

SYSTIMER

[Home](#)[Data Structures](#)[Data Structure Index](#)[Data Fields](#)

Data Structures

Here are the data structures with brief descriptions:

 **SYSTIMER** This structure contains pointer which is used to hold CPU instance handle and variables for priority group



SYSTIMER

[Home](#)

[Data Structures](#)

[Data Structure Index](#)

[Data Fields](#)

[Data Fields](#)

SYSTIMER Struct Reference

Detailed Description

This structure contains pointer which is used to hold CPU instance handle and variables for priority group.

Definition at line **151** of file **SYSTIMER.h**.

```
#include <SYSTIMER.h>
```

Data Fields

bool	<code>init_status</code>
------	--------------------------

Field Documentation

bool SYSTIMER::init_status

APP initialization status to ensure whether SYSTIMER_Init called or not

Definition at line **153** of file **SYSTIMER.h**.

Referenced by **SYSTIMER_Init()**.

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

- **SYSTIMER.h**



SYSTIMER

Home			
Data Structures	Data Structure Index	Data Fields	

Data Structure Index

S



SYSTIMER

S

--

SYSTIMER

Home		
Data Structures	Data Structure Index	Data Fields
All	Variables	

Here is a list of all documented struct and union fields with links to the struct/union documentation for each field:

- `init_status` : [SYSTIMER](#)

--

SYSTIMER

Home		
Data Structures	Data Structure Index	Data Fields
All	Variables	

- init_status : **SYSTIMER**

SYSTIMER

[Home](#)[File List](#)[Globals](#)

File List

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

 [SYSTIMER.c](#)

 [SYSTIMER.h](#)

SYSTIMER

Home		
File List	Globals	
Code		

[Functions](#) | [Variables](#)

SYSTIMER.c File Reference

Detailed Description

Date

2016-08-05 This file is generated by DAVE, User modification to this file will be overwritten at the next code generation.

Definition in file **SYSTIMER.c**.

```
#include "systimer.h"
```

Functions

void **SYSTIMER_Start** (void)
Starts the SysTick timer. [More...](#)

void **SYSTIMER_Stop** (void)
Stops the SysTick timer. [More...](#)

DAVE_APP_VERSION_t **SYSTIMER_GetAppVersion** ()
Get **SYSTIMER** APP version. [More...](#)

SYSTIMER_STATUS_t **SYSTIMER_Init** (**SYSTIMER_t** *handle)
Initializes **SYSTIMER** APP. [More...](#)

uint32_t **SYSTIMER_CreateTimer** (uint32_t period,
SYSTIMER_MODE_t mode,
SYSTIMER_CALLBACK_t callback, void
*args)
Creates a new software timer. This
function cannot be called from an ISR. Use
SYSTIMER_CreateTimerFromISR()
instead. [More...](#)

uint32_t **SYSTIMER_CreateTimerFromISR**
(uint32_t period, **SYSTIMER_MODE_t**
mode, **SYSTIMER_CALLBACK_t** callback,
void *args)
A version of **SYSTIMER_CreateTimer()**
that can be called from an ISR. [More...](#)

SYSTIMER_STATUS_t **SYSTIMER_StartTimer** (uint32_t id)
Starts the software timer. This function
cannot be called from an ISR. Use

SYSTIMER_StartTimerFromISR()

instead. [More...](#)

SYSTIMER_STATUS_t

SYSTIMER_StartTimerFromISR (uint32_t id)

A version of **SYSTIMER_StartTimer()** that can be called from an ISR. [More...](#)

SYSTIMER_STATUS_t

SYSTIMER_StopTimer (uint32_t id)

Stops the software timer. This function cannot be called from an ISR. Use **SYSTIMER_StopTimerFromISR()** instead. [More...](#)

SYSTIMER_STATUS_t

SYSTIMER_StopTimerFromISR (uint32_t id)

A version of **SYSTIMER_StopTimer()** that can be called from an ISR. [More...](#)

SYSTIMER_STATUS_t

SYSTIMER_RestartTimer (uint32_t id, uint32_t microsec)

Function to modify the time interval and restart the timer for the new time interval. This function cannot be called from an ISR. Use **SYSTIMER_RestartTimerFromISR()** instead. [More...](#)

SYSTIMER_STATUS_t

SYSTIMER_RestartTimerFromISR (uint32_t id, uint32_t microsec)

A version of **SYSTIMER_RestartTimer()** that can be called from an ISR. [More...](#)

SYSTIMER_STATUS_t

SYSTIMER_DeleteTimer (uint32_t id)

Deletes the software timer from the timer list. This function cannot be called from an ISR. Use

SYSTIMER_DeleteTimerFromISR()

instead. [More...](#)

SYSTIMER_STATUS_t

SYSTIMER_DeleteTimerFromISR

(uint32_t id)

A version of **SYSTIMER_DeleteTimer()** that can be called from an ISR. [More...](#)

uint32_t **SYSTIMER_GetTime** (void)

Gives the current hardware SysTick time in microsecond since start of hardware SysTick timer. [More...](#)

uint32_t **SYSTIMER_GetTickCount** (void)

Gives the SysTick count. [More...](#)

SYSTIMER_STATE_t **SYSTIMER_GetTimerState** (uint32_t id)

Gives the current state of software timer. [More...](#)

Variables

```
SYSTIMER_OBJECT_t g_timer_tbl [SYSTIMER_CFG_MAX_TMR]
```

Function Documentation

void SYSTIMER_Start (void)

Starts the SysTick timer.

Parameters

None.

Description:

Starts the SysTick timer.

Definition at line **162** of file **SYSTIMER.c**.

void SYSTIMER_Stop (void)

Stops the SysTick timer.

Parameters

None.

Description:

Stops the SysTick timer therefore no software timer will time out.

Definition at line **167** of file **SYSTIMER.c**.

Variable Documentation

SYSTIMER_OBJECT_t g_timer_tbl[SYSTIMER_CFG_MAX_TMR]

Table which save timer control block.

Definition at line **110** of file **SYSTIMER.c**.

Referenced by **SYSTIMER_CreateTimer()**,
SYSTIMER_DeleteTimer(), **SYSTIMER_GetTimerState()**,
SYSTIMER_RestartTimer(), **SYSTIMER_StartTimer()**, and
SYSTIMER_StopTimer().

[Go to the source code of this file.](#)



SYSTIMER

Home		
File List	Globals	
Code		

Data Structures

SYSTIMER.h File Reference

Detailed Description

Date

2016-12-12

NOTE: This file is generated by DAVE. Any manual modification done to this file will be lost when the code is regenerated.

Definition in file **SYSTIMER.h**.

```
#include "xmc_common.h" #include <DAVE_Common.h>
#include "systimer_conf.h"
#include "systimer_extern.h"
```

Data Structures

struct **SYSTIMER**

This structure contains pointer which is used to hold CPU instance handle and variables for priority group. [More...](#)

Typedefs

```
typedef void(* SYSTIMER_CALLBACK_t)(void *args)
```

timer callback function pointer

```
typedef struct SYSTIMER SYSTIMER_t
```

This structure contains pointer which is used to hold CPU instance handle and variables for priority group.

Functions

DAVE_APP_VERSION_t	SYSTIMER_GetAppVersion (void) Get SYSTIMER APP version. More...
SYSTIMER_STATUS_t	SYSTIMER_Init (SYSTIMER_t *h) Initializes SYSTIMER APP. More...
void	SYSTIMER_Start (void) Starts the SysTick timer. More...
void	SYSTIMER_Stop (void) Stops the SysTick timer. More...
uint32_t	SYSTIMER_CreateTimer (uint32_t period, SYSTIMER_MODE_t mode, SYSTIMER_CALLBACK_t callback, void *args) Creates a new software timer. This function cannot be called from an ISR. Use SYSTIMER_CreateTimerFromISR instead. More...
uint32_t	SYSTIMER_CreateTimerFromISR (uint32_t period, SYSTIMER_MODE_t mode, SYSTIMER_CALLBACK_t callback, void *args) A version of SYSTIMER_CreateTimer() that can be called from an ISR. More...
SYSTIMER_STATUS_t	SYSTIMER_StartTimer (uint32_t period, SYSTIMER_MODE_t mode, SYSTIMER_CALLBACK_t callback, void *args)

	Starts the software timer. This function cannot be called from an ISR. Use SYSTIMER_StartTimerFromISR() instead. More...
SYSTIMER_STATUS_t	SYSTIMER_StartTimerFromISR (uint32_t id) A version of SYSTIMER_StartTimer() that can be called from an ISR. More...
SYSTIMER_STATUS_t	SYSTIMER_StopTimer (uint32_t id) Stops the software timer. This function cannot be called from an ISR. Use SYSTIMER_StopTimerFromISR() instead. More...
SYSTIMER_STATUS_t	SYSTIMER_StopTimerFromISR (uint32_t id) A version of SYSTIMER_StopTimer() that can be called from an ISR. More...
SYSTIMER_STATUS_t	SYSTIMER_RestartTimer (uint32_t id, uint32_t microsec) Function to modify the time interval and restart the timer for the new time interval. This function cannot be called from an ISR. Use SYSTIMER_RestartTimerFromISR() instead. More...
SYSTIMER_STATUS_t	SYSTIMER_RestartTimerFromISR (uint32_t id, uint32_t microsec) A version of SYSTIMER_RestartTimer() that can be called from an ISR. More...

be called from an ISR. [More...](#)

SYSTIMER_STATUS_t **SYSTIMER_DeleteTimer** (uint32_t id)
Deletes the software timer from the timer list. This function cannot be called from an ISR. Use **SYSTIMER_DeleteTimerFromISR** instead. [More...](#)

SYSTIMER_STATUS_t **SYSTIMER_DeleteTimerFromISR** (uint32_t id)
A version of **SYSTIMER_DeleteTimer()** that can be called from an ISR. [More...](#)

uint32_t **SYSTIMER_GetTime** (void)
Gives the current hardware SysTick time in microsecond since start of hardware SysTick timer. [More...](#)

uint32_t **SYSTIMER_GetTickCount** (void)
Gives the SysTick count. [More...](#)

SYSTIMER_STATE_t **SYSTIMER_GetTimerState** (uint32_t id)
Gives the current state of software timer. [More...](#)

enum **SYSTIMER_STATUS** {
 SYSTIMER_STATUS_SUCCESS
 SYSTIMER_STATUS_FAILURE }
This enumeration indicates status of **SYSTIMER**. [More...](#)

SYSTIMER_STATE {

enum **SYSTIMER_STATE_NOT_INITIAL**
= 0U, SYSTIMER_STATE_RUNN
SYSTIMER_STATE_STOPPED }
This enumeration defines possible timer state. [More...](#)

enum **SYSTIMER_MODE {**
SYSTIMER_MODE_ONE_SHOT :
SYSTIMER_MODE_PERIODIC }
Enumeration values which describ timer types. [More...](#)

typedef enum **SYSTIMER_STATUS** **SYSTIMER_STATUS_t**
This enumeration indicates status **SYSTIMER**.

typedef enum **SYSTIMER_STATE** **SYSTIMER_STATE_t**
This enumeration defines possible timer state.

typedef enum **SYSTIMER_MODE** **SYSTIMER_MODE_t**
Enumeration values which describ timer types.

[Go to the source code of this file.](#)

SYSTIMER

Home						
File List		Globals				
All		Functions	Variables	Typedefs	Enumerations	Enumerator
g	s					

Here is a list of all documented functions, variables, defines, enums, and typedefs with links to the documentation:

- g -

- `g_timer_tbl` : [SYSTIMER.c](#)

- s -

- `SYSTIMER_CALLBACK_t` : [SYSTIMER.h](#)
- `SYSTIMER_CreateTimer()` : [SYSTIMER.c](#) , [SYSTIMER.h](#)
- `SYSTIMER_CreateTimerFromISR()` : [SYSTIMER.c](#) , [SYSTIMER.h](#)
- `SYSTIMER_DeleteTimer()` : [SYSTIMER.c](#) , [SYSTIMER.h](#)
- `SYSTIMER_DeleteTimerFromISR()` : [SYSTIMER.c](#) , [SYSTIMER.h](#)
- `SYSTIMER_GetAppVersion()` : [SYSTIMER.c](#) , [SYSTIMER.h](#)
- `SYSTIMER_GetTickCount()` : [SYSTIMER.c](#) , [SYSTIMER.h](#)
- `SYSTIMER_GetTime()` : [SYSTIMER.c](#) , [SYSTIMER.h](#)
- `SYSTIMER_GetTimerState()` : [SYSTIMER.c](#) , [SYSTIMER.h](#)
- `SYSTIMER_Init()` : [SYSTIMER.c](#) , [SYSTIMER.h](#)
- `SYSTIMER_MODE` : [SYSTIMER.h](#)
- `SYSTIMER_MODE_ONE_SHOT` : [SYSTIMER.h](#)
- `SYSTIMER_MODE_PERIODIC` : [SYSTIMER.h](#)
- `SYSTIMER_MODE_t` : [SYSTIMER.h](#)
- `SYSTIMER_RestartTimer()` : [SYSTIMER.c](#) , [SYSTIMER.h](#)
- `SYSTIMER_RestartTimerFromISR()` : [SYSTIMER.h](#) , [SYSTIMER.c](#)
- `SYSTIMER_Start()` : [SYSTIMER.c](#) , [SYSTIMER.h](#)
- `SYSTIMER_StartTimer()` : [SYSTIMER.h](#) , [SYSTIMER.c](#)
- `SYSTIMER_StartTimerFromISR()` : [SYSTIMER.c](#) , [SYSTIMER.h](#)
- `SYSTIMER_STATE` : [SYSTIMER.h](#)

- SYSTIMER_STATE_NOT_INITIALIZED : **SYSTIMER.h**
 - SYSTIMER_STATE_RUNNING : **SYSTIMER.h**
 - SYSTIMER_STATE_STOPPED : **SYSTIMER.h**
 - SYSTIMER_STATE_t : **SYSTIMER.h**
 - SYSTIMER_STATUS : **SYSTIMER.h**
 - SYSTIMER_STATUS_FAILURE : **SYSTIMER.h**
 - SYSTIMER_STATUS_SUCCESS : **SYSTIMER.h**
 - SYSTIMER_STATUS_t : **SYSTIMER.h**
 - SYSTIMER_Stop() : **SYSTIMER.c** , **SYSTIMER.h**
 - SYSTIMER_StopTimer() : **SYSTIMER.c** , **SYSTIMER.h**
 - SYSTIMER_StopTimerFromISR() : **SYSTIMER.h** , **SYSTIMER.c**
 - SYSTIMER_t : **SYSTIMER.h**
-
-

SYSTIMER

Home					
File List	Globals				
All	Functions	Variables	Typedefs	Enumerations	Enumerator
s					

- S -

- SYSTIMER_CreateTimer() : [SYSTIMER.c](#) , [SYSTIMER.h](#)
- SYSTIMER_CreateTimerFromISR() : [SYSTIMER.h](#) , [SYSTIMER.c](#)
- SYSTIMER_DeleteTimer() : [SYSTIMER.c](#) , [SYSTIMER.h](#)
- SYSTIMER_DeleteTimerFromISR() : [SYSTIMER.h](#) , [SYSTIMER.c](#)
- SYSTIMER_GetAppVersion() : [SYSTIMER.c](#) , [SYSTIMER.h](#)
- SYSTIMER_GetTickCount() : [SYSTIMER.c](#) , [SYSTIMER.h](#)
- SYSTIMER_GetTime() : [SYSTIMER.c](#) , [SYSTIMER.h](#)
- SYSTIMER_GetTimerState() : [SYSTIMER.h](#) , [SYSTIMER.c](#)
- SYSTIMER_Init() : [SYSTIMER.c](#) , [SYSTIMER.h](#)
- SYSTIMER_RestartTimer() : [SYSTIMER.c](#) , [SYSTIMER.h](#)
- SYSTIMER_RestartTimerFromISR() : [SYSTIMER.c](#) , [SYSTIMER.h](#)
- SYSTIMER_Start() : [SYSTIMER.h](#) , [SYSTIMER.c](#)
- SYSTIMER_StartTimer() : [SYSTIMER.c](#) , [SYSTIMER.h](#)
- SYSTIMER_StartTimerFromISR() : [SYSTIMER.h](#) , [SYSTIMER.c](#)
- SYSTIMER_Stop() : [SYSTIMER.c](#) , [SYSTIMER.h](#)
- SYSTIMER_StopTimer() : [SYSTIMER.h](#) , [SYSTIMER.c](#)
- SYSTIMER_StopTimerFromISR() : [SYSTIMER.h](#) , [SYSTIMER.c](#)



SYSTIMER

Home						
File List		Globals				
All	Functions	Variables	Typedefs	Enumerations	Enumerator	

- g_timer_tbl : **SYSTIMER.c**



SYSTIMER

Home					
File List		Globals			
All	Functions	Variables	Typedefs	Enumerations	Enumerator

- SYSTIMER_CALLBACK_t : [SYSTIMER.h](#)
- SYSTIMER_MODE_t : [SYSTIMER.h](#)
- SYSTIMER_STATE_t : [SYSTIMER.h](#)
- SYSTIMER_STATUS_t : [SYSTIMER.h](#)
- SYSTIMER_t : [SYSTIMER.h](#)



SYSTIMER

Home					
File List		Globals			
All	Functions	Variables	Typedefs	Enumerations	Enumerator

- SYSTIMER_MODE : [SYSTIMER.h](#)
- SYSTIMER_STATE : [SYSTIMER.h](#)
- SYSTIMER_STATUS : [SYSTIMER.h](#)



SYSTIMER

Home					
File List	Globals				
All	Functions	Variables	Typedefs	Enumerations	Enumerator

- SYSTIMER_MODE_ONE_SHOT : [SYSTIMER.h](#)
- SYSTIMER_MODE_PERIODIC : [SYSTIMER.h](#)
- SYSTIMER_STATE_NOT_INITIALIZED : [SYSTIMER.h](#)
- SYSTIMER_STATE_RUNNING : [SYSTIMER.h](#)
- SYSTIMER_STATE_STOPPED : [SYSTIMER.h](#)
- SYSTIMER_STATUS_FAILURE : [SYSTIMER.h](#)
- SYSTIMER_STATUS_SUCCESS : [SYSTIMER.h](#)



SYSTIMER

Home		
File List	Globals	
Code		

SYSTIMER.h

[Go to the documentation of this file.](#)

1

72 /*****

73 * HEADER FILES

74

*****/

75

76 #ifndef SYSTIMER_H

77 #define SYSTIMER_H

78

79 #include "xmc_common.h"

80 #include <DAVE_Common.h>

81 #include "systimer_conf.h"

82

83 /*****

84 * MACROS

85

*****/

86

87 #if (!((XMC_LIB_MAJOR_VERSION == 2U) && \

88 (XMC_LIB_MINOR_VERSION >= 0U) && \

89 (XMC_LIB_PATCH_VERSION >= 0U)))

90 #error "SYSTIMER requires XMC Peripheral Library v2.0.0 or
higher"

91 #endif

92

93 /*****

94 * ENUMS

95

96

105 typedef enum SYSTIMER_STATUS

106 {

107 SYSTIMER_STATUS_SUCCESS = 0U,

108 SYSTIMER_STATUS_FAILURE

109 } SYSTIMER_STATUS_t;

110

114 typedef enum SYSTIMER_STATE

115 {

116 SYSTIMER_STATE_NOT_INITIALIZED = 0U,

117 SYSTIMER_STATE_RUNNING,

118 SYSTIMER_STATE_STOPPED

119 } SYSTIMER_STATE_t;

120

124 typedef enum SYSTIMER_MODE

125 {

126 SYSTIMER_MODE_ONE_SHOT = 0U,

127 SYSTIMER_MODE_PERIODIC

128 } SYSTIMER_MODE_t;

129

134 /*****

135 * DATA STRUCTURES

136 *****/

145 typedef void (*SYSTIMER_CALLBACK_t)(void *args);

146

151 typedef struct SYSTIMER

152 {

153 bool init_status;

154 } SYSTIMER_t;

159 /*****

160 * API Prototypes

161

162

```

163 /* Support for C++ codebase */
164 #ifdef __cplusplus
165 extern "C" {
166 #endif
167
200 DAVE_APP_VERSION_t SYSTIMER_GetAppVersion(void);
201
238 SYSTIMER_STATUS_t SYSTIMER_Init(SYSTIMER_t *handle);
239
247 void SYSTIMER_Start(void);
248
256 void SYSTIMER_Stop(void);
257
325 uint32_t SYSTIMER_CreateTimer
326 (
327  uint32_t period,
328  SYSTIMER_MODE_t mode,
329  SYSTIMER_CALLBACK_t callback,
330  void *args
331 );
332
344 uint32_t SYSTIMER_CreateTimerFromISR
345 (
346  uint32_t period,
347  SYSTIMER_MODE_t mode,
348  SYSTIMER_CALLBACK_t callback,
349  void *args
350 );
351
406 SYSTIMER_STATUS_t SYSTIMER_StartTimer(uint32_t id);
407
414 SYSTIMER_STATUS_t SYSTIMER_StartTimerFromISR(uint32_t
id);
415
484 SYSTIMER_STATUS_t SYSTIMER_StopTimer(uint32_t id);
485
492 SYSTIMER_STATUS_t SYSTIMER_StopTimerFromISR(uint32_t

```

```
id);
493
576 SYSTIMER_STATUS_t SYSTIMER_RestartTimer(uint32_t id,
uint32_t microsec);
577
585 SYSTIMER_STATUS_t
SYSTIMER_RestartTimerFromISR(uint32_t id, uint32_t microsec);
586
667 SYSTIMER_STATUS_t SYSTIMER_DeleteTimer(uint32_t id);
668
675 SYSTIMER_STATUS_t SYSTIMER_DeleteTimerFromISR(uint32_t
id);
676
734 uint32_t SYSTIMER_GetTime(void);
735
796 uint32_t SYSTIMER_GetTickCount(void);
797
878 SYSTIMER_STATE_t SYSTIMER_GetTimerState(uint32_t id);
879
884 /* Support for C++ codebase */
885 #ifdef __cplusplus
886 }
887 #endif
888
889 /* Inclusion of APP extern file */
890 #include "systimer_extern.h"
891
892 #endif /* SYSTIMER_H */
```

SYSTIMER

Home		
File List	Globals	
Code		

SYSTIMER.c

[Go to the documentation of this file.](#)

1

78 /*****

79 * HEADER FILES

80

*****/

81

82 /* Included to access APP data structure, functions & enumerations

*/

83 #include "systimer.h"

84

85 /*****

86 * MACROS

87

*****/

88

89 #define HW_TIMER_ADDITIONAL_CNT (1U)

90

91 /*****

92 * LOCAL DATA

93

*****/

94 /* SYSTIMER_OBJECT structure acts as the timer control block */

95

96 typedef struct SYSTIMER_OBJECT

97 {

98 struct SYSTIMER_OBJECT *next;

```

99 struct SYSTIMER_OBJECT *prev;
100 SYSTIMER_CALLBACK_t callback;
101 void *args;
102 uint32_t id;
103 uint32_t count;
104 uint32_t reload;
105 SYSTIMER_MODE_t mode;
106 SYSTIMER_STATE_t state;
107 } SYSTIMER_OBJECT_t;
108
110 SYSTIMER_OBJECT_t g_timer_tbl[SYSTIMER_CFG_MAX_TMR];
111
112 /* The header of the timer Control list. */
113 SYSTIMER_OBJECT_t *g_timer_list = NULL;
114
115 /* Timer ID tracker */
116 uint32_t g_timer_tracker = 0U;
117
118 /* SysTick counter */
119 volatile uint32_t g_systick_count = 0U;
120
121 /*****
122  * LOCAL ROUTINES
123  *****/
124
125 /*
126  * This function is called to insert a timer into the timer list.
127  */
128 static void SYSTIMER_InsertTimerList(uint32_t tbl_index);
129
130 /*
131  * This function is called to remove a timer from the timer list.
132  */
133 static void SYSTIMER_RemoveTimerList(uint32_t tbl_index);
134
135 /*

```

```

136 * Handler function called from SysTick event handler.
137 */
138 static void SYSTIMER_ITimerHandler(void);
139
140 /*
141 * SysTick handler which is the main interrupt service routine to
service the
142 * system timer's configured
143 */
144 void SysTick_Handler(void);
145
146 /*****
147 * API IMPLEMENTATION
148 *****/
149 __attribute__((always_inline)) __STATIC_INLINE uint32_t
critical_section_enter(void)
150 {
151     uint32_t status;
152     status = __get_PRIMASK();
153     __disable_irq ();
154     return status;
155 }
156
157 __attribute__((always_inline)) __STATIC_INLINE void
critical_section_exit(uint32_t status)
158 {
159     __set_PRIMASK(status);
160 }
161
162 void SYSTIMER_Start(void)
163 {
164     SysTick->CTRL |= SysTick_CTRL_ENABLE_Msk;
165 }
166
167 void SYSTIMER_Stop(void)
168 {
169     SysTick->CTRL &= ~SysTick_CTRL_ENABLE_Msk;

```

```

170 }
171
172 /*
173  * This function is called to insert a timer into the timer list.
174  */
175 static void SYSTIMER_InsertTimerList(uint32_t tbl_index)
176 {
177     SYSTIMER_OBJECT_t *object_ptr;
178     int32_t delta_ticks;
179     int32_t timer_count;
180     bool found_flag = false;
181
182     /* Get timer time */
183     timer_count = (int32_t)g_timer_tbl[tbl_index].count;
184     /* Check if Timer list is NULL */
185     if (NULL == g_timer_list)
186     {
187         /* Set this as first Timer */
188         g_timer_list = &g_timer_tbl[tbl_index];
189     }
190     /* If not, find the correct place, and insert the specified timer */
191     else
192     {
193         object_ptr = g_timer_list;
194         /* Get timer tick */
195         delta_ticks = timer_count;
196         /* Find correct place for inserting the timer */
197         while ((NULL != object_ptr) && (false == found_flag))
198         {
199             /* Get timer Count Difference */
200             delta_ticks -= (int32_t)object_ptr->count;
201             /* Check for delta ticks < 0 */
202             if (delta_ticks <= 0)
203             {
204                 /* Check If head item */
205                 if (NULL != object_ptr->prev)
206                 {

```

```

207 /* If Insert to list */
208 object_ptr->prev->next = &g_timer_tbl[tbl_index];
209 g_timer_tbl[tbl_index].prev = object_ptr->prev;
210 g_timer_tbl[tbl_index].next = object_ptr;
211 object_ptr->prev = &g_timer_tbl[tbl_index];
212 }
213 else
214 {
215 /* Set Timer as first item */
216 g_timer_tbl[tbl_index].next = g_timer_list;
217 g_timer_list->prev = &g_timer_tbl[tbl_index];
218 g_timer_list = &g_timer_tbl[tbl_index];
219 }
220 g_timer_tbl[tbl_index].count = g_timer_tbl[tbl_index].next->count +
(uint32_t)delta_ticks;
221 g_timer_tbl[tbl_index].next->count -= g_timer_tbl[tbl_index].count;
222 found_flag = true;
223 }
224 /* Check for last item in list */
225 else
226 {
227 if ((delta_ticks > 0) && (NULL == object_ptr->next))
228 {
229 /* Yes, insert into */
230 g_timer_tbl[tbl_index].prev = object_ptr;
231 object_ptr->next = &g_timer_tbl[tbl_index];
232 g_timer_tbl[tbl_index].count = (uint32_t)delta_ticks;
233 found_flag = true;
234 }
235 }
236 /* Get the next item in timer list */
237 object_ptr = object_ptr->next;
238 }
239 }
240 }
241
242 /*

```

```

243 * This function is called to remove a timer from the timer list.
244 */
245 static void SYSTIMER_IRemoveTimerList(uint32_t tbl_index)
246 {
247     SYSTIMER_OBJECT_t *object_ptr;
248
249     object_ptr = &g_timer_tbl[tbl_index];
250     /* Check whether only one timer available */
251     if ((NULL == object_ptr->prev) && (NULL == object_ptr->next ))
252     {
253         /* set timer list as NULL */
254         g_timer_list = NULL;
255     }
256     /* Check if the first item in timer list */
257     else if (NULL == object_ptr->prev)
258     {
259         /* Remove timer from list, and reset timer list */
260         g_timer_list = object_ptr->next;
261         g_timer_list->prev = NULL;
262         g_timer_list->count += object_ptr->count;
263         object_ptr->next = NULL;
264     }
265     /* Check if the last item in timer list */
266     else if (NULL == object_ptr->next)
267     {
268         /* Remove timer from list */
269         object_ptr->prev->next = NULL;
270         object_ptr->prev = NULL;
271     }
272     else
273     {
274         /* Remove timer from list */
275         object_ptr->prev->next = object_ptr->next;
276         object_ptr->next->prev = object_ptr->prev;
277         object_ptr->next->count += object_ptr->count;
278         object_ptr->next = NULL;
279         object_ptr->prev = NULL;

```

```

280 }
281
282 }
283
284 /*
285  * Handler function called from SysTick event handler.
286  */
287 static void SYSTIMER_ITimerHandler(void)
288 {
289     SYSTIMER_OBJECT_t *object_ptr;
290
291     /* Get first item of timer list */
292     object_ptr = g_timer_list;
293     while ((NULL != object_ptr) && (0U == object_ptr->count))
294     {
295         if (SYSTIMER_STATE_RUNNING == object_ptr->state)
296         {
297             /* Check whether timer is a one shot timer */
298             if (SYSTIMER_MODE_ONE_SHOT == object_ptr->mode)
299             {
300                 /* Yes, remove this timer from timer list */
301                 SYSTIMER_IRemoveTimerList((uint32_t)object_ptr->id);
302                 /* Set timer status as SYSTIMER_STATE_STOPPED */
303                 object_ptr->state = SYSTIMER_STATE_STOPPED;
304                 /* Call timer callback function */
305                 (object_ptr->callback)(object_ptr->args);
306             }
307             /* Check whether timer is periodic timer */
308             else if (SYSTIMER_MODE_PERIODIC == object_ptr->mode)
309             {
310                 /* Yes, remove this timer from timer list */
311                 SYSTIMER_IRemoveTimerList((uint32_t)object_ptr->id);
312                 /* Reset timer tick */
313                 object_ptr->count = object_ptr->reload;
314                 /* Insert timer into timer list */
315                 SYSTIMER_IInsertTimerList((uint32_t)object_ptr->id);
316                 /* Call timer callback function */

```

```

317 (object_ptr->callback)(object_ptr->args);
318 }
319 }
320 /* Get first item of timer list */
321 object_ptr = g_timer_list;
322 }
323
324 }
325
326 /*
327  * SysTick Event Handler.
328  */
329 void SysTick_Handler(void)
330 {
331     SYSTIMER_OBJECT_t *object_ptr;
332     object_ptr = g_timer_list;
333     g_systick_count++;
334
335     if (NULL != object_ptr)
336     {
337         if (object_ptr->count > 1UL)
338         {
339             object_ptr->count--;
340         }
341         else
342         {
343             object_ptr->count = 0U;
344             SYSTIMER_ITimerHandler();
345         }
346     }
347 }
348
349 /*
350  * Function to retrieve the version of the SYSTIMER APP.
351  */
352 DAVE_APP_VERSION_t SYSTIMER_GetAppVersion()
353 {

```

```

358 DAVE_APP_VERSION_t version;
359
360 version.major = (uint8_t)SYSTIMER_MAJOR_VERSION;
361 version.minor = (uint8_t)SYSTIMER_MINOR_VERSION;
362 version.patch = (uint8_t)SYSTIMER_PATCH_VERSION;
363
364 return (version);
365 }
366
367 /*
368  * Initialization function which initializes the SYSTIMER APP,
369  * configures SysTick timer and SysTick exception.
370 */
371 SYSTIMER_STATUS_t SYSTIMER_Init(SYSTIMER_t *handle)
372 {
373     SYSTIMER_STATUS_t status =
374     SYSTIMER_STATUS_SUCCESS;
375
376     XMC_ASSERT("SYSTIMER_Init: SYSTIMER APP handle pointer
377     uninitialized", (handle != NULL));
378
379     /* Check APP initialization status to ensure whether
380     SYSTIMER_Init called or not, initialize SYSTIMER if
381     SYSTIMER_Init called first time.
382     */
383     if (false == handle->init_status)
384     {
385         /* Initialize the header of the list */
386         g_timer_list = NULL;
387         /* Initialize SysTick timer */
388         status = (SYSTIMER_STATUS_t)SysTick_Config((uint32_t)
389         (SYSTIMER_SYSTICK_CLOCK * SYSTIMER_TICK_PERIOD));
390
391         if (SYSTIMER_STATUS_FAILURE == status)
392         {
393             XMC_DEBUG("SYSTIMER_Init: Timer reload value out of range");
394         }
395     }

```

```

390 else
391 {
392 #if (UC_FAMILY == XMC4)
393 /* setting of First SW Timer period is always and subpriority value
for XMC4000 devices */
394 NVIC_SetPriority(SysTick_IRQn, NVIC_EncodePriority(
395 NVIC_GetPriorityGrouping(), SYSTIMER_PRIORITY,
SYSTIMER_SUBPRIORITY));
396 #elif (UC_FAMILY == XMC1)
397 /* setting of priority value for XMC1000 devices */
398 NVIC_SetPriority(SysTick_IRQn, SYSTIMER_PRIORITY);
399 #endif
400 g_timer_tracker = 0U;
401 /* Update the Initialization status of the SYSTIMER APP instance
*/
402 handle->init_status = true;
403 status = SYSTIMER_STATUS_SUCCESS;
404 }
405 }
406
407 return (status);
408 }
409
410 /*
411 * API for creating a new software Timer instance.
412 */
413 uint32_t SYSTIMER_CreateTimer
414 (
415 uint32_t period,
416 SYSTIMER_MODE_t mode,
417 SYSTIMER_CALLBACK_t callback,
418 void *args
419 )
420 {
421 uint32_t id = 0U;
422 uint32_t count = 0U;
423 uint32_t period_ratio = 0U;

```

```
424
425 XMC_ASSERT("SYSTIMER_CreateTimer: Timer creation failure
due to invalid period value",
426 ((period >= SYSTIMER_TICK_PERIOD_US) && (period > 0U) &&
(period <= 0xFFFFFFFFU)));
427 XMC_ASSERT("SYSTIMER_CreateTimer: Timer creation failure
due to invalid timer mode",
428 ((SYSTIMER_MODE_ONE_SHOT == mode) ||
(SYSTIMER_MODE_PERIODIC == mode)));
429 XMC_ASSERT("SYSTIMER_CreateTimer: Can not create
software without user callback", (NULL != callback));
430
431 if (period < SYSTIMER_TICK_PERIOD_US)
432 {
433     id = 0U;
434 }
435 else
436 {
437     for (count = 0U; count < SYSTIMER_CFG_MAX_TMR; count++)
438     {
439         /* Check for free timer ID */
440         if (0U == (g_timer_tracker & (1U << count)))
441         {
442             /* If yes, assign ID to this timer */
443             g_timer_tracker |= (1U << count);
444             /* Initialize the timer as per input values */
445             g_timer_tbl[count].id = count;
446             g_timer_tbl[count].mode = mode;
447             g_timer_tbl[count].state = SYSTIMER_STATE_STOPPED;
448             period_ratio = (uint32_t)(period / SYSTIMER_TICK_PERIOD_US);
449             g_timer_tbl[count].count = (period_ratio +
HW_TIMER_ADDITIONAL_CNT);
450             g_timer_tbl[count].reload = period_ratio;
451             g_timer_tbl[count].callback = callback;
452             g_timer_tbl[count].args = args;
453             g_timer_tbl[count].prev = NULL;
454             g_timer_tbl[count].next = NULL;
```

```

455 id = count + 1U;
456 break;
457 }
458 }
459
460 }
461
462 return (id);
463 }
464
465 uint32_t SYSTIMER_CreateTimerFromISR
466 (
467     uint32_t period,
468     SYSTIMER_MODE_t mode,
469     SYSTIMER_CALLBACK_t callback,
470     void *args
471 )
472 {
473     uint32_t id;
474
475     uint32_t ics;
476     ics = critical_section_enter();
477
478     id = SYSTIMER_CreateTimer(period, mode, callback, args);
479
480     critical_section_exit(ics);
481
482     return (id);
483 }
484
485 /*
486  * API to start the software timer.
487  */
488 SYSTIMER_STATUS_t SYSTIMER_StartTimer(uint32_t id)
489 {
490     SYSTIMER_STATUS_t status;
491

```

```

492 status = SYSTIMER_STATUS_FAILURE;
493
494 XMC_ASSERT("SYSTIMER_StartTimer: Failure in timer restart
operation due to invalid timer ID",
495 ((id <= SYSTIMER_CFG_MAX_TMR) && (id > 0U)));
496 XMC_ASSERT("SYSTIMER_StartTimer: Error during start of
software timer", (0U != (g_timer_tracker & (1U << (id - 1U)))));
497
498 /* Check if timer is running */
499 if (SYSTIMER_STATE_STOPPED == g_timer_tbl[id - 1U].state)
500 {
501 g_timer_tbl[id - 1U].state = SYSTIMER_STATE_RUNNING;
502 g_timer_tbl[id - 1U].count = (g_timer_tbl[id - 1U].reload +
HW_TIMER_ADDITIONAL_CNT);
503
504 /* Insert this timer into timer list */
505 SYSTIMER_InsertTimerList((id - 1U));
506
507 status = SYSTIMER_STATUS_SUCCESS;
508 }
509
510 return (status);
511 }
512
513 SYSTIMER_STATUS_t SYSTIMER_StartTimerFromISR(uint32_t
id)
514 {
515 SYSTIMER_STATUS_t status;
516
517 uint32_t ics;
518 ics = critical_section_enter();
519
520 status = SYSTIMER_StartTimer(id);
521
522 critical_section_exit(ics);
523
524 return (status);

```

```

525 }
526
527 /*
528  * API to stop the software timer.
529 */
530 SYSTIMER_STATUS_t SYSTIMER_StopTimer(uint32_t id)
531 {
532     SYSTIMER_STATUS_t status;
533
534     status = SYSTIMER_STATUS_SUCCESS;
535
536     XMC_ASSERT("SYSTIMER_StopTimer: Failure in timer restart
operation due to invalid timer ID",
537 ((id <= SYSTIMER_CFG_MAX_TMR) && (id > 0U)));
538     XMC_ASSERT("SYSTIMER_StopTimer: Error during stop of
software timer", (0U != (g_timer_tracker & (1U << (id - 1U)))));
539
540     if (SYSTIMER_STATE_NOT_INITIALIZED == g_timer_tbl[id -
1U].state)
541     {
542         status = SYSTIMER_STATUS_FAILURE;
543     }
544     else
545     {
546         /* Check whether Timer is in Stop state */
547         if (SYSTIMER_STATE_RUNNING == g_timer_tbl[id - 1U].state)
548         {
549             g_timer_tbl[id - 1U].state = SYSTIMER_STATE_STOPPED;
550             /* remove Timer from node list */
551             SYSTIMER_IRemoveTimerList(id - 1U);
552         }
553     }
554
555     return (status);
556 }
557
558 SYSTIMER_STATUS_t SYSTIMER_StopTimerFromISR(uint32_t

```

```

id)
559 {
560 SYSTIMER_STATUS_t status;
561
562 uint32_t ics;
563 ics = critical_section_enter();
564
565 status = SYSTIMER_StopTimer(id);
566
567 critical_section_exit(ics);
568
569 return (status);
570 }
571
572 /*
573  * API to reinitialize the time interval and to start the timer.
574  */
575 SYSTIMER_STATUS_t SYSTIMER_RestartTimer(uint32_t id,
uint32_t microsec)
576 {
577 uint32_t period_ratio = 0U;
578 SYSTIMER_STATUS_t status;
579
580 status = SYSTIMER_STATUS_SUCCESS;
581
582 XMC_ASSERT("SYSTIMER_RestartTimer: Failure in timer restart
operation due to invalid timer ID",
583 ((id <= SYSTIMER_CFG_MAX_TMR) && (id > 0U)));
584 XMC_ASSERT("SYSTIMER_RestartTimer: Error during restart of
software timer", (0U != (g_timer_tracker & (1U << (id - 1U)))));
585 XMC_ASSERT("SYSTIMER_RestartTimer: Can not restart timer
due to invalid period value",
586 (microsec >= SYSTIMER_TICK_PERIOD_US) && (microsec >
0U));
587
588
589 if (SYSTIMER_STATE_NOT_INITIALIZED == g_timer_tbl[id -

```

```

1U].state)
590 {
591 status = SYSTIMER_STATUS_FAILURE;
592 }
593 else
594 {
595 /* check whether timer is in run state */
596 if( SYSTIMER_STATE_STOPPED != g_timer_tbl[id - 1U].state)
597 {
598 /* Stop the timer */
599 status = SYSTIMER_StopTimer(id);
600 }
601 if (SYSTIMER_STATUS_SUCCESS == status)
602 {
603 period_ratio = (uint32_t)(microsec /
SYSTIMER_TICK_PERIOD_US);
604 g_timer_tbl[id - 1U].reload = period_ratio;
605 /* Start the timer */
606 status = SYSTIMER_StartTimer(id);
607 }
608 }
609
610 return (status);
611 }
612
613 SYSTIMER_STATUS_t
SYSTIMER_RestartTimerFromISR(uint32_t id, uint32_t microsec)
614 {
615 SYSTIMER_STATUS_t status;
616
617 uint32_t ics;
618 ics = critical_section_enter();
619
620 status = SYSTIMER_RestartTimerFromISR(id, microsec)
621
622 critical_section_exit(ics);
623

```

```

624 return (status);
625 }
626
627 /*
628  * Function to delete the Timer instance.
629  */
630 SYSTIMER_STATUS_t SYSTIMER_DeleteTimer(uint32_t id)
631 {
632     SYSTIMER_STATUS_t status;
633
634     status = SYSTIMER_STATUS_SUCCESS;
635
636     XMC_ASSERT("SYSTIMER_DeleteTimer: Failure in timer restart
operation due to invalid timer ID",
637 ((id <= SYSTIMER_CFG_MAX_TMR) && (id > 0U)));
638 XMC_ASSERT("SYSTIMER_DeleteTimer: Error during deletion of
software timer", (0U != (g_timer_tracker & (1U << (id - 1U)))));
639
640 /* Check whether Timer is in delete state */
641 if (SYSTIMER_STATE_NOT_INITIALIZED == g_timer_tbl[id -
1U].state)
642 {
643     status = SYSTIMER_STATUS_FAILURE;
644 }
645 else
646 {
647     status = SYSTIMER_StopTimer(id);
648
649     /* Set timer status as SYSTIMER_STATE_NOT_INITIALIZED */
650     g_timer_tbl[id - 1U].state =
SYSTIMER_STATE_NOT_INITIALIZED;
651     /* Release resource which are hold by this timer */
652     g_timer_tracker &= ~(1U << (id - 1U));
653
654 }
655
656 return (status);

```

```

657 }
658
659 SYSTIMER_STATUS_t SYSTIMER_DeleteTimerFromISR(uint32_t
id)
660 {
661     SYSTIMER_STATUS_t status;
662
663     uint32_t ics;
664     ics = critical_section_enter();
665
666     status = SYSTIMER_DeleteTimer(id);
667
668     critical_section_exit(ics);
669
670     return (status);
671 }
672
673 /*
674  * API to get the current SysTick time in microsecond.
675  */
676 uint32_t SYSTIMER_GetTime(void)
677 {
678     return (g_systick_count * SYSTIMER_TICK_PERIOD_US);
679 }
680
681 /*
682  * API to get the SysTick count.
683  */
684 uint32_t SYSTIMER_GetTickCount(void)
685 {
686     return (g_systick_count);
687 }
688
689 /*
690  * API to get the current state of software timer.
691  */
692 SYSTIMER_STATE_t SYSTIMER_GetTimerState(uint32_t id)

```

```
693 {  
694 return (g_timer_tbl[id - 1U].state);  
695 }
```



SYSTIMER

[Home](#)

[Code](#) >

Code Directory Reference

Files

file **SYSTIMER.c** [code]

file **SYSTIMER.h** [code]



SYSTIMER

Home

Methods

SYSTIMER_STATUS_t **SYSTIMER_Init** (SYSTIMER_t *handle)
Initializes **SYSTIMER** APP. [More...](#)

SYSTIMER_CreateTimer (uint32_t period,
uint32_t **SYSTIMER_MODE_t** mode,
SYSTIMER_CALLBACK_t callback, void
*args)
Creates a new software timer. This
function cannot be called from an ISR. Use
SYSTIMER_CreateTimerFromISR()
instead. [More...](#)

SYSTIMER_CreateTimerFromISR
(uint32_t period, **SYSTIMER_MODE_t**
uint32_t mode, **SYSTIMER_CALLBACK_t** callback,
void *args)
A version of **SYSTIMER_CreateTimer()**
that can be called from an ISR. [More...](#)

SYSTIMER_STATUS_t **SYSTIMER_StartTimer** (uint32_t id)
Starts the software timer. This function
cannot be called from an ISR. Use
SYSTIMER_StartTimerFromISR()
instead. [More...](#)

SYSTIMER_STATUS_t **SYSTIMER_StartTimerFromISR** (uint32_t
id)
A version of **SYSTIMER_StartTimer()** that

can be called from an ISR. [More...](#)

SYSTIMER_STATUS_t **SYSTIMER_StopTimer** (uint32_t id)
Stops the software timer. This function cannot be called from an ISR. Use **SYSTIMER_StopTimerFromISR()** instead. [More...](#)

SYSTIMER_STATUS_t **SYSTIMER_StopTimerFromISR** (uint32_t id)
A version of **SYSTIMER_StopTimer()** that can be called from an ISR. [More...](#)

SYSTIMER_STATUS_t **SYSTIMER_RestartTimer** (uint32_t id, uint32_t microsec)
Function to modify the time interval and restart the timer for the new time interval. This function cannot be called from an ISR. Use **SYSTIMER_RestartTimerFromISR()** instead. [More...](#)

SYSTIMER_STATUS_t **SYSTIMER_RestartTimerFromISR** (uint32_t id, uint32_t microsec)
A version of **SYSTIMER_RestartTimer()** that can be called from an ISR. [More...](#)

SYSTIMER_STATUS_t **SYSTIMER_DeleteTimer** (uint32_t id)
Deletes the software timer from the timer list. This function cannot be called from an ISR. Use **SYSTIMER_DeleteTimerFromISR()** instead. [More...](#)

SYSTIMER_STATUS_t **SYSTIMER_DeleteTimerFromISR**

(uint32_t id)
A version of **SYSTIMER_DeleteTimer()**
that can be called from an ISR. [More...](#)

uint32_t **SYSTIMER_GetTime** (void)
Gives the current hardware SysTick time in
microsecond since start of hardware
SysTick timer. [More...](#)

uint32_t **SYSTIMER_GetTickCount** (void)
Gives the SysTick count. [More...](#)

SYSTIMER_STATE_t **SYSTIMER_GetTimerState** (uint32_t id)
Gives the current state of software timer.
[More...](#)

DAVE_APP_VERSION_t **SYSTIMER_GetAppVersion** (void)
Get **SYSTIMER** APP version. [More...](#)

void **SYSTIMER_Start** (void)
Starts the SysTick timer. [More...](#)

void **SYSTIMER_Stop** (void)
Stops the SysTick timer. [More...](#)

Detailed Description

Methods

Function Documentation

```
uint32_t SYSTIMER_CreateTimer ( uint32_t          period  
                                SYSTIMER_MODE_t   mode  
                                SYSTIMER_CALLBACK_t callback  
                                void *             args  
                                )
```

Creates a new software timer. This function cannot be called from an ISR. Use [SYSTIMER_CreateTimerFromISR\(\)](#) instead.

Parameters

- period** timer period value in microseconds. Range: (SYSTIMER_TICK_PERIOD_US) to pow(2,32).
- mode** Mode of timer(ONE_SHOT/PERIODIC). Refer [SYSTIMER_MODE_t](#) for details.
- callback** Call back function of the timer(No Macros are allowed).
- args** Call back function parameter.

Returns

uint32_t returns timer ID if timer created successfully otherwise returns 0 if timer creation failed. Range: 0 to 16, 0: Invalid timer ID, 1-16: Valid timer ID.

Description:

API for creating a new software timer instance. This also add created software timer to timer list.

Note :

1. This APP uses SysTick exception for controlling the timer list. Call back function registered through this function will be called in SysTick exception when the software timer is expired i.e the software timers callback is executed in the interrupt context.

1. Due to time at which software timer creation asked by user will not be in synchronize with Hardware SysTick timer, the

count value used during creation of software timer will not create starting/initial period same as expected value. It is decided to add one extra count(HW_TIMER_ADDITIONAL_CNT) with Software timer. Impact of this additional count(HW_TIMER_ADDITIONAL_CNT) is, first SW timer period(Initial one) is always equal to or more than expected/configured.

2. Callbacks are executed in round robin manner if more than one software timers are created with same period value. Last created software is having higher priority and its associated callback is executed first.
3. Avoid any call to wait, infinitive while loop, blocking calls or creating software timer in ISR because their behavior could be corrupted when called from an ISR.
4. Software timers are based on 24-bit Hardware SysTick counters, so maximum counts can achieve is $\text{pow}(2,24) * (1/\text{fCPU}) * 1\text{E}6$, where fCPU is in hertz. Software timers are designed for times between microseconds and seconds. For longer times, application code need to ensure to take necessary action.
5. Software timer period value must be equal to SysTick Interval or integer multiple of a number with SysTick interval (i.e. SysTick Interval * n, where n is integer number, n can be 1,2,3,4... but n should not be fractional or float number). And also software timer period value should not be 0 or less than Hardware SysTick Interval.

Example Usage:

```
#include <DAVE.h>
#define ONESEC 1000000U
void LED_Toggle_EverySec(void)
{
    // Add user code here
}
int main(void)
```

```

{
uint32_t TimerId;
// ... Initializes APPs configuration ...
DAVE_Init(); // SYSTIMER APP Initialized during DAVE Initialization
// Create Software timer
TimerId =
(uint32_t)SYSTIMER_CreateTimer(ONESEC,SYSTIMER_MODE_PERIODIC,
(void*)LED_Toggle_EverySec,NULL);
if (TimerId != 0U)
{
//software timer is created successfully
//Add user code here
}
else
{
// //software timer creation is failed
}
while (1)
{
}
return (1);
}

```

Definition at line **414** of file **SYSTIMER.c**.

References **g_timer_tbl**, **SYSTIMER_MODE_ONE_SHOT**, **SYSTIMER_MODE_PERIODIC**, and **SYSTIMER_STATE_STOPPED**.

Referenced by **SYSTIMER_CreateTimerFromISR()**.

uint32_t SYSTIMER_CreateTimerFromISR (uint32_t
SYSTIMER_MODE_t
SYSTIMER_CALLBACK

void *
)

A version of **SYSTIMER_CreateTimer()** that can be called from an ISR.

Parameters

- period** timer period value in microseconds. Range: (SYSTIMER_TICK_PERIOD_US) to pow(2,32).
- mode** Mode of timer(ONE_SHOT/PERIODIC). Refer **SYSTIMER_MODE_t** for details.
- callback** Call back function of the timer(No Macros are allowed).
- args** Call back function parameter.

Returns

uint32_t returns timer ID if timer created successfully otherwise returns 0 if timer creation failed. Range: 0 to 16, 0: Invalid timer ID, 1-16: Valid timer ID.

Definition at line **466** of file **SYSTIMER.c**.

References **SYSTIMER_CreateTimer()**.

SYSTIMER_STATUS_t SYSTIMER_DeleteTimer (uint32_t id)

Deletes the software timer from the timer list. This function cannot be called from an ISR. Use **SYSTIMER_DeleteTimerFromISR()** instead.

Parameters

- id** timer ID obtained from SYSTIMER_CreateTimer. Range : 1 to 16

Returns

SYSTIMER_STATUS_t APP status. Refer

SYSTIMER_STATUS_t for details.

Description:

API for deleting the created software timer instance from timer list.

Note : This API must be called after software timer is created using SYSTIMER_CreateTimer API with generated ID and enable XMC_ASSERT for better understanding of API behavioral in run time.

Example Usage:

```
#include <DAVE.h>
#define ONESEC 1000000U
void LED_Toggle_EverySec(void)
{
    // Add user code here
}
int main(void)
{
    uint32_t TimerId;
    SYSTIMER_STATUS_t status;
    // ... Initializes APPs configuration ...
    DAVE_Init(); // SYSTIMER APP Initialized during DAVE Initialization
    // Create Software timer
    TimerId =
    (uint32_t)SYSTIMER_CreateTimer(ONESEC,SYSTIMER_MODE_PERIODIC,
    (void*)LED_Toggle_EverySec,NULL);
    if (TimerId != 0U)
    {
        //timer is created successfully, now start/run software timer
        status = SYSTIMER_StartTimer(TimerId);
        if (status == SYSTIMER_STATUS_SUCCESS)
        {
            // User code
            status = SYSTIMER_StopTimer(TimerId);
            //User code
            if (status == SYSTIMER_STATUS_SUCCESS)
```

```

{
//User code
status = SYSTIMER_DeleteTimer(TimerId);
if (status == SYSTIMER_STATUS_SUCCESS)
{
// Software timer has deleted
}
else
{
// Error during software timer delete operation
}
}
else
{
// Error during software timer stop operation
}
}
else
{
// Error during software timer start operation
}
}
else
{
// timer ID Can not be zero
}
// ... infinite loop ...
while (1)
{
}
return (1);
}

```

Definition at line **630** of file **SYSTIMER.c**.

References **g_timer_tbl**, **SYSTIMER_STATE_NOT_INITIALIZED**,

SYSTIMER_STATUS_FAILURE, **SYSTIMER_STATUS_SUCCESS**, and **SYSTIMER_StopTimer()**.

Referenced by **SYSTIMER_DeleteTimerFromISR()**.

SYSTIMER_STATUS_t SYSTIMER_DeleteTimerFromISR (uint32_t id)

A version of **SYSTIMER_DeleteTimer()** that can be called from an ISR.

Parameters

id timer ID obtained from SYSTIMER_CreateTimer. Range : 1 to 16

Returns

SYSTIMER_STATUS_t APP status. Refer **SYSTIMER_STATUS_t** for details.

Definition at line **659** of file **SYSTIMER.c**.

References **SYSTIMER_DeleteTimer()**.

DAVE_APP_VERSION_t SYSTIMER_GetAppVersion (void)

Get **SYSTIMER** APP version.

Returns

DAVE_APP_VERSION_t APP version information (major, minor and patch number).

Description:

The function can be used to check application software compatibility with a specific version of the APP.

```
#include <DAVE.h>
int main(void)
```

```

{
DAVE_Init();
DAVE_APP_VERSION_t systimer_version;
systimer_version = SYSTIMER_GetAppVersion();
if ((systimer_version.major == 4U) && (systimer_version.minor ==
1U))
{
// Add application code here
while (1)
{
}
}
return(1);
}

```

Definition at line **356** of file **SYSTIMER.c**.

uint32_t SYSTIMER_GetTickCount (void)

Gives the SysTick count.

Returns

uint32_t returns SysTick count. Range: 0 to pow(2,32).

Description:

API to get hardware SysTick counts since start of hardware SysTick timer.

Example Usage:

```

#include <DAVE.h>
#include <DAVE.h>
#define ONESEC 1000000U
void LED_Toggle_EverySec(void)
{
// Add user code here
}

```

```

int main(void)
{
    uint32_t TimerId;
    uint32_t SysTick_Count;
    SYSTIMER_STATUS_t status;
    // ... Initializes APPs configuration ...
    DAVE_Init(); // SYSTIMER APP Initialized during DAVE Initialization
    // Create Software timer
    TimerId =
    (uint32_t)SYSTIMER_CreateTimer(ONESEC,SYSTIMER_MODE_PERIODIC,
    (void*)LED_Toggle_EverySec,NULL);
    if (TimerId != 0U)
    {
        //timer is created successfully, now start/run software timer
        status = SYSTIMER_StartTimer(TimerId);
        if (status == SYSTIMER_STATUS_SUCCESS)
        {
            // Add user code here
            SysTick_Count = SYSTIMER_GetTickCount();
            // Add user code here
        }
        else
        {
            // Error during software timer start operation
        }
    }
    else
    {
        // timer ID Can not be zero
    }
    // ... infinite loop ...
    while (1)
    {
    }
    return (1);
}

```

Definition at line **684** of file **SYSTIMER.c**.

uint32_t SYSTIMER_GetTime (void)

Gives the current hardware SysTick time in microsecond since start of hardware SysTick timer.

Returns

uint32_t returns current SysTick time in microsecond. Range: (SYSTIMER_TICK_PERIOD_US) to pow(2,32).

Description:

API to get current hardware SysTick time in microsecond since start of hardware SysTick timer.

Example Usage:

```
#include <DAVE.h>
#define ONESEC 1000000U
void LED_Toggle_EverySec(void)
{
    // Add user code here
}
int main(void)
{
    uint32_t TimerId;
    uint32_t SysTick_Time;
    SYSTIMER_STATUS_t status;
    // ... Initializes APPs configuration ...
    DAVE_Init(); // SYSTIMER APP Initialized during DAVE Initialization
    // Create Software timer
    TimerId =
    (uint32_t)SYSTIMER_CreateTimer(ONESEC,SYSTIMER_MODE_PERIODIC,
    (void*)LED_Toggle_EverySec,NULL);
    if (TimerId != 0U)
```

```

{
//timer is created successfully, now start/run software timer
status = SYSTIMER_StartTimer(TimerId);
if (status == SYSTIMER_STATUS_SUCCESS)
{
// Add user code here
SysTick_Time = SYSTIMER_GetTime();
// Add user code here
}
else
{
// Error during software timer start operation
}
}
else
{
// timer ID Can not be zero
}
// ... infinite loop ...
while (1)
{
}
return (1);
}

```

Definition at line **676** of file **SYSTIMER.c**.

SYSTIMER_STATE_t SYSTIMER_GetTimerState (uint32_t id)

Gives the current state of software timer.

Parameters

id timer ID obtained from SYSTIMER_CreateTimer. Range : 1 to 16

Returns

SYSTIMER_STATE_t Software timer state. Refer **SYSTIMER_STATE_t** for details.

Description:

API to get current software timer state.

Example Usage:

```
#include <DAVE.h>
#define ONESEC 1000000U
#define NEW_INTERVAL (ONESEC * 10U)
void LED_Toggle_EverySec(void)
{
    // Add user code here
}
int main(void)
{
    uint32_t TimerId;
    SYSTIMER_STATUS_t status;
    SYSTIMER_STATE_t timer_state;
    // ... Initializes APPs configuration ...
    DAVE_Init(); // SYSTIMER APP Initialized during DAVE Initialization
    // Create Software timer
    TimerId =
    (uint32_t)SYSTIMER_CreateTimer(ONESEC,SYSTIMER_MODE_PERIODIC,
    (void*)LED_Toggle_EverySec,NULL);
    if (TimerId != 0U)
    {
        //timer is created successfully, now start/run software timer
        status = SYSTIMER_StartTimer(TimerId);
        timer_state = SYSTIMER_GetTimerState(TimerId); // use case
        scenario 1
        if (timer_state == SYSTIMER_STATE_RUNNING)
        {
            // software timer start operation is successful
            // Add user code here
        }
    }
}
```

```

else
{
// Error during software timer start operation
}
// Add user code here
// user decided to change software interval, oops but user don't know
the timer state
timer_state = SYSTIMER_GetTimerState(TimerId); // use case
scenario 2
if (timer_state == SYSTIMER_STATE_RUNNING)
{
status = SYSTIMER_StopTimer(TimerId);
status = SYSTIMER_RestartTimer(TimerId,NEW_INTERVAL);
// Add user code here
}
else if (timer_state == SYSTIMER_STATE_STOPPED)
{
status = SYSTIMER_RestartTimer(TimerId,NEW_INTERVAL);
}
else if (timer_state == SYSTIMER_STATE_NOT_INITIALIZED)
{
// user has already deleted this software timer but need to recreate
TimerId =
(uint32_t)SYSTIMER_CreateTimer(NEW_INTERVAL,SYSTIMER_MODI
(void*)LED_Toggle_EverySec,NULL);
status = SYSTIMER_StartTimer(TimerId);
// Add user code here
}
}
else
{
// timer ID Can not be zero
}
// ... infinite loop ...
while (1)
{
}

```

```
return (1);  
}
```

Definition at line **692** of file **SYSTIMER.c**.

References **g_timer_tbl**.

SYSTIMER_STATUS_t SYSTIMER_Init (SYSTIMER_t * **handle)**

Initializes **SYSTIMER** APP.

Parameters

handle Pointer pointing to **SYSTIMER** APP data structure.
Refer **SYSTIMER_t** for details.

Returns

SYSTIMER_STATUS_t APP status. Refer
SYSTIMER_STATUS_t for details.

Description:

Initializes the SysTick counter as per the SysTick interval specified by the user and start the SysTick counter. It also initializes global variables.

Example Usage:

```
#include <DAVE.h> //Declarations from DAVE Code Generation  
(includes SFR declaration)  
int main(void)  
{  
    SYSTIMER_STATUS_t init_status;  
    init_status =  
    (SYSTIMER_STATUS_t)SYSTIMER_Init(&SYSTIMER_0); //  
    Initialization of SYSTIMER APP  
    if (init_status == SYSTIMER_STATUS_SUCCESS)  
    {
```

```

// Add application code here
while(1)
{
}
else
{
XMC_DEBUG("main: Application initialization failed");
while(1)
{
}
}
return (1);
}

```

Definition at line **370** of file **SYSTIMER.c**.

References **SYSTIMER::init_status**,
SYSTIMER_STATUS_FAILURE, and
SYSTIMER_STATUS_SUCCESS.

SYSTIMER_STATUS_t SYSTIMER_RestartTimer (uint32_t **id**,
uint32_t **microsec**
)

Function to modify the time interval and restart the timer for the new time interval. This function cannot be called from an ISR. Use **SYSTIMER_RestartTimerFromISR()** instead.

Parameters

id ID of already created system timer. Range : 1 to 16
microsec new time interval. Range:
(SYSTIMER_TICK_PERIOD_US) to pow(2,32).

Returns

SYSTIMER_STATUS_t APP status. Refer

SYSTIMER_STATUS_t for details.

Description:

API for restarting the created software timer instance with new time interval.

Note : This API must be called after software timer is created using SYSTIMER_CreateTimer API with generated ID and enable XMC_ASSERT for better understanding of API behavioral in run time.

Example Usage:

Demonstrate SYSTIMER_RestartTimer API

```
#include <DAVE.h>
#define ONESEC 1000000U
#define NEW_INTERVAL (ONESEC * 10U)
void LED_Toggle_EverySec(void)
{
    // Add user code here
}
int main(void)
{
    uint32_t TimerId;
    SYSTIMER_STATUS_t status;
    // ... Initializes APPs configuration ...
    DAVE_Init(); // SYSTIMER APP Initialized during DAVE Initialization
    // Create Software timer
    TimerId =
    (uint32_t)SYSTIMER_CreateTimer(ONESEC,SYSTIMER_MODE_PERIODIC,
    (void*)LED_Toggle_EverySec,NULL);
    if (TimerId != 0U)
    {
        //timer is created successfully
        // Start/Run Software timer
        status = SYSTIMER_StartTimer(TimerId);
        if (status == SYSTIMER_STATUS_SUCCESS)
        {
            // User code
        }
    }
}
```

```
status = SYSTIMER_StopTimer(TimerId);
//User code
if (status == SYSTIMER_STATUS_SUCCESS)
{
    //User code
    status = SYSTIMER_RestartTimer(TimerId,NEW_INTERVAL);
    if (status == SYSTIMER_STATUS_SUCCESS)
    {
        // timer configured with the new time interval and is running
    }
    else
    {
        // Error during software timer restart operation
    }
}
else
{
    // Error during software timer stop operation
}
}
else
{
    // Error during software timer start operation
}
}
else
{
    // timer ID can not be zero
}
while (1)
{
}
return (1);
}
```

Definition at line **575** of file **SYSTIMER.c**.

References **g_timer_tbl**, **SYSTIMER_StartTimer()**, **SYSTIMER_STATE_NOT_INITIALIZED**, **SYSTIMER_STATE_STOPPED**, **SYSTIMER_STATUS_FAILURE**, **SYSTIMER_STATUS_SUCCESS**, and **SYSTIMER_StopTimer()**.

```
SYSTIMER_STATUS_t SYSTIMER_RestartTimerFromISR ( uint32_t  
                                                    uint32_t  
                                                    )
```

A version of **SYSTIMER_RestartTimer()** that can be called from an ISR.

Parameters

id ID of already created system timer. Range : 1 to 16
microsec new time interval. Range:
(SYSTIMER_TICK_PERIOD_US) to pow(2,32).

Returns

SYSTIMER_STATUS_t APP status. Refer **SYSTIMER_STATUS_t** for details.

Definition at line **613** of file **SYSTIMER.c**.

References **SYSTIMER_RestartTimerFromISR()**.

Referenced by **SYSTIMER_RestartTimerFromISR()**.

```
void SYSTIMER_Start ( void )
```

Starts the SysTick timer.

Parameters

None.

Description:

Starts the SysTick timer.

Definition at line **162** of file **SYSTIMER.c**.

SYSTIMER_STATUS_t SYSTIMER_StartTimer (uint32_t id)

Starts the software timer. This function cannot be called from an ISR. Use **SYSTIMER_StartTimerFromISR()** instead.

Parameters

id timer ID obtained from SYSTIMER_CreateTimer. Range : 1 to 16

Returns

SYSTIMER_STATUS_t APP status. Refer **SYSTIMER_STATUS_t** for details.

Description:

API for starting a software timer instance.

Note : This API must be called after software timer is created using SYSTIMER_CreateTimer API with generated ID and enable XMC_ASSERT for better understanding of API behavioral in run time.

Example Usage:

```
#include <DAVE.h>
#define ONESEC 1000000U
void LED_Toggle_EverySec(void)
{
    // Add user code here
}
int main(void)
{
    uint32_t TimerId;
    SYSTIMER_STATUS_t status;
```

```

// ... Initializes APPs configuration ...
DAVE_Init(); // SYSTIMER APP Initialized during DAVE Initialization
// Create Software timer
TimerId =
(uint32_t)SYSTIMER_CreateTimer(ONESEC,SYSTIMER_MODE_PERIODIC,
(void*)LED_Toggle_EverySec,NULL);
if (TimerId != 0U)
{
//timer is created successfully, now start/run software timer
status = SYSTIMER_StartTimer(TimerId);
if (status == SYSTIMER_STATUS_SUCCESS)
{
// Software timer is running
// Add user code here
}
else
{
// Error during software timer start operation
}
}
else
{
// timer ID Can not be zero
}
// ... infinite loop ...
while (1)
{
}
return (1);
}

```

Definition at line **488** of file **SYSTIMER.c**.

References **g_timer_tbl**, **SYSTIMER_STATE_RUNNING**,
SYSTIMER_STATE_STOPPED, **SYSTIMER_STATUS_FAILURE**,
and **SYSTIMER_STATUS_SUCCESS**.

Referenced by [SYSTIMER_RestartTimer\(\)](#), and [SYSTIMER_StartTimerFromISR\(\)](#).

SYSTIMER_STATUS_t SYSTIMER_StartTimerFromISR (uint32_t id

A version of [SYSTIMER_StartTimer\(\)](#) that can be called from an ISR.

Parameters

id timer ID obtained from SYSTIMER_CreateTimer. Range : 1 to 16

Returns

SYSTIMER_STATUS_t APP status. Refer [SYSTIMER_STATUS_t](#) for details.

Definition at line **513** of file [SYSTIMER.c](#).

References [SYSTIMER_StartTimer\(\)](#).

void SYSTIMER_Stop (void)

Stops the SysTick timer.

Parameters

None.

Description:

Stops the SysTick timer therefore no software timer will time out.

Definition at line **167** of file [SYSTIMER.c](#).

SYSTIMER_STATUS_t SYSTIMER_StopTimer (uint32_t id)

Stops the software timer. This function cannot be called from an ISR. Use **SYSTIMER_StopTimerFromISR()** instead.

Parameters

id timer ID obtained from SYSTIMER_CreateTimer. Range : 1 to 16

Returns

SYSTIMER_STATUS_t APP status. Refer **SYSTIMER_STATUS_t** for details.

Description:

API to stop created software timer instance.

Note : This API must be called after software timer is created using SYSTIMER_CreateTimer API with generated ID and enable XMC_ASSERT for better understanding of API behavioral in run time.

Example Usage:

```
#include <DAVE.h>
#define ONESEC 1000000U
void LED_Toggle_EverySec(void)
{
    // Add user code here
}
int main(void)
{
    uint32_t TimerId;
    SYSTIMER_STATUS_t status;
    // ... Initializes APPs configuration ...
    DAVE_Init(); // SYSTIMER APP Initialized during DAVE Initialization
    // Create Software timer
    TimerId =
    (uint32_t)SYSTIMER_CreateTimer(ONESEC,SYSTIMER_MODE_PERIODIC,
    (void*)LED_Toggle_EverySec,NULL);
    if (TimerId != 0U)
```

```

{
//timer is created successfully, now start/run software timer
status = SYSTIMER_StartTimer(TimerId);
if (status == SYSTIMER_STATUS_SUCCESS)
{
// Software timer is running
// Add user code here
//stop the timer
status = SYSTIMER_StopTimer(TimerId);
if (status == SYSTIMER_STATUS_SUCCESS)
{
//Software timer has stopped
}
else
{
// Error during software timer stop operation
}
}
else
{
// Error during software timer start operation
}
}
else
{
// timer ID Can not be zero
}
// ... infinite loop ...
while (1)
{
}
return (1);
}

```

Definition at line **530** of file **SYSTIMER.c**.

References [g_timer_tbl](#), [SYSTIMER_STATE_NOT_INITIALIZED](#), [SYSTIMER_STATE_RUNNING](#), [SYSTIMER_STATE_STOPPED](#), [SYSTIMER_STATUS_FAILURE](#), and [SYSTIMER_STATUS_SUCCESS](#).

Referenced by [SYSTIMER_DeleteTimer\(\)](#), [SYSTIMER_RestartTimer\(\)](#), and [SYSTIMER_StopTimerFromISR\(\)](#).

[SYSTIMER_STATUS_t](#) SYSTIMER_StopTimerFromISR (uint32_t **id**

A version of [SYSTIMER_StopTimer\(\)](#) that can be called from an ISR.

Parameters

id timer ID obtained from [SYSTIMER_CreateTimer](#). Range : 1 to 16

Returns

[SYSTIMER_STATUS_t](#) APP status. Refer [SYSTIMER_STATUS_t](#) for details.

Definition at line **558** of file [SYSTIMER.c](#).

References [SYSTIMER_StopTimer\(\)](#).

SYSTIMER

[Home](#)

Enumerations

enum

SYSTIMER_STATUS {
SYSTIMER_STATUS_SUCCESS
SYSTIMER_STATUS_FAILURE }
This enumeration indicates status
SYSTIMER. [More...](#)

enum

SYSTIMER_STATE {
SYSTIMER_STATE_NOT_INITIAL
= 0U, **SYSTIMER_STATE_RUNNI**
SYSTIMER_STATE_STOPPED }
This enumeration defines possible
timer state. [More...](#)

enum

SYSTIMER_MODE {
SYSTIMER_MODE_ONE_SHOT :
SYSTIMER_MODE_PERIODIC }
Enumeration values which describ
timer types. [More...](#)

typedef enum **SYSTIMER_STATUS**

SYSTIMER_STATUS_t
This enumeration indicates status
SYSTIMER.

typedef enum **SYSTIMER_STATE**

SYSTIMER_STATE_t
This enumeration defines possible
timer state.

typedef enum **SYSTIMER_MODE**

SYSTIMER_MODE_t

Enumeration values which describe timer types.

Detailed Description

Enumeration Type Documentation

enum **SYSTIMER_MODE**

Enumeration values which describes timer types.

Enumerator	
<i>SYSTIMER_MODE_ONE_SHOT</i>	timer type is one shot
<i>SYSTIMER_MODE_PERIODIC</i>	timer type is periodic

Definition at line **124** of file **SYSTIMER.h**.

enum **SYSTIMER_STATE**

This enumeration defines possible timer state.

Enumerator	
<i>SYSTIMER_STATE_NOT_INITIALIZED</i>	The timer is in uninitialized state
<i>SYSTIMER_STATE_RUNNING</i>	The timer is in running state
<i>SYSTIMER_STATE_STOPPED</i>	The timer is in stop state

Definition at line **114** of file **SYSTIMER.h**.

enum SYSTIMER_STATUS

This enumeration indicates status of **SYSTIMER**.

Enumerator	
<i>SYSTIMER_STATUS_SUCCESS</i>	Status Success if initialization is successful
<i>SYSTIMER_STATUS_FAILURE</i>	Status Failure if initialization is failed

Definition at line **105** of file **SYSTIMER.h**.



SYSTIMER

[Home](#)

Data structures

typedef void(* **SYSTIMER_CALLBACK_t**)(void *args)
timer callback function pointer

typedef struct **SYSTIMER** **SYSTIMER_t**
This structure contains pointer which is used to hold CPU instance handle and variables for priority group.

Detailed Description

--