

# MTD2A\_timer

**MTD2A: Model Train Detection And Action – arduino library** <https://github.com/MTD2A/MTD2A>

Jørgen Bo Madsen / V1.1 / 08-10-2025

MTD2A\_timer is an easy-to-use, advanced and functional C++ class for non-blocking time control. MTD2A supports parallel processing and asynchronous execution.

The class is among a number of logical building blocks that solve different functions.

## Common to all building blocks are:

- They support a wide range of input sensors and output devices
- Are simple to use to build complex solutions with few commands
- They operate non-blocking, process-oriented and state-driven
- Offers extensive control and troubleshooting information
- Thoroughly documented with examples

## Table of contents

MTD2A_binary_input .....	1
Feature Description .....	1
Process phases.....	2
Set and Get Features Overview .....	3
Print_conf();.....	3
Precise timing .....	4

## Feature Description

MTD2A\_binary\_input process consists of 3 functions:

1. `MTD2A_timer object_name ("object_name" , CountdownMS );`
2. `object_name.timer ( { RESET_TIMER | START_TIMER | PAUSE_TIMER | STOP_TIMER }, countdownMS) ;`  
Called in `void setup () ;`
3. `MTD2A_loop_execute () ;` Called as the last instruction in `void loop () ;`

The first argument uses the default value and the function can be called with none and up to 2 arguments.

```
MTD2A_timer object_name ();  
MTD2A_timer object_name ("Object_name");  
MTD2A_timer object_name ("Object_name", countdownMS);  
Default: ("Object_name", 0 );
```

## Eksempel

```
// Event if-condition controlled. Recommended for simple solutions
// Read infrared sensor. Short detection: One LED blink.
// continuously detection: continuously LED blink
// Suitable for Unpredictable process flows such as object detection with a sensor
if (IR_sensor.get_processState() == ACTIVE) {
    Serial.println ("Object detected");
    if (timer_GL1.get_processState() == COMPLETE) {
        timer_GL1.timer (START_TIMER, 1000); // 1 second
        green_LED_1.activate (500);          // 0,5 second
    }
}
```

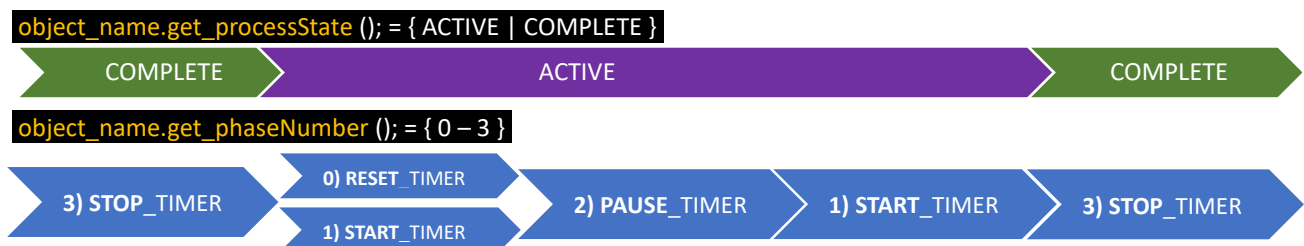
More examples and youtube demo video:

<https://github.com/MTD2A/MTD2A/tree/main/examples>

DEMO video: [https://youtu.be/UU4k4\\_8GWfM](https://youtu.be/UU4k4_8GWfM)

## Process phases

Depending on the current configuration, the process is carried out in between 2 and multiple phases with pauses.



3. The initial phase when the program starts **STOP\_TIMER**.
0. When function **object\_name.timer (RESET\_TIMER);** Is called. Can occur at any time.
1. When function **object\_name.timer (START\_TIMER);** Is called. May occur several times after each pause.
2. When function **object\_name.timer (PAUSE\_TIMER);** Is called. May occur several times after start and reset.
3. When function **object\_name.timer (STOP\_TIMER);** Is called or when the time has expired. **countDownMS** = 0.

Global Number Constants:

**RESET\_TIMER**, **START\_TIMER**, **PAUSE\_TIMER** og **STOP\_TIMER**

The immediate phase shift can be identified by function: **object\_name.get\_phaseChange (); = { true | false }**

## Proces status

When transitioning to **START\_TIMER** or **RESET\_TIMER**, ProcessState switches to **ACTIVE**.

When transitioning to **STOP\_TIMER**, the processState switches to **COMPLETE**.

ProcessState remains **ACTIVE** during **PAUSE\_TIMER**.

## Timing

The time period can be set when the object is instantiated. Subsequently, new time periods can be defined with the functin: **object\_name.set\_countDownMS ( {0 - 4294967295} );** But only when processState is **COMPLETE**

Refer to the document MTD2A.PDF and the section "Cadence", "Synchronization" and "Execution speed".

## Set and Get Features Overview

Set functions	Comment
set_countDownMS ( {0- 4294967295} );	Set new count down time in milliseconds
set_debugPrint ( { <b>ENABLE</b>   DISABLE} );	Activate print phase number and text.
set_errorPrint ( { <b>ENABLE</b>   DISABLE} );	Activate error messages.

Get functions	Comment
get_processtState (); return <b>bool</b> {ACTIVE   COMPLETE}	Process state
get_phaseChange (); return <b>bool</b> {true   false}	Momentarily phase change (one loop time)
get_phaseNumber (); return <b>uint8_t</b> {0- 3}	RESET_TIMER = 0, START_TIMER =1, PAUSE_TIMER = 2, STOP_TIMER = 3
get_startTimeMS (); return <b>uint32_t</b> milliseconds	Last START_TIMER
get_stopTimeMS (); return <b>uint32_t</b> milliseconds	STOP_TIMER or countDownMS is zero
get_pauseTimeMS (); return <b>uint32_t</b> milliseconds	Acuumulated pause time (sum of all pause periods). Zero if no pause initiated
get_remainTimeMS (); return <b>uint32_t</b> milliseconds	Remining time since first start
get_elapsedTimeMS (); return <b>uint32_t</b> milliseconds	elapsed time since first start
get_reset_error (); return <b>uint8_t</b> {0-255}	Get error/warning number and reset number: Error [1 – 127] warning [128 – 255]

Operator overloading	Function
object_name_1 == object_name_2	<b>bool</b> processState_1 == processState_2
object_name_1 != object_name_2	<b>bool</b> processState_1 != processState_2
object_name_1 > object_name_2	<b>bool</b> processState_1 = ACTIVE & processState_2 = COMPLETE
object_name_1 < object_name_2	<b>bool</b> processState_1 = COMPLETE & processState_2 = ACTIVE
object_name_1 >> object_name_2	<b>bool</b> stopTimeMS_1 > stopTimeMS_2
object_name_1 << object_name_2	<b>bool</b> stopTimeMS_1 < stopTimeMS_2

### Print\_conf();

**object\_name.print\_conf ();**

```
MTD2A_timer:
-----
objectName      : Timer
processState    : COMPLETE
phaseText       : [3] Stop timer
debugPrint      : ENABLE
globalDebugPr   : DISABLE
errorPrint      : DISABLE
globalErrorPr   : ENABLE
errorNumber     : 0 OK
countDownMS     : 20000
startTimeMS     : 4015
stopTimeMS      : 19011
remainTimeMS    : 9896
elapsedTimeMS   : 10104
pauseTimeMS     : 5999
pauseBeginMS    : 13012
pauseEndMS      : 16012
```

If there are multiple breaks during the time period, it shows the total pause time **pauseTimeMS**, but **pauseBeginMS** and **pauseEndMS** only appear for the last pause.

## Precise timing

If greater time accuracy is desired, it is recommended to use the Arduino ESP32 and the cadence `DELAY_1MS`

```
MTD2A_timer:
-----
objectName      : Timer
processState    : COMPLETE
phaseText       : [3] Stop timer
debugPrint      : ENABLE
globalDebugPr   : DISABLE
errorPrint      : DISABLE
globalErrorPr   : ENABLE
errorNumber     : 0 OK
countDownMS     : 8000
remainTimeMS    : 6000
elapsedTimeMS   : 2000
startTimeMS     : 4576
stopTimeMS      : 9576
pauseTimeMS     : 3000
pauseBeginMS    : 6576
pauseEndMS      : 9576
```