_julie

## 4.41.1 Variable Documentation

### 4.41.1.1 img_julie

```
const uigfx_image_t img_julie [extern]
```

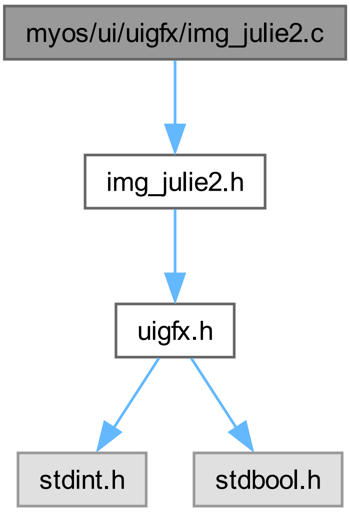
**Copyright**

Definition at line 115 of file img_julie.c.

## 4.42 myos/ui/uigfx/img_julie2.c File Reference

```
#include "img_julie2.h"
```

Include dependency graph for img_julie2.c:



### Variables

- const uigfx_image_t img_julie2 = { image_data_julie2, 128, 64, 1 }

### 4.42.1 Variable Documentation

#### 4.42.1.1 img_julie2

```
const uigfx_image_t img_julie2 = { image_data_julie2, 128, 64, 1 }
```

**Copyright**

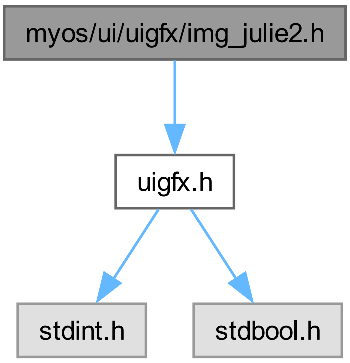
https://opensource.org/license/mit/

Definition at line 117 of file img_julie2.c.

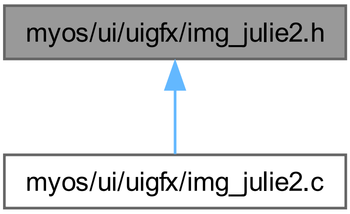
## 4.43 myos/ui/uigfx/img_julie2.h File Reference

```
#include "uigfx.h"
```

Include dependency graph for img_julie2.h:



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



## Variables

- const uigfx_image_t img_julie2

### 4.43.1 Variable Documentation

#### 4.43.1.1 img_julie2

```
const uigfx_image_t img_julie2  [extern]
```

**Copyright**

Definition at line 117 of file img_julie2.c.

## 4.44 myos/ui/uigfx/pat_chess_large.c File Reference

```
#include "pat_chess_large.h"
```

Include dependency graph for pat_chess_large.c:

## Variables

- const uigfx_image_t pat_chess_large = { image_data_pat_chess_large, 8, 8, 1 }

### 4.44.1 Variable Documentation

#### 4.44.1.1 pat_chess_large

```
const uigfx_image_t pat_chess_large = { image_data_pat_chess_large, 8, 8, 1 }
```

**Copyright**

Definition at line 59 of file pat_chess_large.c.

## 4.45 myos/ui/uigfx/pat_chess_large.h File Reference

```
#include "uigfx.h"
```

Include dependency graph for pat_chess_large.h: This graph shows which files directly or indirectly include this file:

### Variables

- const uigfx_image_t pat_chess_large

### 4.45.1 Variable Documentation

#### 4.45.1.1 pat_chess_large

```
const uigfx_image_t pat_chess_large [extern]
```

**Copyright**

https://opensource.org/license/mit/

Definition at line 59 of file pat_chess_large.c.

## 4.46 myos/ui/uigfx/pat_chess_medium.c File Reference

```
#include "pat_chess_medium.h"
```

Include dependency graph for pat_chess_medium.c:

### Variables

- const uigfx_image_t pat_chess_medium = { image_data_pat_chess_medium, 8, 8, 1 }

### 4.46.1 Variable Documentation

#### 4.46.1.1 pat_chess_medium

```
const uigfx_image_t pat_chess_medium = { image_data_pat_chess_medium, 8, 8, 1 }
```

**Copyright**

Definition at line 60 of file pat_chess_medium.c.

## 4.47 myos/ui/uigfx/pat_chess_medium.h File Reference

```
#include "uigfx.h"
```

Include dependency graph for pat_chess_medium.h: This graph shows which files directly or indirectly include this file:

### Variables

- const uigfx_image_t pat_chess_medium

### 4.47.1 Variable Documentation

#### 4.47.1.1 pat_chess_medium

```
const uigfx_image_t pat_chess_medium [extern]
```

**Copyright**

```
https://opensource.org/license/mit/
```

Definition at line 60 of file pat_chess_medium.c.

## 4.48 myos/ui/uigfx/pat_chess_small.c File Reference

```
#include "pat_chess_small.h"
```

Include dependency graph for pat_chess_small.c:

### Variables

- const uigfx_image_t pat_chess_small = { image_data_pat_chess_small, 8, 8, 1 }

### 4.48.1 Variable Documentation

#### 4.48.1.1 pat_chess_small

```
const uigfx_image_t pat_chess_small = { image_data_pat_chess_small, 8, 8, 1 }
```

**Copyright**

Definition at line 59 of file pat_chess_small.c.

## 4.49 myos/ui/uigfx/pat_chess_small.h File Reference

```
#include "uigfx.h"
```

Include dependency graph for pat_chess_small.h: This graph shows which files directly or indirectly include this file:

### Variables

- const uigfx_image_t pat_chess_small

### 4.49.1 Variable Documentation

#### 4.49.1.1 pat_chess_small

```
const uigfx_image_t pat_chess_small [extern]
```

**Copyright**

Definition at line 59 of file pat_chess_small.c.

## 4.50 myos/ui/uigfx/pat_egypt.c File Reference

```
#include "pat_egypt.h"
```

Include dependency graph for pat_egypt.c:

### Variables

- const uigfx_image_t pat_egypt = { image_data_pat_egypt, 8, 8, 1 }

### 4.50.1 Variable Documentation

#### 4.50.1.1 pat_egypt

```
const uigfx_image_t pat_egypt = { image_data_pat_egypt, 8, 8, 1 }
```

**Copyright**

Definition at line 60 of file pat_egypt.c.

## 4.51 myos/ui/uigfx/pat_egypt.h File Reference

```
#include "uigfx.h"
```

Include dependency graph for pat_egypt.h: This graph shows which files directly or indirectly include this file:

### Variables

- const uigfx_image_t pat_egypt

### 4.51.1 Variable Documentation

#### 4.51.1.1 pat_egypt

```
const uigfx_image_t pat_egypt [extern]
```

**Copyright**

Definition at line 60 of file pat_egypt.c.

## 4.52 myos/ui/uigfx/uigfx.c File Reference

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "uigfx.h"
#include "uigfx_win_conf.h"
```

Include dependency graph for uigfx.c:

### Functions

- void uigfx_set_widget (uigfx_widget_t ∗widget)
- void uigfx_select_screen_widget ()
- void uigfx_init_widget (uigfx_widget_t ∗widget, int16_t xpos, int16_t ypos, uint16_t xres, uint16_t yres)
- void uigfx_draw_pixel (uint8_t ∗buff, int16_t x, int16_t y, uigfx_color_t c)
- void uigfx_draw_hline (uint8_t ∗buf, int16_t x0, int16_t y0, int16_t w, uigfx_color_t c)
- void uigfx_draw_vline (uint8_t ∗buf, int16_t x0, int16_t y0, int16_t h, uigfx_color_t c)
- void uigfx_draw_line (uint8_t ∗buf, int16_t x0, int16_t y0, int16_t x1, int16_t y1, uigfx_color_t c)
- void uigfx_draw_rectangle (uint8_t ∗buf, int16_t x0, int16_t y0, int16_t l, int16_t h, uigfx_color_t c)
- void uigfx_draw_filled_rectangle (uint8_t ∗buf, int16_t x0, int16_t y0, int16_t l, int16_t h, uigfx_color_t c)
- void uigfx_draw_circle (uint8_t ∗buf, int16_t x0, int16_t y0, int16_t r, uigfx_color_t c)
- void uigfx_draw_filled_circle (uint8_t ∗buf, int16_t x0, int16_t y0, int16_t r, uigfx_color_t c)
- void uigfx_draw_ellipse (uint8_t ∗buf, int16_t x0, int16_t y0, int16_t a, int16_t b, uigfx_color_t c)
- void uigfx_draw_filled_ellipse (uint8_t ∗buf, int16_t x0, int16_t y0, int16_t a, int16_t b, uigfx_color_t c)
- void uigfx_draw_char (uint8_t ∗buf, uigfx_font_t ∗font, int16_t x0, int16_t y0, char ch, uigfx_color_t c)
- int16_t uigfx_draw_put_char (uint8_t ∗buf, uigfx_font_t ∗font, int16_t x0, int16_t y0, char ch, uigfx_color_t c)
- void uigfx_draw_string (uint8_t ∗buf, uigfx_font_t ∗font, int16_t x0, int16_t y0, char ∗str, uigfx_color_t c)
- int16_t uigfx_string_newline (uigfx_font_t ∗font, int16_t y0)
- void uigfx_clear (uint8_t ∗buf, uigfx_color_t c)
- void uigfx_draw_image (uint8_t ∗buf, uigfx_image_t ∗image, int16_t x0, int16_t y0)
- void uigfx_draw_widget (uint8_t ∗buf, uigfx_widget_t ∗widget)

## Variables

- const uigfx_widget_t uigfx_screen_widget = { 0, 0, 128 , 64 }
- uigfx_widget_t ∗ uigfx_current_widget = &uigfx_screen_widget

## 4.52.1 Function Documentation

### 4.52.1.1 uigfx_clear()

```
void uigfx_clear (
             uint8_t ∗ buf,
             uigfx_color_t c )
```

Definition at line 374 of file uigfx.c.

Here is the call graph for this function:

### 4.52.1.2 uigfx_draw_char()

```
void uigfx_draw_char (
             uint8_t ∗ buf,
             uigfx_font_t ∗ font,
             int16_t x0,
             int16_t y0,
             char ch,
             uigfx_color_t c )
```

Definition at line 348 of file uigfx.c.

Here is the caller graph for this function:

### 4.52.1.3 uigfx_draw_circle()

```
void uigfx_draw_circle (
             uint8_t ∗ buf,
             int16_t x0,
             int16_t y0,
             int16_t r,
             uigfx_color_t c )
```

Definition at line 212 of file uigfx.c.

Here is the call graph for this function:

#### 4.52.1.4 uigfx_draw_ellipse()

```
void uigfx_draw_ellipse (
             uint8_t * buf,
             int16_t x0,
             int16_t y0,
             int16_t a,
             int16_t b,
             uigfx_color_t c )
```

Definition at line 275 of file uigfx.c.

Here is the call graph for this function:

#### 4.52.1.5 uigfx_draw_filled_circle()

```
void uigfx_draw_filled_circle (
             uint8_t * buf,
             int16_t x0,
             int16_t y0,
             int16_t r,
             uigfx_color_t c )
```

Definition at line 243 of file uigfx.c.

Here is the call graph for this function:

#### 4.52.1.6 uigfx_draw_filled_ellipse()

```
void uigfx_draw_filled_ellipse (
             uint8_t * buf,
             int16_t x0,
             int16_t y0,
             int16_t a,
             int16_t b,
             uigfx_color_t c )
```

Definition at line 312 of file uigfx.c.

Here is the call graph for this function:

#### 4.52.1.7 uigfx_draw_filled_rectangle()

```
void uigfx_draw_filled_rectangle (
             uint8_t * buf,
             int16_t x0,
             int16_t y0,
             int16_t l,
             int16_t h,
             uigfx_color_t c )
```

Definition at line 187 of file uigfx.c.

Here is the call graph for this function: Here is the caller graph for this function:

#### 4.52.1.8 uigfx_draw_hline()

```
void uigfx_draw_hline (
             uint8_t * buf,
             int16_t x0,
             int16_t y0,
             int16_t w,
             uigfx_color_t c )
```

Definition at line 93 of file uigfx.c.

Here is the call graph for this function: Here is the caller graph for this function:

#### 4.52.1.9 uigfx_draw_image()

```
void uigfx_draw_image (
             uint8_t * buf,
             uigfx_image_t * image,
             int16_t x0,
             int16_t y0 )
```

Definition at line 388 of file uigfx.c.

Here is the call graph for this function: Here is the caller graph for this function:

#### 4.52.1.10 uigfx_draw_line()

```
void uigfx_draw_line (
             uint8_t * buf,
             int16_t x0,
             int16_t y0,
             int16_t x1,
             int16_t y1,
             uigfx_color_t c )
```

Definition at line 139 of file uigfx.c.

Here is the call graph for this function:

#### 4.52.1.11 uigfx_draw_pixel()

```
void uigfx_draw_pixel (
             uint8_t * buff,
             int16_t x,
             int16_t y,
             uigfx_color_t c )
```

Definition at line 64 of file uigfx.c.

Here is the caller graph for this function:

#### 4.52.1.12 uigfx_draw_put_char()

```
int16_t uigfx_draw_put_char (
            uint8_t * buf,
            uigfx_font_t * font,
            int16_t x0,
            int16_t y0,
            char ch,
            uigfx_color_t c )
```

Definition at line 353 of file uigfx.c.

#### 4.52.1.13 uigfx_draw_rectangle()

```
void uigfx_draw_rectangle (
            uint8_t * buf,
            int16_t x0,
            int16_t y0,
            int16_t l,
            int16_t h,
            uigfx_color_t c )
```

Definition at line 158 of file uigfx.c.

Here is the call graph for this function: Here is the caller graph for this function:

#### 4.52.1.14 uigfx_draw_string()

```
void uigfx_draw_string (
            uint8_t * buf,
            uigfx_font_t * font,
            int16_t x0,
            int16_t y0,
            char * str,
            uigfx_color_t c )
```

Definition at line 360 of file uigfx.c.

Here is the call graph for this function: Here is the caller graph for this function:

#### 4.52.1.15 uigfx_draw_vline()

```
void uigfx_draw_vline (
            uint8_t * buf,
            int16_t x0,
            int16_t y0,
            int16_t h,
            uigfx_color_t c )
```

Definition at line 117 of file uigfx.c.

Here is the call graph for this function: Here is the caller graph for this function:

#### 4.52.1.16 uigfx_draw_widget()

```
void uigfx_draw_widget (
             uint8_t * buf,
             uigfx_widget_t * widget )
```

Definition at line 412 of file uigfx.c.

#### 4.52.1.17 uigfx_init_widget()

```
void uigfx_init_widget (
             uigfx_widget_t * widget,
             int16_t xpos,
             int16_t ypos,
             uint16_t xres,
             uint16_t yres )
```

Definition at line 53 of file uigfx.c.

#### 4.52.1.18 uigfx_select_screen_widget()

```
void uigfx_select_screen_widget ( )
```

Definition at line 47 of file uigfx.c.

#### 4.52.1.19 uigfx_set_widget()

```
void uigfx_set_widget (
             uigfx_widget_t * widget )
```

Definition at line 42 of file uigfx.c.

Here is the caller graph for this function:

#### 4.52.1.20 uigfx_string_newline()

```
int16_t uigfx_string_newline (
             uigfx_font_t * font,
             int16_t y0 )
```

Definition at line 369 of file uigfx.c.

### 4.52.2 Variable Documentation

#### 4.52.2.1 uigfx_current_widget

```
uigfx_widget_t* uigfx_current_widget = &uigfx_screen_widget
```

Definition at line 40 of file uigfx.c.

#### 4.52.2.2 uigfx_screen_widget

```
const uigfx_widget_t uigfx_screen_widget = { 0, 0, 128 , 64 }
```

**Copyright**

Definition at line 39 of file uigfx.c.

## 4.53 myos/ui/uigfx/uigfx.h File Reference

```
#include <stdint.h>
#include <stdbool.h>
```

Include dependency graph for uigfx.h: This graph shows which files directly or indirectly include this file:

## Data Structures

- struct uigfx_font_t
- struct uigfx_widget_t
- struct uigfx_image_t

## Macros

- #define UIGFX_XRES 128
- #define UIGFX_YRES 64
- #define UIGFX_BPP 1

## Typedefs

- typedef bool uigfx_color_t

## Functions

- void uigfx_set_widget (uigfx_widget_t ∗widget)
- void uigfx_init_widget (uigfx_widget_t ∗widget, int16_t xpos, int16_t ypos, uint16_t xres, uint16_t yres)
- void uigfx_select_screen_widget ()
- void uigfx_draw_pixel (uint8_t ∗buff, int16_t x, int16_t y, uigfx_color_t c)
- void uigfx_draw_hline (uint8_t ∗buf, int16_t x0, int16_t y0, int16_t w, uigfx_color_t c)
- void uigfx_draw_vline (uint8_t ∗buf, int16_t x0, int16_t y0, int16_t h, uigfx_color_t c)
- void uigfx_draw_line (uint8_t ∗buf, int16_t x0, int16_t y0, int16_t x1, int16_t y1, uigfx_color_t c)
- void uigfx_draw_rectangle (uint8_t ∗buf, int16_t x0, int16_t y0, int16_t l, int16_t h, uigfx_color_t c)
- void uigfx_draw_filled_rectangle (uint8_t ∗buf, int16_t x0, int16_t y0, int16_t l, int16_t h, uigfx_color_t c)
- void uigfx_draw_circle (uint8_t ∗buf, int16_t x0, int16_t y0, int16_t r, uigfx_color_t c)
- void uigfx_draw_filled_circle (uint8_t ∗buf, int16_t x0, int16_t y0, int16_t r, uigfx_color_t c)
- void uigfx_draw_ellipse (uint8_t ∗buf, int16_t x0, int16_t y0, int16_t a, int16_t b, uigfx_color_t c)
- void uigfx_draw_filled_ellipse (uint8_t ∗buf, int16_t x0, int16_t y0, int16_t a, int16_t b, uigfx_color_t c)
- void uigfx_draw_char (uint8_t ∗buf, uigfx_font_t ∗font, int16_t x0, int16_t y0, char ch, uigfx_color_t c)
- void uigfx_draw_string (uint8_t ∗buf, uigfx_font_t ∗font, int16_t x0, int16_t y0, char ∗str, uigfx_color_t c)
- int16_t uigfx_draw_put_char (uint8_t ∗buf, uigfx_font_t ∗font, int16_t x0, int16_t y0, char ch, uigfx_color_t c)
- int16_t uigfx_string_newline (uigfx_font_t ∗font, int16_t y0)
- void uigfx_draw_image (uint8_t ∗buf, uigfx_image_t ∗image, int16_t x0, int16_t y0)

## Variables

- const uigfx_widget_t uigfx_screen_widget
- uigfx_widget_t ∗ uigfx_current_widget

### 4.53.1 Macro Definition Documentation

#### 4.53.1.1 UIGFX_BPP

```
#define UIGFX_BPP 1
```

Definition at line 38 of file uigfx.h.

#### 4.53.1.2 UIGFX_XRES

```
#define UIGFX_XRES 128
```

**Copyright**

Definition at line 36 of file uigfx.h.

#### 4.53.1.3 UIGFX_YRES

```
#define UIGFX_YRES 64
```

Definition at line 37 of file uigfx.h.

### 4.53.2 Typedef Documentation

#### 4.53.2.1 uigfx_color_t

```
typedef bool uigfx_color_t
```

Definition at line 41 of file uigfx.h.

### 4.53.3 Function Documentation

#### 4.53.3.1 uigfx_draw_char()

```
void uigfx_draw_char (
             uint8_t * buf,
             uigfx_font_t * font,
             int16_t x0,
             int16_t y0,
             char ch,
             uigfx_color_t c )
```

Definition at line 348 of file uigfx.c.

Here is the caller graph for this function:

#### 4.53.3.2 uigfx_draw_circle()

```
void uigfx_draw_circle (
             uint8_t * buf,
             int16_t x0,
             int16_t y0,
             int16_t r,
             uigfx_color_t c )
```

Definition at line 212 of file uigfx.c.

Here is the call graph for this function:

#### 4.53.3.3 uigfx_draw_ellipse()

```
void uigfx_draw_ellipse (
             uint8_t * buf,
             int16_t x0,
             int16_t y0,
             int16_t a,
             int16_t b,
             uigfx_color_t c )
```

Definition at line 275 of file uigfx.c.

Here is the call graph for this function:

#### 4.53.3.4 uigfx_draw_filled_circle()

```
void uigfx_draw_filled_circle (
             uint8_t * buf,
             int16_t x0,
             int16_t y0,
             int16_t r,
             uigfx_color_t c )
```

Definition at line 243 of file uigfx.c.

Here is the call graph for this function:

#### 4.53.3.5 uigfx_draw_filled_ellipse()

```
void uigfx_draw_filled_ellipse (
             uint8_t * buf,
             int16_t x0,
             int16_t y0,
             int16_t a,
             int16_t b,
             uigfx_color_t c )
```

Definition at line 312 of file uigfx.c.

Here is the call graph for this function:

#### 4.53.3.6 uigfx_draw_filled_rectangle()

```
void uigfx_draw_filled_rectangle (
             uint8_t * buf,
             int16_t x0,
             int16_t y0,
             int16_t l,
             int16_t h,
             uigfx_color_t c )
```

Definition at line 187 of file uigfx.c.

Here is the call graph for this function: Here is the caller graph for this function:

#### 4.53.3.7 uigfx_draw_hline()

```
void uigfx_draw_hline (
             uint8_t * buf,
             int16_t x0,
             int16_t y0,
             int16_t w,
             uigfx_color_t c )
```

Definition at line 93 of file uigfx.c.

Here is the call graph for this function: Here is the caller graph for this function:

#### 4.53.3.8 uigfx_draw_image()

```
void uigfx_draw_image (
             uint8_t * buf,
             uigfx_image_t * image,
             int16_t x0,
             int16_t y0 )
```

Definition at line 388 of file uigfx.c.

Here is the call graph for this function: Here is the caller graph for this function:

#### 4.53.3.9 uigfx_draw_line()

```
void uigfx_draw_line (
             uint8_t * buf,
             int16_t x0,
             int16_t y0,
             int16_t x1,
             int16_t y1,
             uigfx_color_t c )
```

Definition at line 139 of file uigfx.c.

Here is the call graph for this function:

#### 4.53.3.10 uigfx_draw_pixel()

```
void uigfx_draw_pixel (
             uint8_t * buff,
             int16_t x,
             int16_t y,
             uigfx_color_t c )
```

Definition at line 64 of file uigfx.c.

Here is the caller graph for this function:

#### 4.53.3.11 uigfx_draw_put_char()

```
int16_t uigfx_draw_put_char (
             uint8_t * buf,
             uigfx_font_t * font,
             int16_t x0,
             int16_t y0,
             char ch,
             uigfx_color_t c )
```

Definition at line 353 of file uigfx.c.

#### 4.53.3.12 uigfx_draw_rectangle()

```
void uigfx_draw_rectangle (
             uint8_t * buf,
             int16_t x0,
             int16_t y0,
             int16_t l,
             int16_t h,
             uigfx_color_t c )
```

Definition at line 158 of file uigfx.c.

Here is the call graph for this function: Here is the caller graph for this function:

#### 4.53.3.13 uigfx_draw_string()

```
void uigfx_draw_string (
             uint8_t * buf,
             uigfx_font_t * font,
             int16_t x0,
             int16_t y0,
             char * str,
             uigfx_color_t c )
```

Definition at line 360 of file uigfx.c.

Here is the call graph for this function: Here is the caller graph for this function:

#### 4.53.3.14 uigfx_draw_vline()

```
void uigfx_draw_vline (
             uint8_t * buf,
             int16_t x0,
             int16_t y0,
             int16_t h,
             uigfx_color_t c )
```

Definition at line 117 of file uigfx.c.

Here is the call graph for this function: Here is the caller graph for this function:

#### 4.53.3.15 uigfx_init_widget()

```
void uigfx_init_widget (
             uigfx_widget_t * widget,
             int16_t xpos,
             int16_t ypos,
             uint16_t xres,
             uint16_t yres )
```

Definition at line 53 of file uigfx.c.

#### 4.53.3.16 uigfx_select_screen_widget()

```
void uigfx_select_screen_widget ( )
```

Definition at line 47 of file uigfx.c.

#### 4.53.3.17 uigfx_set_widget()

```
void uigfx_set_widget (
             uigfx_widget_t * widget )
```

Definition at line 42 of file uigfx.c.

Here is the caller graph for this function:

#### 4.53.3.18 uigfx_string_newline()

```
int16_t uigfx_string_newline (
             uigfx_font_t * font,
             int16_t y0 )
```

Definition at line 369 of file uigfx.c.

### 4.53.4 Variable Documentation

#### 4.53.4.1 uigfx_current_widget

```
uigfx_widget_t* uigfx_current_widget
```

Definition at line 76 of file uigfx.h.

#### 4.53.4.2 uigfx_screen_widget

```
const uigfx_widget_t uigfx_screen_widget
```

Definition at line 75 of file uigfx.h.

## 4.54 myos/ui/uigfx/uigfx_font4x6.c File Reference

```
#include "uigfx_font4x6.h"
```

Include dependency graph for uigfx_font4x6.c:

## Variables

- const uigfx_font_t uigfx_font4x6 = {4,6,uigfx_font4x6_data,uigfx_draw_char_font4x6}

### 4.54.1 Variable Documentation

#### 4.54.1.1 uigfx_font4x6

```
const uigfx_font_t uigfx_font4x6 = {4,6,uigfx_font4x6_data,uigfx_draw_char_font4x6}
```

**Copyright**

Definition at line 161 of file uigfx_font4x6.c.

## 4.55 myos/ui/uigfx/uigfx_font4x6.h File Reference

```
#include "uigfx.h"
```

Include dependency graph for uigfx_font4x6.h: This graph shows which files directly or indirectly include this file:

## Variables

- const uigfx_font_t uigfx_font4x6

### 4.55.1 Variable Documentation

#### 4.55.1.1 uigfx_font4x6

```
const uigfx_font_t uigfx_font4x6  [extern]
```

**Copyright**

Definition at line 161 of file uigfx_font4x6.c.

## 4.56 myos/ui/uigfx/uigfx_font8x8_c64.c File Reference

```
#include "uigfx_font8x8_c64.h"
```

Include dependency graph for uigfx_font8x8_c64.c:

### Variables

- const uigfx_font_t uigfx_font8x8_c64 = {8,8,uigfx_font8x8_c64_data,uigfx_draw_char_font8x8_c64}

### 4.56.1 Variable Documentation

#### 4.56.1.1 uigfx_font8x8_c64

```
const uigfx_font_t uigfx_font8x8_c64 = {8,8,uigfx_font8x8_c64_data,uigfx_draw_char_font8x8_↩
c64}
```

**Copyright**

Definition at line 1007 of file uigfx_font8x8_c64.c.

## 4.57 myos/ui/uigfx/uigfx_font8x8_c64.h File Reference

```
#include "uigfx.h"
```

Include dependency graph for uigfx_font8x8_c64.h: This graph shows which files directly or indirectly include this file:

### Variables

- const uigfx_font_t uigfx_font8x8_c64

### 4.57.1 Variable Documentation

#### 4.57.1.1 uigfx_font8x8_c64

```
const uigfx_font_t uigfx_font8x8_c64 [extern]
```

**Copyright**

Definition at line 1007 of file uigfx_font8x8_c64.c.

## 4.58 myos/ui/uigfx/uigfx_font8x8_vic.c File Reference

```
#include "uigfx_font8x8_vic.h"
```

Include dependency graph for uigfx_font8x8_vic.c:

### Variables

- const uigfx_font_t uigfx_font8x8_vic = {8,8,uigfx_font8x8_vic_data,uigfx_draw_char_font8x8_vic}

### 4.58.1 Variable Documentation

#### 4.58.1.1 uigfx_font8x8_vic

```
const uigfx_font_t uigfx_font8x8_vic = {8,8,uigfx_font8x8_vic_data,uigfx_draw_char_font8x8_↩
vic}
```

**Copyright**

Definition at line 1006 of file uigfx_font8x8_vic.c.

## 4.59 myos/ui/uigfx/uigfx_font8x8_vic.h File Reference

```
#include "uigfx.h"
```

Include dependency graph for uigfx_font8x8_vic.h: This graph shows which files directly or indirectly include this file:

### Variables

- const uigfx_font_t uigfx_font8x8_vic

### 4.59.1 Variable Documentation

#### 4.59.1.1 uigfx_font8x8_vic

```
const uigfx_font_t uigfx_font8x8_vic  [extern]
```

**Copyright**

Definition at line 1006 of file uigfx_font8x8_vic.c.

## 4.60 myos/ui/uigfx/uigfx_win.c File Reference

```
#include "uigfx_win.h"
#include "uigfx_win_conf.h"
#include "string.h"
#include "uigfx.h"
```

Include dependency graph for uigfx_win.c:

### Functions

- void uigfx_draw_desktop_wallpaper (uint8_t ∗buff, uigfx_image_t ∗wallpaper)
- void uigfx_draw_window (uint8_t ∗buff, uigfx_window_t ∗win)
- void uigfx_init_window (uigfx_window_t ∗win, char ∗title, int16_t xpos, int16_t ypos, uint16_t xres, uint16_t yres)
- void uigfx_select_window_widget (uigfx_window_t ∗win)

### 4.60.1 Function Documentation

#### 4.60.1.1 uigfx_draw_desktop_wallpaper()

```
void uigfx_draw_desktop_wallpaper (
             uint8_t * buff,
             uigfx_image_t * wallpaper )
```

**Copyright**

Definition at line 37 of file uigfx_win.c.

Here is the call graph for this function:

#### 4.60.1.2 uigfx_draw_window()

```
void uigfx_draw_window (
             uint8_t * buff,
             uigfx_window_t * win )
```

Definition at line 55 of file uigfx_win.c.

Here is the call graph for this function:

#### 4.60.1.3 uigfx_init_window()

```
void uigfx_init_window (
             uigfx_window_t * win,
             char * title,
             int16_t xpos,
             int16_t ypos,
             uint16_t xres,
             uint16_t yres )
```

Definition at line 67 of file uigfx_win.c.

#### 4.60.1.4 uigfx_select_window_widget()

```
void uigfx_select_window_widget (
            uigfx_window_t * win )
```

Definition at line 91 of file uigfx_win.c.

## 4.61 myos/ui/uigfx/uigfx_win.h File Reference

```
#include "uigfx.h"
```

Include dependency graph for uigfx_win.h: This graph shows which files directly or indirectly include this file:

### Data Structures

- struct uigfx_window_t

### Functions

- void uigfx_draw_window (uint8_t ∗buff, uigfx_window_t ∗win)
- void uigfx_init_window (uigfx_window_t ∗win, char ∗title, int16_t xpos, int16_t ypos, uint16_t xres, uint16_t yres)
- uigfx_widget_t ∗ uigfx_get_window_widget (uigfx_window_t ∗win)
- void uigfx_draw_desktop_wallpaper (uint8_t ∗buff, uigfx_image_t ∗wallpaper)

### 4.61.1 Function Documentation

#### 4.61.1.1 uigfx_draw_desktop_wallpaper()

```
void uigfx_draw_desktop_wallpaper (
            uint8_t * buff,
            uigfx_image_t * wallpaper )
```

**Copyright**

Definition at line 37 of file uigfx_win.c.

Here is the call graph for this function:

#### 4.61.1.2 uigfx_draw_window()

```
void uigfx_draw_window (
             uint8_t * buff,
             uigfx_window_t * win )
```

Definition at line 55 of file uigfx_win.c.

Here is the call graph for this function:

#### 4.61.1.3 uigfx_get_window_widget()

```
uigfx_widget_t* uigfx_get_window_widget (
             uigfx_window_t * win )
```

#### 4.61.1.4 uigfx_init_window()

```
void uigfx_init_window (
             uigfx_window_t * win,
             char * title,
             int16_t xpos,
             int16_t ypos,
             uint16_t xres,
             uint16_t yres )
```

Definition at line 67 of file uigfx_win.c.

## 4.62 myos/ui/uigfx/uigfx_win_conf.h File Reference

```
#include "fonts/uigfx_font4x6.h"
#include "fonts/uigfx_font8x8_c64.h"
#include "fonts/uigfx_font8x8_vic.h"
```

Include dependency graph for uigfx_win_conf.h: This graph shows which files directly or indirectly include this file:

### Macros

- #define UIGFX_COLOR_BLACK 0
- #define UIGFX_COLOR_WHITE 1
- #define UIGFX_WIN_TITLE_FONT uigfx_font8x8_c64
- #define UIGFX_WIN_TITLE_FONT_COLOR UIGFX_COLOR_BLACK
- #define UIGFX_WIN_TITLE_BG_COLOR UIGFX_COLOR_WHITE
- #define UIGFX_WIN_TITLE_BORDER 1
- #define UIGFX_WIN_BORDER_COLOR UIGFX_WIN_TITLE_BG_COLOR
- #define UIGFX_WIN_WIDGET_BG_COLOR UIGFX_COLOR_BLACK

## 4.62.1 Macro Definition Documentation

### 4.62.1.1 UIGFX_COLOR_BLACK

```
#define UIGFX_COLOR_BLACK 0
```

**Copyright**

Definition at line 37 of file uigfx_win_conf.h.

### 4.62.1.2 UIGFX_COLOR_WHITE

```
#define UIGFX_COLOR_WHITE 1
```

Definition at line 38 of file uigfx_win_conf.h.

### 4.62.1.3 UIGFX_WIN_BORDER_COLOR

```
#define UIGFX_WIN_BORDER_COLOR UIGFX_WIN_TITLE_BG_COLOR
```

Definition at line 45 of file uigfx_win_conf.h.

#### 4.62.1.4 UIGFX_WIN_TITLE_BG_COLOR

```
#define UIGFX_WIN_TITLE_BG_COLOR UIGFX_COLOR_WHITE
```

Definition at line 42 of file uigfx_win_conf.h.

#### 4.62.1.5 UIGFX_WIN_TITLE_BORDER

```
#define UIGFX_WIN_TITLE_BORDER 1
```

Definition at line 43 of file uigfx_win_conf.h.

#### 4.62.1.6 UIGFX_WIN_TITLE_FONT

```
#define UIGFX_WIN_TITLE_FONT uigfx_font8x8_c64
```

Definition at line 40 of file uigfx_win_conf.h.

#### 4.62.1.7 UIGFX_WIN_TITLE_FONT_COLOR

```
#define UIGFX_WIN_TITLE_FONT_COLOR UIGFX_COLOR_BLACK
```

Definition at line 41 of file uigfx_win_conf.h.

#### 4.62.1.8 UIGFX_WIN_WIDGET_BG_COLOR

```
#define UIGFX_WIN_WIDGET_BG_COLOR UIGFX_COLOR_BLACK
```

Definition at line 46 of file uigfx_win_conf.h.

## 4.63 myos/ui/uileds.c File Reference

```
#include "uileds.h"
```

Include dependency graph for uileds.c:

### Functions

- void uileds_set_pattern (uileds_t ∗led, uileds_state_t ∗pattern, bool inverted)
- void uileds_sync (uileds_t ∗which, uileds_t ∗with, bool inverted)
- void uileds_handler ()

## 4.63.1 Function Documentation

### 4.63.1.1 uileds_handler()

```
void uileds_handler (
            void  )
```

Definition at line 50 of file uileds.c.

Here is the caller graph for this function:

### 4.63.1.2 uileds_set_pattern()

```
void uileds_set_pattern (
            uileds_t * led,
            uileds_state_t * pattern,
            bool inverted )
```

**Copyright**

https://opensource.org/license/mit/

Copyright 2013-2023 Marco Bacchi marco@bacchi.at

Definition at line 34 of file uileds.c.

### 4.63.1.3 uileds_sync()

```
void uileds_sync (
            uileds_t * which,
            uileds_t * with,
            bool inverted )
```

Definition at line 41 of file uileds.c.

## 4.64 myos/ui/uileds.h File Reference

```
#include <stdint.h>
#include <stdbool.h>
#include <stddef.h>
```

Include dependency graph for uileds.h: This graph shows which files directly or indirectly include this file:

### Data Structures

- struct uileds_state_t
- struct uileds_t

### Macros

- #define UILEDS_STATE_OFF 0
- #define UILEDS_STATE_ON 1
- #define UILEDS_INV 1
- #define UILEDS_ORIG 0
- #define UILEDS_PATTERN(name, size)  uileds_state_t uileds_pattern_##name [size]
- #define UILEDS_PATTERN_BEGIN(name)  const uileds_state_t uileds_pattern_##name [ ] = {
- #define EXTERN_UILEDS_PATTERN(name)  extern const uileds_state_t uileds_pattern_##name [ ]
- #define UILEDS_STATE(state, duration)  {state,duration},
- #define UILEDS_PATTERN_END() {0,0} };
- #define UILEDS_LED(name)
- #define EXTERN_UILEDS_LED(name)  extern uileds_t name
- #define UILEDS_SET_PATTERN(led_name, pattern_name, invert)  uileds_set_pattern(&led_name,(uileds_state_t∗)&uileds↩_pattern_##pattern_name,invert)
- #define UILEDS_LEDS(...)
- #define UILEDS_COUNT (uileds_count)
- #define UILEDS_SYNC(which, with, inverted) uileds_sync (&which,&with,inverted)
- #define ULEDS_DISABLE_LED(led_name) do { led_name.pattern = NULL; led_name.set_led(UILEDS_STATE_OFF);}while(0)

### Typedefs

- typedef void(∗ uileds_set_t) (bool)
- typedef struct uileds_t uileds_t

### Functions

- void uileds_set_pattern (uileds_t ∗led, uileds_state_t ∗pattern, bool inverted)
- void uileds_handler (void)
- void uileds_sync (uileds_t ∗which, uileds_t ∗with, bool inverted)

### Variables

- const uileds_t ∗const uileds_all_leds [ ]
- const unsigned uileds_count

## 4.64.1 Macro Definition Documentation

### 4.64.1.1 EXTERN_UILEDS_LED

```
#define EXTERN_UILEDS_LED(
              name )  extern uileds_t name
```

Definition at line 83 of file uileds.h.

### 4.64.1.2 EXTERN_UILEDS_PATTERN

```
#define EXTERN_UILEDS_PATTERN(
              name )  extern const uileds_state_t uileds_pattern_##name []
```

Definition at line 70 of file uileds.h.

### 4.64.1.3 UILEDS_COUNT

```
#define UILEDS_COUNT (uileds_count)
```

Definition at line 93 of file uileds.h.

### 4.64.1.4 UILEDS_INV

```
#define UILEDS_INV 1
```

Definition at line 41 of file uileds.h.

### 4.64.1.5 UILEDS_LED

```
#define UILEDS_LED(
              name )
```

**Value:**

```
void uileds_set_##name(bool state); \
uileds_t name = {uileds_set_##name}; \
void uileds_set_##name(bool state)
```

Definition at line 78 of file uileds.h.

#### 4.64.1.6 UILEDS_LEDS

```
#define UILEDS_LEDS(
              ... )
```

**Value:**

```
    const uileds_t* const uileds_all_leds [] = { __VA_ARGS__ }; \
    const unsigned uileds_count = (sizeof(uileds_all_leds)/sizeof(*uileds_all_leds));
```

Definition at line 89 of file uileds.h.

#### 4.64.1.7 UILEDS_ORIG

```
#define UILEDS_ORIG 0
```

Definition at line 42 of file uileds.h.

#### 4.64.1.8 UILEDS_PATTERN

```
#define UILEDS_PATTERN(
              name,
              size )  uileds_state_t uileds_pattern_##name [size]
```

Definition at line 63 of file uileds.h.

#### 4.64.1.9 UILEDS_PATTERN_BEGIN

```
#define UILEDS_PATTERN_BEGIN(
              name )  const uileds_state_t uileds_pattern_##name [] = {
```

Definition at line 67 of file uileds.h.

#### 4.64.1.10 UILEDS_PATTERN_END

```
#define UILEDS_PATTERN_END( ) {0,0} };
```

Definition at line 76 of file uileds.h.

#### 4.64.1.11 UILEDS_SET_PATTERN

```
#define UILEDS_SET_PATTERN(
              led_name,
              pattern_name,
              invert )  uileds_set_pattern(&led_name,(uileds_state_t*)&uileds_pattern_##pattern↩
_name,invert)
```

Definition at line 86 of file uileds.h.

#### 4.64.1.12 UILEDS_STATE

```
#define UILEDS_STATE(
              state,
              duration )  {state,duration},
```

Definition at line 73 of file uileds.h.

#### 4.64.1.13 UILEDS_STATE_OFF

```
#define UILEDS_STATE_OFF 0
```

**Copyright**

Definition at line 38 of file uileds.h.

#### 4.64.1.14 UILEDS_STATE_ON

```
#define UILEDS_STATE_ON 1
```

Definition at line 39 of file uileds.h.

#### 4.64.1.15 UILEDS_SYNC

```
#define UILEDS_SYNC(
              which,
              with,
              inverted ) uileds_sync (&which,&with,inverted)
```

Definition at line 95 of file uileds.h.

#### 4.64.1.16 ULEDS_DISABLE_LED

```
#define ULEDS_DISABLE_LED(
              led_name ) do { led_name.pattern = NULL; led_name.set_led(UILEDS_STATE_OFF);}while(0)
```

Definition at line 97 of file uileds.h.

## 4.64.2 Typedef Documentation

#### 4.64.2.1 uileds_set_t

```
typedef void(* uileds_set_t) (bool)
```

Definition at line 44 of file uileds.h.

#### 4.64.2.2 uileds_t

```
typedef struct uileds_t uileds_t
```

Definition at line 44 of file uileds.h.

## 4.64.3 Function Documentation

#### 4.64.3.1 uileds_handler()

```
void uileds_handler (
            void )
```

Definition at line 50 of file uileds.c.

Here is the caller graph for this function:

#### 4.64.3.2 uileds_set_pattern()

```
void uileds_set_pattern (
            uileds_t * led,
            uileds_state_t * pattern,
            bool inverted )
```

**Copyright**

https://opensource.org/license/mit/

Copyright 2013-2023 Marco Bacchi marco@bacchi.at

Definition at line 34 of file uileds.c.

#### 4.64.3.3 uileds_sync()

```
void uileds_sync (
            uileds_t * which,
            uileds_t * with,
            bool inverted )
```

Definition at line 41 of file uileds.c.

### 4.64.4 Variable Documentation

#### 4.64.4.1 uileds_all_leds

```
const uileds_t* const uileds_all_leds[ ]  [extern]
```

#### 4.64.4.2 uileds_count

```
const unsigned uileds_count  [extern]
```

## 4.65 myos/ui/uileds_patterns.c File Reference

```
#include "uileds_patterns.h"
```

Include dependency graph for uileds_patterns.c:

### Variables

- const uileds_state_t uileds_pattern_on [ ]
- const uileds_state_t uileds_pattern_off [ ]
- const uileds_state_t uileds_pattern_fast_flashing [ ]
- const uileds_state_t uileds_pattern_medium_flashing [ ]
- const uileds_state_t uileds_pattern_slow_flashing [ ]
- const uileds_state_t uileds_pattern_single_flash [ ]
- const uileds_state_t uileds_pattern_double_flash [ ]
- const uileds_state_t uileds_pattern_heart_beat [ ]
- const uileds_state_t uileds_pattern_triple_flash [ ]

### 4.65.1 Variable Documentation

#### 4.65.1.1 uileds_pattern_double_flash

```
const uileds_state_t uileds_pattern_double_flash[ ]
```

**Initial value:**

```
= {
    {  1  , 6 },
    {  0  , 6 },
    {  1  , 6 },
    {  0  , 82 },
{0,0} }
```

Definition at line 60 of file uileds_patterns.c.

#### 4.65.1.2 uileds_pattern_fast_flashing

```
const uileds_state_t uileds_pattern_fast_flashing[ ]
```

**Initial value:**

```
= {
    {  1  , 10 },
    {  0  , 10 },
{0,0} }
```

Definition at line 40 of file uileds_patterns.c.

#### 4.65.1.3 uileds_pattern_heart_beat

```
const uileds_state_t uileds_pattern_heart_beat[ ]
```

**Initial value:**

```
= {
    {  1  , 8 },
    {  0  , 6 },
    {  1  , 3 },
    {  0  , 83 },
{0,0} }
```

Definition at line 67 of file uileds_patterns.c.

#### 4.65.1.4 uileds_pattern_medium_flashing

```
const uileds_state_t uileds_pattern_medium_flashing[ ]
```

**Initial value:**

```
= {
    {  1  , 20 },
    {  0  , 20 },
{0,0} }
```

Definition at line 45 of file uileds_patterns.c.

#### 4.65.1.5 uileds_pattern_off

```
const uileds_state_t uileds_pattern_off[ ]
```

**Initial value:**

```
= {
    {  0  , 1 },
{0,0} }
```

Definition at line 36 of file uileds_patterns.c.

#### 4.65.1.6 uileds_pattern_on

```
const uileds_state_t uileds_pattern_on[]
```

**Initial value:**

```
= {
    {  1  , 1 },
{0,0} }
```

**Copyright**

https://opensource.org/license/mit/

Definition at line 32 of file uileds_patterns.c.

#### 4.65.1.7 uileds_pattern_single_flash

```
const uileds_state_t uileds_pattern_single_flash[]
```

**Initial value:**

```
= {
    {  1  , 6 },
    {  0  , 94 },
{0,0} }
```

Definition at line 55 of file uileds_patterns.c.

#### 4.65.1.8 uileds_pattern_slow_flashing

```
const uileds_state_t uileds_pattern_slow_flashing[]
```

**Initial value:**

```
= {
    {  1  , 40 },
    {  0  , 40 },
{0,0} }
```

Definition at line 50 of file uileds_patterns.c.

#### 4.65.1.9 uileds_pattern_triple_flash

```
const uileds_state_t uileds_pattern_triple_flash[ ]
```

**Initial value:**

```
= {
    {  1  , 6 },
    {  0  , 6 },
    {  1  , 6 },
    {  0  , 6 },
    {  1  , 6 },
    {  0  , 70 },
{0,0} }
```

Definition at line 74 of file uileds_patterns.c.

## 4.66 myos/ui/uileds_patterns.h File Reference

```
#include "uileds.h"
```

Include dependency graph for uileds_patterns.h: This graph shows which files directly or indirectly include this file:

### Variables

- const uileds_state_t uileds_pattern_on [ ]
- const uileds_state_t uileds_pattern_off [ ]
- const uileds_state_t uileds_pattern_fast_flashing [ ]
- const uileds_state_t uileds_pattern_medium_flashing [ ]
- const uileds_state_t uileds_pattern_slow_flashing [ ]
- const uileds_state_t uileds_pattern_single_flash [ ]
- const uileds_state_t uileds_pattern_double_flash [ ]
- const uileds_state_t uileds_pattern_heart_beat [ ]
- const uileds_state_t uileds_pattern_triple_flash [ ]

### 4.66.1 Variable Documentation

#### 4.66.1.1 uileds_pattern_double_flash

```
const uileds_state_t uileds_pattern_double_flash[ ]  [extern]
```

Definition at line 60 of file uileds_patterns.c.

#### 4.66.1.2 uileds_pattern_fast_flashing

```
const uileds_state_t uileds_pattern_fast_flashing[ ]  [extern]
```

Definition at line 40 of file uileds_patterns.c.

#### 4.66.1.3 uileds_pattern_heart_beat

```
const uileds_state_t uileds_pattern_heart_beat[ ]  [extern]
```

Definition at line 67 of file uileds_patterns.c.

#### 4.66.1.4 uileds_pattern_medium_flashing

```
const uileds_state_t uileds_pattern_medium_flashing[ ]  [extern]
```

Definition at line 45 of file uileds_patterns.c.

#### 4.66.1.5 uileds_pattern_off

```
const uileds_state_t uileds_pattern_off[ ]  [extern]
```

Definition at line 36 of file uileds_patterns.c.

#### 4.66.1.6 uileds_pattern_on

```
const uileds_state_t uileds_pattern_on[ ]  [extern]
```

**Copyright**

Definition at line 32 of file uileds_patterns.c.

#### 4.66.1.7 uileds_pattern_single_flash

```
const uileds_state_t uileds_pattern_single_flash[ ]  [extern]
```

Definition at line 55 of file uileds_patterns.c.

#### 4.66.1.8 uileds_pattern_slow_flashing

```
const uileds_state_t uileds_pattern_slow_flashing[ ]  [extern]
```

Definition at line 50 of file uileds_patterns.c.

#### 4.66.1.9 uileds_pattern_triple_flash

```
const uileds_state_t uileds_pattern_triple_flash[ ]  [extern]
```

Definition at line 74 of file uileds_patterns.c.

## 4.67 myos/ui/uileds_process.c File Reference

```
#include "uileds_process.h"
#include "etimer.h"
```

Include dependency graph for uileds_process.c:

### Functions

- PROCESS (uileds_process, uileds_process)
- PROCESS_THREAD (uileds_process)

### 4.67.1 Function Documentation

#### 4.67.1.1 PROCESS()

```
PROCESS (
            uileds_process ,
            uileds_process  )
```

**Copyright**

#### 4.67.1.2 PROCESS_THREAD()

```
PROCESS_THREAD (
            uileds_process  )
```

Definition at line 34 of file uileds_process.c.

Here is the call graph for this function:

## 4.68 myos/ui/uileds_process.h File Reference

```
#include "uileds.h"
#include "process.h"
```

Include dependency graph for uileds_process.h: This graph shows which files directly or indirectly include this file:

### Functions

- PROCESS_EXTERN (uileds_process)

### 4.68.1 Function Documentation

#### 4.68.1.1 PROCESS_EXTERN()

```
PROCESS_EXTERN (
            uileds_process )
```

**Copyright**

# Index