TimerClass

1.0

Generated by Doxygen 1.8.17

Thu Mar 19 2020 18:59:10

1 Modules Index

1.1 Modules List

Here is a list of all modules with brief descriptions:

timeclass
The TimeClass module contains the TTime class used by the TTimer class for practical timing **2**

timerclass
Module containing the TTimer class for practical timing **10**

2 Data Type Index

2.1 Data Types List

Here are the data types with brief descriptions:

3 File Index

3.1 File List

Here is a list of all files with brief descriptions:

4 Module Documentation

4.1 timeclass Module Reference

The **TimeClass module** contains the TTime class used by the TTimer class for practical timing.

Data Types

- module ttime

  The TTime class contains all time functionality with regard to a single time stamp.

Functions/Subroutines

- pure type(ttime) function constructor ()

  Constructor for the TTime class.
- subroutine settime (this)

  Function to set the TTime instance to the current time, with millisecond resolution.
- pure subroutine calculatejdn (this)

  Calculating the Julian Day number based on the Gregorian date set in the TTime object.
- pure subroutine setjdn (this, JDN, IO)

  Set the Julian Day Number of a TTime instance manually.
- pure integer(kind=4) function getjuliandaynumber (this)

  Function returning the Julian Day Number as a 4-byte integer.
- pure subroutine setgregoriandatefromjdn (this, IO)

Subroutine which transforms the set Julian Day Number into a Gregorian Calender date.

- pure subroutine copy (this, from)

  Function to copy one TTime instance to the current one via the "=" assignment. This is the function performing the actual operator overloading.
- pure type(ttime) function add (this, that)

  Function to add two TTime instance via the "+" operator. This is the function performing the actual operator overloading.
- pure type(ttime) function subtract (this, that)

  Function to subtract two TTime instance via the "-" operator. This is the function performing the actual operator overloading.
- pure logical function isleapyear (this)

  Function returning true/false if the year of the TTime instance is a leap year.
- pure character(len=255) function gettimestring (this, fmt)

  Returns a string with the time as a string.
- pure real(dp) function gettimeseconds (this)

  Function returning the time in (fractional) seconds (double precision).
- subroutine destructor (this)

  Destructor of the TTime object instance.

4.1.1 Detailed Description

The **TimeClass module** contains the TTime class used by the TTimer class for practical timing.

Author

Dr. Dr. Danny E. P. Vanpoucke

Version

2.0-3 (upgrades deprecated timing module)

Date

19-03-2020

Copyright

`https://dannyvanpoucke.be`

Warning

Internally, Julian Day Numbers are used to compare dates. As a result, *negative* dates are not accepted. If such dates are created (*e.g.*, due to a subtraction), then the date is set to zero.

This module makes use of:

- nothing; this module is fully independent

4.1.2 Function/Subroutine Documentation

4.1.2.1 add()

```
pure type(ttime) function timeclass::add (
            class(ttime), intent(in) this,
            class(ttime), intent(in) that )  [private]
```

Function to add two TTime instance via the "+" operator. This is the function performing the actual operator overloading.

Adding two full dates is maybe a bit strange to do. In our case, we don't just add the days and add the months, but we add the days of the year (via their Julian Day Numbers) and transform these back to months and days. (why keep life simple if we can make it complicated?)

usage:

```
total = this + that
```

This line also calls the assignment operator.

Parameters

in	*this*	The TTime instance before the "+" operator.
in	*that*	The TTime instance after the "+" operator.

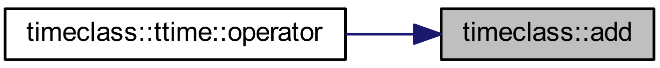
Returns

Total The TTime instance representing the sum.

Definition at line 264 of file TimeClass.f03.

Referenced by timeclass::ttime::operator().

Here is the caller graph for this function:



4.1.2.2 calculatejdn()

```
pure subroutine timeclass::calculatejdn (
            class(ttime), intent(inout) this )  [private]
```

Calculating the Julian Day number based on the Gregorian date set in the TTime object.

In practice the Gregorian calender date set in the TTime instance is transformed into a Julian Day Number, and stored in the instance as ttime::jdn.

Parameters

`in,out`	*this*	The TTime instance being called.

Definition at line 139 of file TimeClass.f03.

4.1.2.3 constructor() `pure type(ttime) function timeclass::constructor [private]`

Constructor for the TTime class.

Note

This constructor does not set the time. It just enters zero's.

usage:
```
Type(TTime) :: T
t = ttime()
```

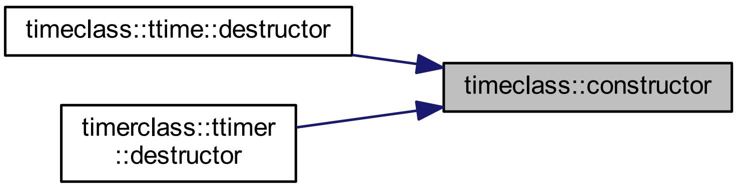
Returns

Time An instance of the TTime class.

Definition at line 98 of file TimeClass.f03.

Referenced by timeclass::ttime::destructor(), and timerclass::ttimer::destructor().

Here is the caller graph for this function:



4.1.2.4 copy()
```
pure subroutine timeclass::copy (
        class(ttime), intent(inout) this,
        class(ttime), intent(in) from )  [private]
```

Function to copy one TTime instance to the current one via the "=" assignment. This is the function performing the actual operator overloading.

usage:
```
tnew = told
```

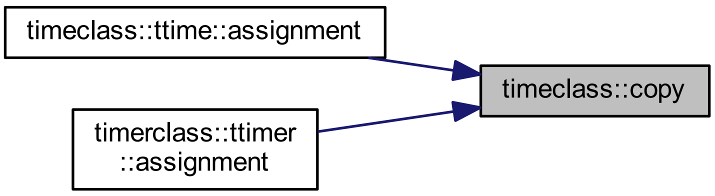
Parameters

`in,out`	*this*	The TTime instance before the "=" assignment.
`in`	*from*	The TTime instance after the "=" assignment.

Definition at line 233 of file TimeClass.f03.

Referenced by timeclass::ttime::assignment(), and timerclass::ttimer::assignment().

Here is the caller graph for this function:



4.1.2.5 destructor() `subroutine timeclass::destructor ( type(ttime) this ) [private]`

Destructor of the TTime object instance.

This subroutine is automatically called upon finalization of the instance.

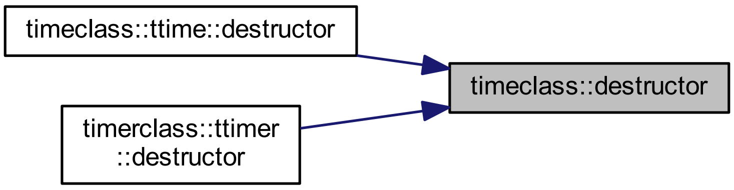
Parameters

`in,out`	*this*	The instance of the TTime class in need of destruction.

Definition at line 419 of file TimeClass.f03.

Referenced by timeclass::ttime::destructor(), and timerclass::ttimer::destructor().

Here is the caller graph for this function:



4.1.2.6 getjuliandaynumber() `pure integer(kind=4) function timeclass::getjuliandaynumber ( class(ttime), intent(in) this ) [private]`

Function returning the Julian Day Number as a 4-byte integer.

Use integer(kind=4).

Parameters

`in,out`	*this*	The TTime instance being called.

Returns

JDN The integer Julian Day Number.

Definition at line 184 of file TimeClass.f03.

4.1.2.7 gettimeseconds() `pure real(dp) function timeclass::gettimeseconds ( class(ttime), intent(in) this ) [private]`

Function returning the time in (fractional) seconds (double precision).

Parameters

`in`	*this*	The TTime instance.

Returns

sec total number of seconds representing the "time" as a double precision value.

Definition at line 402 of file TimeClass.f03.

4.1.2.8 gettimestring() `pure character(len=255) function timeclass::gettimestring ( class(ttime), intent(in) this, character(len=*), intent(in), optional fmt ) [private]`

Returns a string with the time as a string.

Format options for fmt:

- full: dd/mm/yyyy hh:mm:ss.mmm
- date: dd/mm/yyyy
- time: hh:mm:ss.mmm
- days: Gives the total time in fractional days (best used for time-differences). Uses the Julian Day Number.
- hours: Same as days, but transformed to hours.
- seconds: Same as days, but transformed to seconds.

Parameters

in	*this*	The TTime instance transform into a string.
in	*fmt*	String representing the possible formatting. [**OPTIONAL**, **DEFAULT** = full]

Returns

TS String with formatted time.

Definition at line 356 of file TimeClass.f03.

4.1.2.9 isleapyear() `pure logical function timeclass::isleapyear ( class(ttime), intent(in) this ) [private]`

Function returning true/false if the year of the TTime instance is a leap year.

A leap year is a multiple of 4, but not 100, unless 400

Parameters

in	*this*	The TTime instance to check the leap-year.

Returns

Leap Boolean indicating if the year is a leap-year.

Definition at line 327 of file TimeClass.f03.

4.1.2.10 setgregoriandatefromjdn()

```
pure subroutine timeclass::setgregoriandatefromjdn (
        class(ttime), intent(inout) this,
        integer, intent(out), optional IO )  [private]
```

Subroutine which transforms the set Julian Day Number into a Gregorian Calender date.

Note

The routine is only valid for a JDN>=0.

Parameters

in,out	*this*	The TTime instance being called.
out	*IO*	Returns 0 on success, and -1 for failure. [**OPTIONAL**]

Definition at line 199 of file TimeClass.f03.

4.1.2.11 setjdn()

```
pure subroutine timeclass::setjdn (
        class(ttime), intent(inout) this,
        integer(kind=4), intent(in) JDN,
        integer, intent(out), optional IO )  [private]
```

Set the Julian Day Number of a TTime instance manually.

Note

The Julian Day Number should be >=0. For negative values it is set to 0, and an error value is set to IO

Parameters

in,out	*this*	The TTime instance being called.
in	*JDN*	A positive integer(kind=4) value representing a valid Julian Day Number.
out	*IO*	Integer value returning 0 upon success, and a negative value(=JDN) in case of failure.

Definition at line 158 of file TimeClass.f03.

4.1.2.12 settime()

```
subroutine timeclass::settime (
        class(ttime), intent(inout) this )  [private]
```

Function to set the TTime instance to the current time, with millisecond resolution.

Note

As this subroutine uses the date_and_time intrinsic it is an *impure* subroutine.

Parameters

`in,out`	*this*	The TTime instance being called.

Definition at line 116 of file TimeClass.f03.

4.1.2.13 subtract() `pure type(ttime) function timeclass::subtract ( class(ttime), intent(in) *this*, class(ttime), intent(in) *that* ) [private]`

Function to subtract two TTime instance via the "-" operator. This is the function performing the actual operator overloading.

usage:

```
total = this - that
```

This line also calls the assignment operator.

Note

The result should remain a positive number.

Parameters

`in`	*this*	The TTime instance before the "-" operator.
`in`	*that*	The TTime instance after the "-" operator.

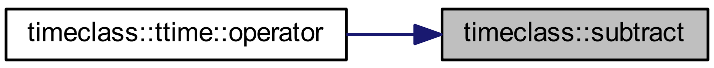
Returns

Total The TTime instance representing the difference.

Definition at line 300 of file TimeClass.f03.

Referenced by timeclass::ttime::operator().

Here is the caller graph for this function:



4.2 timerclass Module Reference

Module containing the TTimer class for practical timing.

Data Types

- module ttimer

  A class for keeping track of time in calculations, by creating a list of timestamps.

Functions/Subroutines

- pure type(ttimer) function constructor ()

  Constructor for the TTimer instances.
- integer function start (this)

  Start the TTimer instance (clean start, everything is reset). If the timer was already running it is reset first.
- integer function resume (this)

  Restart the TTimer instance after a pause. If the timer is not paused, nothing will happen and TS=-1 is returned.
- integer function addtimeflag (this)

  Add an additional timestamp without changing the status (running/paused) of the TTimer instance.
- integer function interrupt (this, IO)

  Puts the TTimer instance on hold.
- integer function stoptimer (this, IO)

  Stops the TTimer instance (terminal fashion...no restart possible).
- pure subroutine reset (this)

  Start the TTimer instance. If the timer was already running it is reset first.
- integer function addtimestamp (this)

  Add a timestamp to a running TTimer instance, returning the index of the timestamp.
- pure real(dp) function getelapsedtime_total (this, INCL_PAUSE)

  Returns the number of seconds which elapsed between the start and stop timestamps.
- pure real(dp) function getelapsedtime_steps (this, Tstart, Tend, INCL_PAUSE)

  Returns the number of seconds which elapsed between two timestamps. This is always a positive value.
- pure character(len=50) function getelapsedtimestring (this, TSTART, TSTOP, INCL_PAUSE, FMT)

  Returns a string representing the elapsed time. +.
- subroutine printelapsedtimereport (this, message, Tstart, Tstop, UN, INCL_PAUSE)

  Print small timings report to unit UN.
- pure subroutine copy (this, from)

  Function to copy one TTimer instance to the current one via the "=" assignment.
- subroutine destructor (this)

  Destructor of the TTimer class. Cleans up the instance upon finalization.

4.2.1 Detailed Description

Module containing the TTimer class for practical timing.

Author

Dr. Dr. Danny E. P. Vanpoucke

Version

2.0-3 (upgrades deprecated timing module)

Date

19-03-2020

Copyright

`https://dannyvanpoucke.be`

This module makes use of:

- TimeClass

4.2.2 Function/Subroutine Documentation

4.2.2.1 addtimeflag() `integer function timerclass::addtimeflag ( class(ttimer), intent(inout) this ) [private]`

Add an additional timestamp without changing the status (running/paused) of the TTimer instance.

If the timer is stopped nothing will happen, and -1 is returned.

Parameters

`in,out`	*this*	The TTimer instance.

Returns

TS The index of the timestamp.

Definition at line 165 of file TimerClass.f03.

4.2.2.2 addtimestamp() `integer function timerclass::addtimestamp ( class(ttimer), intent(inout) this ) [private]`

Add a timestamp to a running TTimer instance, returning the index of the timestamp.

In case the timer is not running, nothing is done and -1 is returned.

Parameters

`in,out`	*this*	The TTimer instance.

Returns

TS The index of the timestamp.

Definition at line 279 of file TimerClass.f03.

4.2.2.3 constructor() `pure type(ttimer) function timerclass::constructor [private]`

Constructor for the TTimer instances.

Usage:

```
Type(TTimer) :: T
t=ttimer()
```

Returns

Returns a TTimer object

Definition at line 104 of file TimerClass.f03.

4.2.2.4 copy()

```
pure subroutine timerclass::copy (
        class(ttimer), intent(inout) this,
        class(ttimer), intent(in) from )  [private]
```

Function to copy one TTimer instance to the current one via the "=" assignment.

Parameters

in,out	*this*	The TTimer instance before the "=" assignment.
in	*from*	The TTimer instance after the "=" assignment.

Definition at line 504 of file TimerClass.f03.

4.2.2.5 destructor()

```
subroutine timerclass::destructor (
        type(ttimer) this )  [private]
```

Destructor of the TTimer class. Cleans up the instance upon finalization.

Parameters

in,out	*this*	The TTimer instance being destroyed automatically.

Definition at line 529 of file TimerClass.f03.

4.2.2.6 getelapsedtime_steps()

```
pure real(dp) function timerclass::getelapsedtime_steps (
        class(ttimer), intent(in) this,
        integer, intent(in) Tstart,
        integer, intent(in) Tend,
        logical, intent(in), optional INCL_PAUSE )  [private]
```

Returns the number of seconds which elapsed between two timestamps. This is always a positive value.

NOTE:

- If the timestamps are out of range, then -1 is returned.
- If the start comes after end, then they are exchanged such that a positive time is obtained.

Parameters

in	*this*	The TTimer instance.
in	*Tstart,Tend*	The indices of the start and end time.
in	*INCL_PAUSE*	Logical indicating if the time during which the timer was paused should be included as well. [**OPTIONAL**, **DEFAULT** = .false.]

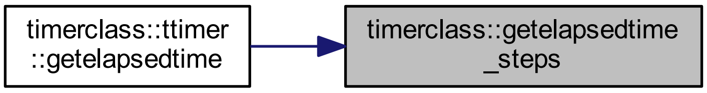
Returns

Seconds The (fractional) number of seconds elapsed, as double precision real.

Definition at line 348 of file TimerClass.f03.

Referenced by timerclass::ttimer::getelapsedtime().

Here is the caller graph for this function:


timerclass::ttimer ::getelapsedtime
timerclass::getelapsedtime _steps

4.2.2.7 getelapsedtime_total()

```
pure real(dp) function timerclass::getelapsedtime_total (
        class(ttimer), intent(in) this,
        logical, intent(in), optional INCL_PAUSE )  [private]
```

Returns the number of seconds which elapsed between the start and stop timestamps.

Parameters

in	*this*	The TTimer instance.
in	*INCL_PAUSE*	Logical indicating if the time during which the timer was paused should be included as well. [**OPTIONAL**, **DEFAULT** = .false.]

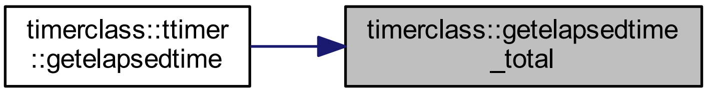
Returns

Seconds The (fractional) number of seconds elapsed, as double precision real.

Definition at line 320 of file TimerClass.f03.

Referenced by timerclass::ttimer::getelapsedtime().

Here is the caller graph for this function:


timerclass::ttimer ::getelapsedtime
timerclass::getelapsedtime _total

4.2.2.8 getelapsedtimestring()

```
pure character(len=50) function timerclass::getelapsedtimestring
(
            class(ttimer), intent(in) this,
            integer, intent(in), optional TSTART,
            integer, intent(in), optional TSTOP,
            logical, intent(in), optional INCL_PAUSE,
            character(len=*), intent(in), optional FMT )  [private]
```

Returns a string representing the elapsed time. +.

Parameters

in	*this*	The TTimer instance.
in	*TSTART,TSTOP*	Indices of the start and end-times. [**OPTIONAL**, **DEFAULT:** TSTART=1, TSTOP=index of last timestamp]
in	*INCL_PAUSE*	Logical indicating if the time during which the timer was paused should be included as well. [**OPTIONAL**, **DEFAULT** = .false.]
in	*FMT*	string indicating the format for the time-string. • sec : xxx.xxx secs • hms : xxxx h xx min xx.xxx secs • dhms: xx days xx h xx min xx.xxx secs • hour: xxx.xxx hours • day : xxx.xxx days [**OPTIONAL**, **DEFAULT=** sec]

Returns

str The string with the formatted time.

Definition at line 402 of file TimerClass.f03.

4.2.2.9 interrupt()

```
integer function timerclass::interrupt (
            class(ttimer), intent(inout) this,
            integer, intent(out), optional IO )  [private]
```

Puts the TTimer instance on hold.

If the timer was not running nothing will happen. The optional IO parameter will return an error code.
IO values:

- 0 : all is well
- -1 : The timer was not running, so nothing to pause.
- -2 : The timer was already stopped/paused, so nothing to pause.

In case of error, TS will be set to -1.

Parameters

in,out	*this*	The TTimer instance.
out	*IO*	Optional parameter giving the error-status. [**OPTIONAL** ; **DEFAULT=** 0]

Returns

TS The index of the final timestamp.

Definition at line 193 of file TimerClass.f03.

4.2.2.10 printelapsedtimereport()

```
subroutine timerclass::printelapsedtimereport (
        class(ttimer), intent(inout) this,
        character(len=*), intent(in) message,
        integer, intent(in) Tstart,
        integer, intent(in) Tstop,
        integer, intent(in) UN,
        logical, intent(in), optional INCL_PAUSE )  [private]
```

Print small timings report to unit UN.

Parameters

in	*this*	The TTimer instance.
in	*message*	String containing a message to include.
in	*Tstart,Tstop*	Integer indexes of the start and stop times
in	*UN*	Integer indicating the unit to write the report to.
in	*INCL_PAUSE*	Logical indicating if the time during which the timer was paused should be included as well. [**OPTIONAL**, **DEFAULT** = .false.]

Definition at line 470 of file TimerClass.f03.

4.2.2.11 reset()

```
pure subroutine timerclass::reset (
        class(ttimer), intent(inout) this )  [private]
```

Start the TTimer instance. If the timer was already running it is reset first.

Parameters

in,out	*this*	The TTimer instance.

Definition at line 255 of file TimerClass.f03.

4.2.2.12 resume() `integer function timerclass::resume ( class(ttimer), intent(inout) this )  [private]`

Restart the TTimer instance after a pause. If the timer is not paused, nothing will happen and TS=-1 is returned.

Parameters

`in,out`	*this*	The TTimer instance.

Returns

TS The index of the first timestamp.

Definition at line 142 of file TimerClass.f03.

4.2.2.13 start() `integer function timerclass::start ( class(ttimer), intent(inout) this )  [private]`

Start the TTimer instance (clean start, everything is reset). If the timer was already running it is reset first.

Parameters

`in,out`	*this*	The TTimer instance.

Returns

TS The index of the first timestamp.

Definition at line 124 of file TimerClass.f03.

4.2.2.14 stoptimer() `integer function timerclass::stoptimer ( class(ttimer), intent(inout) this, integer, intent(out), optional IO )  [private]`

Stops the TTimer instance (terminal fashion...no restart possible).

If the timer was not running nothing will happen. The optional IO parameter will return an error code.
IO values:

- 0 : all is well
- -1 : The timer was not running, so nothing to stop.
- -2 : The timer was already stopped, so nothing to stop.

In case of error, TS will be set to -1.

Parameters

`in,out`	*this*	The TTimer instance.
`out`	*IO*	Optional parameter giving the error-status. [**OPTIONAL** ; **DEFAULT=** 0]

Returns

TS The index of the final timestamp.

Definition at line 229 of file TimerClass.f03.

5 Data Type Documentation

5.1 timeclass::ttime Module Reference

The TTime class contains all time functionality with regard to a single time stamp.

Public Member Functions

- procedure, pass, public settime

  Function to set the TTime instance to the current time, with millisecond resolution.
- procedure, pass, public getjuliandaynumber

  Function returning the Julian Day Number as a 4-byte integer.
- procedure, pass, public isleapyear

  Function returning true/false if the year of the TTime instance is a leap year.
- procedure, pass, public gettimestring

  Returns a string with the time as a string.
- procedure, pass, public gettimeseconds

  Function returning the time in (fractional) seconds (double precision).
- generic, public assignment => copy

  Function to copy one TTime instance to the current one via the "=" assignment. This is the function performing the actual operator overloading.
- generic, public operator => add

  Function to add two TTime instance via the "+" operator. This is the function performing the actual operator overloading.
- generic, public operator => subtract

  Function to subtract two TTime instance via the "-" operator. This is the function performing the actual operator overloading.

Protected Member Functions

- final destructor

  Destructor of the TTime object instance.

Private Member Functions

- procedure, pass(this) calculatejdn

    Calculating the Julian Day number based on the Gregorian date set in the TTime object.
- procedure, pass(this) setjdn

    Set the Julian Day Number of a TTime instance manually.
- procedure, pass(this) setgregoriandatefromjdn

    Subroutine which transforms the set Julian Day Number into a Gregorian Calender date.
- procedure, pass(this) copy

    Function to copy one TTime instance to the current one via the "=" assignment. This is the function performing the actual operator overloading.
- procedure, pass(this) add

    Function to add two TTime instance via the "+" operator. This is the function performing the actual operator overloading.
- procedure, pass(this) subtract

    Function to subtract two TTime instance via the "-" operator. This is the function performing the actual operator overloading.
- pure type(ttime) function constructor ()

    Constructor for the TTime class.

Private Attributes

- integer year

    The year.
- integer month

    The month (as integer).
- integer day

    Day of the month.
- integer(kind=4) jdn

    The Julian Day Number, as a long-int (4-byte)
- real daysecs

    Number of seconds of the day, with millisecond resolution.

5.1.1 Detailed Description

The TTime class contains all time functionality with regard to a single time stamp.

Definition at line 50 of file TimeClass.f03.

5.1.2 Member Function/Subroutine Documentation

5.1.2.1 add() `procedure, pass(this) timeclass::ttime::add [private]`

Function to add two TTime instance via the "+" operator. This is the function performing the actual operator overloading.

Adding two full dates is maybe a bit strange to do. In our case, we don't just add the days and add the months, but we add the days of the year (via their Julian Day Numbers) and transform these back to months and days. (why keep life simple if we can make it complicated?)

usage:
```
total = this + that
```

This line also calls the assignment operator.

Parameters

in	*this*	The TTime instance before the "+" operator.
in	*that*	The TTime instance after the "+" operator.

Returns

Total The TTime instance representing the sum.

Definition at line 65 of file TimeClass.f03.

5.1.2.2 assignment() `generic, public timeclass::ttime::assignment`

Function to copy one TTime instance to the current one via the "=" assignment. This is the function performing the actual operator overloading.

usage:
```
tnew = told
```

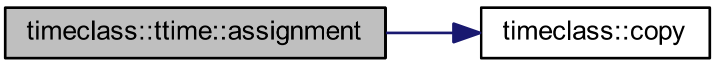
Parameters

in,out	*this*	The TTime instance before the "=" assignment.
in	*from*	The TTime instance after the "=" assignment.

Definition at line 70 of file TimeClass.f03.

References timeclass::copy().

Here is the call graph for this function:



5.1.2.3 calculatejdn() `procedure, pass(this) timeclass::ttime::calculatejdn [private]`

Calculating the Julian Day number based on the Gregorian date set in the TTime object.

In practice the Gregorian calender date set in the TTime instance is transformed into a Julian Day Number, and stored in the instance as ttime::jdn.

Parameters

in,out	*this*	The TTime instance being called.

Definition at line 60 of file TimeClass.f03.

5.1.2.4 constructor() `pure type(ttime) function timeclass::ttime::constructor [private]`

Constructor for the TTime class.

Note

This constructor does not set the time. It just enters zero's.

usage:
```
Type(TTime) :: T
t = ttime()
```

Returns

Time An instance of the TTime class.

Definition at line 98 of file TimeClass.f03.

5.1.2.5 copy() `procedure, pass(this) timeclass::ttime::copy [private]`

Function to copy one TTime instance to the current one via the "=" assignment. This is the function performing the actual operator overloading.

usage:
```
tnew = told
```

Parameters

in,out	*this*	The TTime instance before the "=" assignment.
in	*from*	The TTime instance after the "=" assignment.

Definition at line 64 of file TimeClass.f03.

5.1.2.6 destructor() `final timeclass::ttime::destructor [final], [protected]`

Destructor of the TTime object instance.

This subroutine is automatically called upon finalization of the instance.

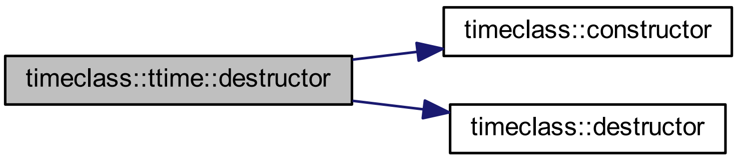
Parameters

`in,out`	*this*	The instance of the TTime class in need of destruction.

Definition at line 74 of file TimeClass.f03.

References timeclass::constructor(), and timeclass::destructor().

Here is the call graph for this function:



5.1.2.7 getjuliandaynumber() `procedure, pass, public timeclass::ttime::getjuliandaynumber`

Function returning the Julian Day Number as a 4-byte integer.

Use integer(kind=4).

Parameters

`in,out`	*this*	The TTime instance being called.

Returns

JDN The integer Julian Day Number.

Definition at line 63 of file TimeClass.f03.

5.1.2.8 gettimeseconds() `procedure, pass, public timeclass::ttime::gettimeseconds`

Function returning the time in (fractional) seconds (double precision).

Parameters

`in`	*this*	The TTime instance.

Returns

sec total number of seconds representing the "time" as a double precision value.

Definition at line 69 of file TimeClass.f03.

5.1.2.9 gettimestring() `procedure, pass, public timeclass::ttime::gettimestring`

Returns a string with the time as a string.

Format options for fmt:

- full: dd/mm/yyyy hh:mm:ss.mmm
- date: dd/mm/yyyy
- time: hh:mm:ss.mmm
- days: Gives the total time in fractional days (best used for time-differences). Uses the Julian Day Number.
- hours: Same as days, but transformed to hours.
- seconds: Same as days, but transformed to seconds.

Parameters

`in`	*this*	The TTime instance transform into a string.
`in`	*fmt*	String representing the possible formatting. [**OPTIONAL**, **DEFAULT** = full]

Returns

TS String with formatted time.

Definition at line 68 of file TimeClass.f03.

5.1.2.10 isleapyear() `procedure, pass, public timeclass::ttime::isleapyear`

Function returning true/false if the year of the TTime instance is a leap year.

A leap year is a multiple of 4, but not 100, unless 400

Parameters

`in`	*this*	The TTime instance to check the leap-year.

Returns

Leap Boolean indicating if the year is a leap-year.

Definition at line 67 of file TimeClass.f03.

5.1.2.11 operator() `[1/2]` `generic, public timeclass::ttime::operator`

Function to add two TTime instance via the "+" operator. This is the function performing the actual operator overloading.

Adding two full dates is maybe a bit strange to do. In our case, we don't just add the days and add the months, but we add the days of the year (via their Julian Day Numbers) and transform these back to months and days. (why keep life simple if we can make it complicated?)

usage:
```
total = this + that
```

This line also calls the assignment operator.

Parameters

`in`	*this*	The TTime instance before the "+" operator.
`in`	*that*	The TTime instance after the "+" operator.

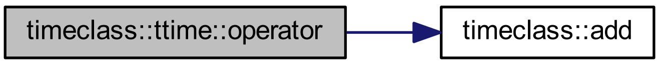
Returns

Total The TTime instance representing the sum.

Definition at line 71 of file TimeClass.f03.

References timeclass::add().

Here is the call graph for this function:



5.1.2.12 operator() `[2/2]` `generic, public timeclass::ttime::operator`

Function to subtract two TTime instance via the "-" operator. This is the function performing the actual operator overloading.

usage:
```
total = this - that
```

This line also calls the assignment operator.

Note

The result should remain a positive number.

Parameters

`in`	*this*	The TTime instance before the "-" operator.
`in`	*that*	The TTime instance after the "-" operator.

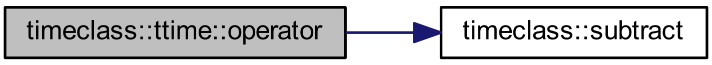
Returns

Total The TTime instance representing the difference.

Definition at line 72 of file TimeClass.f03.

References timeclass::subtract().

Here is the call graph for this function:



5.1.2.13 setgregoriandatefromjdn() `procedure, pass(this) timeclass::ttime::setgregoriandatefromjdn [private]`

Subroutine which transforms the set Julian Day Number into a Gregorian Calender date.

Note

The routine is only valid for a JDN>=0.

Parameters

`in,out`	*this*	The TTime instance being called.
`out`	*IO*	Returns 0 on success, and -1 for failure. [**OPTIONAL**]

Definition at line 62 of file TimeClass.f03.

5.1.2.14 setjdn() `procedure, pass(this) timeclass::ttime::setjdn [private]`

Set the Julian Day Number of a TTime instance manually.

Note

The Julian Day Number should be >=0. For negative values it is set to 0, and an error value is set to IO

Parameters

`in,out`	*this*	The TTime instance being called.
`in`	*JDN*	A positive integer(kind=4) value representing a valid Julian Day Number.
`out`	*IO*	Integer value returning 0 upon success, and a negative value(=JDN) in case of failure.

Definition at line 61 of file TimeClass.f03.

5.1.2.15 settime() `procedure, pass, public timeclass::ttime::settime`

Function to set the TTime instance to the current time, with millisecond resolution.

Note

As this subroutine uses the date_and_time intrinsic it is an *impure* subroutine.

Parameters

`in,out`	*this*	The TTime instance being called.

Definition at line 59 of file TimeClass.f03.

5.1.2.16 subtract() `procedure, pass(this) timeclass::ttime::subtract [private]`

Function to subtract two TTime instance via the "-" operator. This is the function performing the actual operator overloading.

usage:
```
total = this - that
```

This line also calls the assignment operator.

Note

The result should remain a positive number.

Parameters

`in`	*this*	The TTime instance before the "-" operator.
`in`	*that*	The TTime instance after the "-" operator.

Returns

Total The TTime instance representing the difference.

Definition at line 66 of file TimeClass.f03.

5.1.3 Member Data Documentation

5.1.3.1 day `integer timeclass::ttime::day [private]`

Day of the month.

Definition at line 54 of file TimeClass.f03.

5.1.3.2 daysecs `real timeclass::ttime::daysecs [private]`

Number of seconds of the day, with millisecond resolution.

Definition at line 56 of file TimeClass.f03.

5.1.3.3 jdn `integer(kind=4) timeclass::ttime::jdn [private]`

The Julian Day Number, as a long-int (4-byte)

Definition at line 55 of file TimeClass.f03.

5.1.3.4 month `integer timeclass::ttime::month [private]`

The month (as integer).

Definition at line 53 of file TimeClass.f03.

5.1.3.5 year `integer timeclass::ttime::year [private]`

The year.

Definition at line 52 of file TimeClass.f03.

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

- TimeClass.f03

5.2 timerclass::ttimer Module Reference

A class for keeping track of time in calculations, by creating a list of timestamps.

Public Member Functions

- procedure, pass, public start

    Start the TTimer instance (clean start, everything is reset). If the timer was already running it is reset first.
- procedure, pass, public interrupt

    Puts the TTimer instance on hold.
- procedure, pass, public resume

    Restart the TTimer instance after a pause. If the timer is not paused, nothing will happen and TS=-1 is returned.
- procedure, pass, public addtimeflag

    Add an additional timestamp without changing the status (running/paused) of the TTimer instance.
- procedure, pass, public stoptimer

    Stops the TTimer instance (terminal fashion...no restart possible).
- procedure, pass, public reset

    Start the TTimer instance. If the timer was already running it is reset first.
- procedure, pass, public printelapsedtimereport

    Print small timings report to unit UN.
- procedure, pass, public getelapsedtimestring

    Returns a string representing the elapsed time. +.
- generic, public assignment => copy

    Function to copy one TTimer instance to the current one via the "=" assignment.
- generic, public getelapsedtime => getelapsedtime_total, getelapsedtime_steps

    Generic interface to the GetElapsedTime_total and GetElapsedTime_steps methods of the TTimer class.

Protected Member Functions

- final destructor

    Destructor of the TTimer class. Cleans up the instance upon finalization.

Private Member Functions

- procedure, pass(this) getelapsedtime_total

    Returns the number of seconds which elapsed between the start and stop timestamps.
- procedure, pass(this) getelapsedtime_steps

    Returns the number of seconds which elapsed between two timestamps. This is always a positive value.
- procedure, pass(this) copy

    Function to copy one TTimer instance to the current one via the "=" assignment.
- procedure, pass(this) addtimestamp

    Add a timestamp to a running TTimer instance, returning the index of the timestamp.
- pure type(ttimer) function constructor ()

    Constructor for the TTimer instances.

Private Attributes

- integer ntimes

    The number of timestamps stored.
- integer maxtimes

    The size of the timestamp list.
- type(ttime), dimension(:), allocatable timestamps

    Allocatable list of timestamps.
- logical, dimension(:), allocatable timedinterval

    logical indicating if the timer was running during an interval between two timestamps or not
- logical running

    True if the timer is running.
- logical paused

    True if the timer is paused.
- logical stopped

    True if the timer is stopped (final end)

5.2.1 Detailed Description

A class for keeping track of time in calculations, by creating a list of timestamps.

Definition at line 50 of file TimerClass.f03.

5.2.2 Member Function/Subroutine Documentation

5.2.2.1 addtimeflag() `procedure, pass, public timerclass::ttimer::addtimeflag`

Add an additional timestamp without changing the status (running/paused) of the TTimer instance.

If the timer is stopped nothing will happen, and -1 is returned.

Parameters

`in,out`	*this*	The TTimer instance.

Returns

TS The index of the timestamp.

Definition at line 66 of file TimerClass.f03.

5.2.2.2 addtimestamp() `procedure, pass(this) timerclass::ttimer::addtimestamp [private]`

Add a timestamp to a running TTimer instance, returning the index of the timestamp.

In case the timer is not running, nothing is done and -1 is returned.

Parameters

`in,out`	*this*	The TTimer instance.

Returns

TS The index of the timestamp.

Definition at line 74 of file TimerClass.f03.

5.2.2.3 assignment() `generic, public timerclass::ttimer::assignment`

Function to copy one TTimer instance to the current one via the "=" assignment.

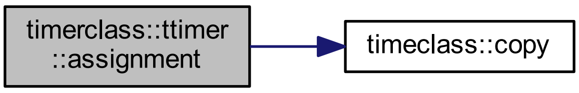
Parameters

`in,out`	*this*	The TTimer instance before the "=" assignment.
`in`	*from*	The TTimer instance after the "=" assignment.

Definition at line 75 of file TimerClass.f03.

References timeclass::copy().

Here is the call graph for this function:



5.2.2.4 constructor() `pure type(ttimer) function timerclass::ttimer::constructor [private]`

Constructor for the TTimer instances.

Usage:
```
Type(TTimer) :: T
t=ttimer()
```

Returns

Returns a TTimer object

Definition at line 104 of file TimerClass.f03.

5.2.2.5 copy() `procedure, pass(this) timerclass::ttimer::copy [private]`

Function to copy one TTimer instance to the current one via the "=" assignment.

Parameters

`in,out`	*this*	The TTimer instance before the "=" assignment.
`in`	*from*	The TTimer instance after the "=" assignment.

Definition at line 73 of file TimerClass.f03.

5.2.2.6 destructor() `final timerclass::ttimer::destructor [final], [protected]`

Destructor of the TTimer class. Cleans up the instance upon finalization.

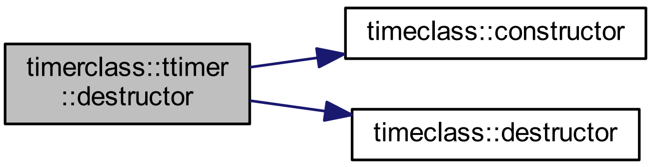
Parameters

`in,out`	*this*	The TTimer instance being destroyed automatically.

Definition at line 81 of file TimerClass.f03.

References timeclass::constructor(), and timeclass::destructor().

Here is the call graph for this function:



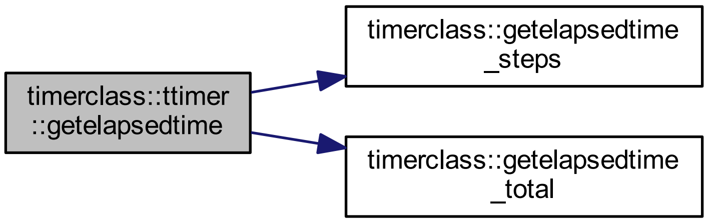
5.2.2.7 getelapsedtime() `generic, public timerclass::ttimer::getelapsedtime`

Generic interface to the GetElapsedTime_total and GetElapsedTime_steps methods of the TTimer class.

Definition at line 76 of file TimerClass.f03.

References timerclass::getelapsedtime_steps(), and timerclass::getelapsedtime_total().

Here is the call graph for this function:


timerclass::ttimer ::getelapsedtime
timerclass::getelapsedtime _steps
timerclass::getelapsedtime _total

5.2.2.8 getelapsedtime_steps() `procedure, pass(this) timerclass::ttimer::getelapsedtime_steps [private]`

Returns the number of seconds which elapsed between two timestamps. This is always a positive value.

NOTE:

- If the timestamps are out of range, then -1 is returned.
- If the start comes after end, then they are exchanged such that a positive time is obtained.

Parameters

`in`	*this*	The TTimer instance.
`in`	*Tstart,Tend*	The indices of the start and end time.
`in`	*INCL_PAUSE*	Logical indicating if the time during which the timer was paused should be included as well. [**OPTIONAL**, **DEFAULT** = .false.]

Returns

Seconds The (fractional) number of seconds elapsed, as double precision real.

Definition at line 72 of file TimerClass.f03.

5.2.2.9 getelapsedtime_total() `procedure, pass(this) timerclass::ttimer::getelapsedtime_total [private]`

Returns the number of seconds which elapsed between the start and stop timestamps.

Parameters

`in`	*this*	The TTimer instance.
`in`	*INCL_PAUSE*	Logical indicating if the time during which the timer was paused should be included as well. [**OPTIONAL**, **DEFAULT** = .false.]

Returns

Seconds The (fractional) number of seconds elapsed, as double precision real.

Definition at line 71 of file TimerClass.f03.

5.2.2.10 getelapsedtimestring() `procedure, pass, public timerclass::ttimer::getelapsedtimestring`

Returns a string representing the elapsed time. +.

Parameters

`in`	*this*	The TTimer instance.
`in`	*TSTART,TSTOP*	Indices of the start and end-times. [**OPTIONAL**, **DEFAULT:** TSTART=1, TSTOP=index of last timestamp]
`in`	*INCL_PAUSE*	Logical indicating if the time during which the timer was paused should be included as well. [**OPTIONAL**, **DEFAULT** = .false.]
`in`	*FMT*	string indicating the format for the time-string. • sec : xxx.xxx secs • hms : xxxx h xx min xx.xxx secs • dhms: xx days xx h xx min xx.xxx secs • hour: xxx.xxx hours • day : xxx.xxx days [**OPTIONAL**, **DEFAULT=** sec]

Returns

str The string with the formatted time.

Definition at line 70 of file TimerClass.f03.

5.2.2.11 interrupt() `procedure, pass, public timerclass::ttimer::interrupt`

Puts the TTimer instance on hold.

If the timer was not running nothing will happen. The optional IO parameter will return an error code.
IO values:

- 0 : all is well
- -1 : The timer was not running, so nothing to pause.
- -2 : The timer was already stopped/paused, so nothing to pause.

In case of error, TS will be set to -1.

Parameters

`in,out`	*this*	The TTimer instance.
`out`	*IO*	Optional parameter giving the error-status. [**OPTIONAL** ; **DEFAULT=** 0]

Returns

TS The index of the final timestamp.

Definition at line 64 of file TimerClass.f03.

5.2.2.12 printelapsedtimereport() `procedure, pass, public timerclass::ttimer::printelapsedtimereport`

Print small timings report to unit UN.

Parameters

`in`	*this*	The TTimer instance.
`in`	*message*	String containing a message to include.
`in`	*Tstart,Tstop*	Integer indexes of the start and stop times
`in`	*UN*	Integer indicating the unit to write the report to.
`in`	*INCL_PAUSE*	Logical indicating if the time during which the timer was paused should be included as well. [**OPTIONAL**, **DEFAULT** = .false.]

Definition at line 69 of file TimerClass.f03.

5.2.2.13 reset() `procedure, pass, public timerclass::ttimer::reset`

Start the TTimer instance. If the timer was already running it is reset first.

Parameters

`in,out`	*this*	The TTimer instance.

Definition at line 68 of file TimerClass.f03.

5.2.2.14 resume() `procedure, pass, public timerclass::ttimer::resume`

Restart the TTimer instance after a pause. If the timer is not paused, nothing will happen and TS=-1 is returned.

Parameters

`in,out`	*this*	The TTimer instance.

Returns

TS The index of the first timestamp.

Definition at line 65 of file TimerClass.f03.

5.2.2.15 start() `procedure, pass, public timerclass::ttimer::start`

Start the TTimer instance (clean start, everything is reset). If the timer was already running it is reset first.

Parameters

`in,out`	*this*	The TTimer instance.

Returns

TS The index of the first timestamp.

Definition at line 63 of file TimerClass.f03.

5.2.2.16 stoptimer() `procedure, pass, public timerclass::ttimer::stoptimer`

Stops the TTimer instance (terminal fashion...no restart possible).

If the timer was not running nothing will happen. The optional IO parameter will return an error code.
IO values:

- 0 : all is well
- -1 : The timer was not running, so nothing to stop.
- -2 : The timer was already stopped, so nothing to stop.

In case of error, TS will be set to -1.

Parameters

`in,out`	*this*	The TTimer instance.
`out`	*IO*	Optional parameter giving the error-status. [**OPTIONAL** ; **DEFAULT=** 0]

Returns

TS The index of the final timestamp.

Definition at line 67 of file TimerClass.f03.

5.2.3 Member Data Documentation

5.2.3.1 maxtimes `integer timerclass::ttimer::maxtimes [private]`

The size of the timestamp list.

Definition at line 54 of file TimerClass.f03.

5.2.3.2 ntimes `integer timerclass::ttimer::ntimes [private]`

The number of timestamps stored.

Definition at line 53 of file TimerClass.f03.

5.2.3.3 paused `logical timerclass::ttimer::paused [private]`

True if the timer is paused.

Definition at line 59 of file TimerClass.f03.

5.2.3.4 running `logical timerclass::ttimer::running [private]`

True if the timer is running.

Definition at line 58 of file TimerClass.f03.

5.2.3.5 stopped `logical timerclass::ttimer::stopped [private]`

True if the timer is stopped (final end)

Definition at line 60 of file TimerClass.f03.

5.2.3.6 timedinterval `logical, dimension(:), allocatable timerclass::ttimer::timedinterval [private]`

logical indicating if the timer was running during an interval between two timestamps or not

Definition at line 56 of file TimerClass.f03.

5.2.3.7 timestamps `type(ttime), dimension(:), allocatable timerclass::ttimer::timestamps [private]`

Allocatable list of timestamps.

Definition at line 55 of file TimerClass.f03.

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

- TimerClass.f03

6 File Documentation

6.1 TimeClass.f03 File Reference

Data Types

- module timeclass::ttime

  The TTime class contains all time functionality with regard to a single time stamp.
- module timeclass::ttime

  The TTime class contains all time functionality with regard to a single time stamp.

Modules

- module timeclass

  *The **TimeClass module** contains the TTime class used by the TTimer class for practical timing.*

Functions/Subroutines

- pure type(ttime) function timeclass::constructor ()

  Constructor for the TTime class.
- subroutine timeclass::settime (this)

  Function to set the TTime instance to the current time, with millisecond resolution.
- pure subroutine timeclass::calculatejdn (this)

  Calculating the Julian Day number based on the Gregorian date set in the TTime object.
- pure subroutine timeclass::setjdn (this, JDN, IO)

  Set the Julian Day Number of a TTime instance manually.
- pure integer(kind=4) function timeclass::getjuliandaynumber (this)

  Function returning the Julian Day Number as a 4-byte integer.
- pure subroutine timeclass::setgregoriandatefromjdn (this, IO)

  Subroutine which transforms the set Julian Day Number into a Gregorian Calender date.
- pure subroutine timeclass::copy (this, from)

  Function to copy one TTime instance to the current one via the "=" assignment. This is the function performing the actual operator overloading.
- pure type(ttime) function timeclass::add (this, that)

  Function to add two TTime instance via the "+" operator. This is the function performing the actual operator overloading.
- pure type(ttime) function timeclass::subtract (this, that)

  Function to subtract two TTime instance via the "-" operator. This is the function performing the actual operator overloading.
- pure logical function timeclass::isleapyear (this)

  Function returning true/false if the year of the TTime instance is a leap year.
- pure character(len=255) function timeclass::gettimestring (this, fmt)

  Returns a string with the time as a string.
- pure real(dp) function timeclass::gettimeseconds (this)

  Function returning the time in (fractional) seconds (double precision).
- subroutine timeclass::destructor (this)

  Destructor of the TTime object instance.

6.2 TimeClass.f03

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!MIT License
!
!Copyright (c) Dr. Dr. Danny E.P. Vanpoucke, https://dannyvanpoucke.be
!
!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
!to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
!copies of the Software, and to permit persons to whom the Software is
!furnished to do so, subject to the following conditions:
!
!The above copyright notice and this permission notice shall be included in all
!copies or substantial portions of the Software.
!
!THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
!IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
!FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
!AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
!LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
!OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
!SOFTWARE.
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
module timeclass
    implicit none
    private


```

```fortran
    !+++++++++++++++++++++++++++++++++++++++
    type, public :: ttime
      private
        integer :: year
        integer :: month
        integer :: day
        integer(kind=4) :: jdn
        real     :: daysecs
    contains
      private
        procedure, pass(this),public :: settime
        procedure, pass(this)        :: calculatejdn
        procedure, pass(this)        :: setjdn
        procedure, pass(this)        :: setgregoriandatefromjdn
        procedure, pass(this),public :: getjuliandaynumber
        procedure, pass(this)        :: copy
        procedure, pass(this)        :: add
        procedure, pass(this)        :: subtract
        procedure, pass(this),public :: isleapyear
        procedure, pass(this),public :: gettimestring
        procedure, pass(this),public :: gettimeseconds
        generic, public :: assignment(=) => copy
        generic, public :: operator(+)   => add
        generic, public :: operator(-)   => subtract
        final :: destructor
    end type ttime

    ! This is the only way a constructor can be created, as no "initial" exists, emulates the C++ constructor behaviour
    interface ttime
        module procedure constructor
    end interface ttime

contains
    !+++++++++++++++++++++++++++++++++++++++++
    pure function constructor()Result(Time)
        type(ttime) :: time

        time%year=0
        time%month=0
        time%day=0
        time%JDN=0
        time%daysecs=0
    end function constructor
    !++++++++++++++++++++++++++++++++++++++++++++++
    subroutine settime(this)
        class(ttime), intent(inout) :: this
        integer time_array(8)

        call date_and_time(values=time_array) ! this function seems to be impure
        this%year=time_array(1)
        this%month=time_array(2)
        this%day=time_array(3)
        this%daysecs=(((real(time_array(5))*60.0 + real(time_array(6)))*60.0)+real(time_array(7)))&
                    & + 0.001*real(time_array(8))
        ! Transform the Gregorian date into a Julian Day Number
        call this%CalculateJDN()

    end subroutine settime
    !+++++++++++++++++++++++++++++++++++++++++++++++
    pure subroutine calculatejdn(this)
        class(ttime), intent(inout) :: this

        this%JDN=int((1461*(this%year+4800+int((this%month-14)/12)))/4)+&
                &int((367*(this%month-2-12*int((this%month-14)/12)))/12)-&
                &int((3*int((this%year+4900+int((this%month-14)/12))/100))/4)&
                &+this%day-32075

    end subroutine calculatejdn
    !+++++++++++++++++++++++++++++++++++++++++++++++
    pure subroutine setjdn(this,JDN,IO)
        class(ttime), intent(inout) :: this
        integer(kind=4), intent(in) :: jdn
        integer, intent(out), optional :: io

        if (jdn>=0) then
            this%JDN=jdn
        else
            this%JDN=0
        end if
        call this%SetGregorianDateFromJDN()

        if (present(io)) then
            io=0
            if (this%JDN<0) io=-1
        end if

    end subroutine setjdn
    !++++++++++++++++++++++++++++++++++++++++++++++
    pure function getjuliandaynumber(this) Result(JDN)
        class(ttime), intent(in) :: this
        integer(kind=4) :: jdn

        jdn=this%JDN

    end function getjuliandaynumber
    !++++++++++++++++++++++++++++++++++++++++++++++
    pure subroutine setgregoriandatefromjdn(this, IO)
        class(ttime), intent(inout) :: this
        integer,intent(out),optional :: io

        integer(kind=4),parameter :: y=4716, j=1401, m=2, n=12, r=4, p=1461, v=3, u=5, s=153, w=2,
 b=274277, c=-38
        integer(kind=4) :: f, e, g, h

        if (present(io)) then
            io=0
            if (this%JDN<0) io=-1
            if (io<0) return
        end if
        f=this%JDN + j +int((int((4*this%JDN + b)/146097)*3)/4) + c
        e=r*f + v
        g=int(mod(e,p)/r)
        h=u*g+w
        this%day=int((mod(h,s))/u)+1
        this%month=mod(int(h/s)+m,n)+1
        this%year=int(e/p)-y+int((n+m-this%month)/n)

    end subroutine setgregoriandatefromjdn
    !++++++++++++++++++++++++++++++++++++++++++++++
    pure subroutine copy(this,from)
        class(ttime), intent(inout) :: this
        class(ttime), intent(in) :: from

        this%year=from%year
        this%month=from%month
        this%day=from%day
        this%daysecs=from%daysecs
        this%JDN=from%JDN

    end subroutine copy
    !++++++++++++++++++++++++++++++++++++++++++++++
    pure function add(this,that) Result(Total)
        class(ttime), intent(in) :: this, that
        Type(ttime) :: total

        integer :: overflow, ios
        integer(kind=4) :: days

        total = ttime()
        total%daysecs=this%daysecs+that%daysecs
        overflow=0
        if (total%daysecs>60.0) then
            overflow=int((total%daysecs - modulo(total%daysecs,60.0))/60.0)
            total%daysecs=modulo(total%daysecs,60.0)
        end if
        ! now using Julian Day Numbers:
        days=this%GetJulianDayNumber()+that%GetJulianDayNumber()+overflow
        call total%SetJDN(days,io=ios) ! this also sets the days, months and years

    end function add
    !++++++++++++++++++++++++++++++++++++++++++++++
    pure function subtract(this,that) Result(Total)
        class(ttime), intent(in) :: this, that
        Type(ttime) :: total

        integer(kind=4) :: overflow, days
        integer :: ios
        !Using julian day numbers and day seconds, this gets a bit more simple

        total = ttime()
        total%daysecs=this%daysecs-that%daysecs
        overflow=0
        do while (total%daysecs<0.0)
            overflow=overflow+1
            total%daysecs=total%daysecs+86400.0
        end do
        days=this%GetJulianDayNumber()-that%GetJulianDayNumber()-overflow
        call total%SetJDN(days,ios)

    end function subtract
    !++++++++++++++++++++++++++++++++++++++++++++++
    pure function isleapyear(this) Result(Leap)
        class(ttime), intent(in) :: this
        logical :: leap

        leap=.false.
        if (mod(this%year,4)==0) then
            leap=.true.
            if (mod(this%year,100)==0) then
                leap=.false.
                if (mod(this%year,400)==0) leap=.true.
            end if
        end if

    end function isleapyear
    !++++++++++++++++++++++++++++++++++++++++++++++
    pure function gettimestring(this,fmt) Result(TS)
    use, intrinsic :: iso_fortran_env
        class(ttime), intent(in) :: this
        character(len=*), intent(in), optional :: fmt
        character(len=255) :: ts

        integer, parameter :: dp = real64
        integer:: h, m
        real :: s
        real(dp) :: fullt
        character(len=50) :: fmtstr


        s=mod(this%daysecs,60.0)
        m=mod(int((this%daysecs-s)/60),60)
        h=int(this%daysecs/3600)
        fmtstr="full"
        if (present(fmt)) fmtstr=fmt

        select case(trim(adjustl(fmtstr)))
            case('full')
                write(ts,'(2(I2,A),I4,2(A,I2),A,F6.3)') this%day,"/",this%month,"/",this%year,"
  ",h,":",m,":",s
            case('date')
                write(ts,'(2(I2,A),I4)') this%day,"/",this%month,"/",this%year
            case('time')
                write(ts,'(2(I2,A),F6.3)') h,":",m,":",s
            case('days')
                fullt=this%GetJulianDayNumber()*1.0_dp + (this%daysecs/86400.0_dp)
                write(ts,'(F20.8,A)') fullt," days"
            case('hours')
                fullt=(this%GetJulianDayNumber()*1.0_dp + (this%daysecs/86400.0_dp))*24.0_dp
                write(ts,'(F20.8,A)') fullt," hours"
            case('seconds')
                fullt=(this%GetJulianDayNumber()*86400.0_dp + this%daysecs)
                write(ts,'(F20.8,A)') fullt," secs"
            case default
                write(ts,'(2(I2,A),I4,2(A,I2),A,F6.3)') this%day,"/",this%month,"/",this%year,"
  ",h,":",m,":",s
        end select

    end function gettimestring
    !+++++++++++++++++++++++++++++++++++++++++++++
    pure function gettimeseconds(this) Result(sec)
    use, intrinsic :: iso_fortran_env
        class(ttime), intent(in) :: this
        integer, parameter :: dp = real64
        real(dp) :: sec

    sec=this%GetJulianDayNumber()*86400.0_dp + this%daysecs

    end function gettimeseconds
    !++++++++++++++++++++++++++++++++++++++++++++
    subroutine destructor(this)
        type(ttime) :: this

    end subroutine destructor


end module
```

6.3 TimerClass.f03 File Reference

Data Types

- module timerclass::ttimer

  A class for keeping track of time in calculations, by creating a list of timestamps.
- module timerclass::ttimer

  A class for keeping track of time in calculations, by creating a list of timestamps.

Modules

- module timerclass

  Module containing the TTimer class for practical timing.

Functions/Subroutines

- pure type(ttimer) function timerclass::constructor ()

  Constructor for the TTimer instances.
- integer function timerclass::start (this)

  Start the TTimer instance (clean start, everything is reset). If the timer was already running it is reset first.
- integer function timerclass::resume (this)

  Restart the TTimer instance after a pause. If the timer is not paused, nothing will happen and TS=-1 is returned.
- integer function timerclass::addtimeflag (this)

  Add an additional timestamp without changing the status (running/paused) of the TTimer instance.
- integer function timerclass::interrupt (this, IO)

  Puts the TTimer instance on hold.
- integer function timerclass::stoptimer (this, IO)

  Stops the TTimer instance (terminal fashion...no restart possible).
- pure subroutine timerclass::reset (this)

  Start the TTimer instance. If the timer was already running it is reset first.
- integer function timerclass::addtimestamp (this)

  Add a timestamp to a running TTimer instance, returning the index of the timestamp.
- pure real(dp) function timerclass::getelapsedtime_total (this, INCL_PAUSE)

  Returns the number of seconds which elapsed between the start and stop timestamps.
- pure real(dp) function timerclass::getelapsedtime_steps (this, Tstart, Tend, INCL_PAUSE)

  Returns the number of seconds which elapsed between two timestamps. This is always a positive value.
- pure character(len=50) function timerclass::getelapsedtimestring (this, TSTART, TSTOP, INCL_PAUSE, F↩
  MT)

  Returns a string representing the elapsed time. +.
- subroutine timerclass::printelapsedtimereport (this, message, Tstart, Tstop, UN, INCL_PAUSE)

  Print small timings report to unit UN.
- pure subroutine timerclass::copy (this, from)

  Function to copy one TTimer instance to the current one via the "=" assignment.
- subroutine timerclass::destructor (this)

  Destructor of the TTimer class. Cleans up the instance upon finalization.

6.4 TimerClass.f03

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!MIT License
!
!Copyright (c) Dr. Dr. Danny E.P. Vanpoucke, https://dannyvanpoucke.be
!
!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
!to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
!copies of the Software, and to permit persons to whom the Software is
!furnished to do so, subject to the following conditions:
!
!The above copyright notice and this permission notice shall be included in all
!copies or substantial portions of the Software.
!
!THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
!IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
!FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
!AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
```

```fortran
!LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
!OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
!SOFTWARE.
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!




!++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
module timerclass
    use timeclass
    implicit none
    private

    !++++++++++++++++++++++++++++++++++++++++++
    type, public :: ttimer
        private
        !! Some EXTRA INFO on \ref ttimer
        integer :: ntimes
        integer :: maxtimes
        Type(ttime), allocatable :: timestamps(:)
        logical, allocatable :: timedinterval(:)
        logical :: running
        logical :: paused
        logical :: stopped
    contains
        private
        procedure, pass(this), public :: start
        procedure, pass(this), public :: interrupt
        procedure, pass(this), public :: resume
        procedure, pass(this), public :: addtimeflag
        procedure, pass(this), public :: stoptimer
        procedure, pass(this), public :: reset
        procedure, pass(this), public :: printelapsedtimereport
        procedure, pass(this), public :: getelapsedtimestring
        procedure, pass(this)         :: getelapsedtime_total
        procedure, pass(this)         :: getelapsedtime_steps
        procedure, pass(this)         :: copy
        procedure, pass(this),        :: addtimestamp
        generic, public :: assignment(=) => copy
        generic, public :: getelapsedtime => getelapsedtime_total, getelapsedtime_steps
        final :: destructor
    end type ttimer


    ! This is the only way a constructor can be created, as no "initial" exists, emulates the C++
 constructor behaviour
    interface ttimer
        module procedure constructor
    end interface ttimer

contains
    !++++++++++++++++++++++++++++++++++++++++++++
    pure function constructor()Result(Timer)
        Type(ttimer) :: timer

    timer%ntimes=0
    timer%maxtimes=10
    allocate(timer%timestamps(1:10))
    allocate(timer%timedInterval(1:10))
    timer%timedInterval(:)=.false.
    timer%running=.false.
    timer%paused=.false.
    timer%stopped=.false.

    end function constructor
    !++++++++++++++++++++++++++++++++++++++++++++
    function start(this) Result(TS)
        class(ttimer), intent(inout) :: this
        integer :: ts

        if (this%running) call this%reset()
        this%running=.true.
        ts=this%AddTimestamp()
        this%ntimes=ts
        this%timedInterval(ts)=.true. ! this timer-interval is accounted

    end function start
    !++++++++++++++++++++++++++++++++++++++++++++
    function resume(this) Result(TS)
        class(ttimer), intent(inout) :: this
        integer :: ts

        ts=-1
        if (this%paused) then
            this%running=.true.
            this%paused=.false.
            ts=this%AddTimestamp()
            this%ntimes=ts
            this%timedInterval(ts)=.true. ! this timer-interval is accounted
        end if

    end function resume
    !++++++++++++++++++++++++++++++++++++++++++++
    function addtimeflag(this) Result(TS)
        class(ttimer), intent(inout) :: this
        integer :: ts

        ts=-1
        if (this%running.or.this%paused) then
            ts=this%AddTimestamp()
            this%ntimes=ts
            this%timedInterval(ts)=this%running ! is this timer-interval accounted?
        end if

    end function addtimeflag
    !++++++++++++++++++++++++++++++++++++++++++++
    function interrupt(this,IO) Result(TS)
        class(ttimer), intent(inout) :: this
        integer, intent(out), optional :: io
        integer :: ts

        ts=-1
        if (this%running) then
            if ((.not.(this%stopped)).and.(.not.(this%paused))) then
                if (present(io)) io=0
                ts=this%AddTimestamp()
                this%paused=.true.
                this%timedInterval(ts)=.false. ! as the timer is pauzed, the following interval should not be accounted
            else
                if (present(io)) io=-2
            end if
        else
            if (present(io)) io=-1
        end if

    end function interrupt
    !++++++++++++++++++++++++++++++++++++++++++++
    function stoptimer(this,IO) Result(TS)
        class(ttimer), intent(inout) :: this
        integer, intent(out), optional :: io
        integer :: ts

        ts=-1
        if (this%running) then
            if (.not.(this%stopped)) then
                if (present(io)) io=0
                ts=this%AddTimestamp()
                this%stopped=.true.
                this%timedInterval(ts)=.false. ! as the timer is stopped, the following interval should not be accounted
            else
                if (present(io)) io=-2
            end if
        else
            if (present(io)) io=-1
        end if

    end function stoptimer
    !++++++++++++++++++++++++++++++++++++++++++++
    pure subroutine reset(this)
        class(ttimer), intent(inout) :: this

        this%ntimes=0
        this%maxtimes=10
        if (allocated(this%timestamps)) deallocate(this%timestamps)
        allocate(this%timestamps(1:10))
        if (allocated(this%timedInterval)) deallocate(this%timedInterval)
        allocate(this%timedInterval(1:10))
        this%timedInterval(:)=.false.
        this%running=.false.
        this%paused=.false.
        this%stopped=.false.

    end subroutine reset
    !++++++++++++++++++++++++++++++++++++++++++++
    function addtimestamp(this)Result(TS)
        class(ttimer), intent(inout) :: this
        integer:: ts

        type(ttime), allocatable :: tmp(:)
        logical, allocatable :: tmpl(:)

        if (this%running) then
            ts=this%ntimes+1
            if (ts>this%maxtimes) then ! we need to extend the arrays
                allocate(tmp(1:this%maxtimes))
                allocate(tmpl(1:this%maxtimes))
                tmp(1:this%maxtimes)=this%timestamps(1:this%maxtimes)
                tmpl(1:this%maxtimes)=this%timedInterval(1:this%maxtimes)
                this%maxtimes=this%maxtimes+10
                if (allocated(this%timestamps)) deallocate(this%timestamps)
                allocate(this%timestamps(1:this%maxtimes))
                if (allocated(this%timedInterval)) deallocate(this%timedInterval)
                allocate(this%timedInterval(1:this%maxtimes))
                this%timestamps(1:this%maxtimes-10)=tmp(1:this%maxtimes-10)
                this%timedInterval(1:this%maxtimes-10)=tmpl(1:this%maxtimes-10)
                this%timedInterval(this%maxtimes-10:this%maxtimes)=.false.
                deallocate(tmp)
                deallocate(tmpl)
            end if
            this%timestamps(ts)=ttime()
            call this%timestamps(ts)%SetTime()
            this%ntimes=ts
        else
            ts=-1
        end if

    end function addtimestamp
    !++++++++++++++++++++++++++++++++++++++++++++
    pure function getelapsedtime_total(this,INCL_PAUSE) Result(sec)
    use, intrinsic :: iso_fortran_env
        class(ttimer), intent(in) :: this
        logical, intent(in), optional :: incl_pause
        integer, parameter :: dp = real64
        real(dp) :: sec

        logical :: pauze

    pauze=.false.
    if (present(incl_pause)) pauze=incl_pause

    sec=this%GetElapsedTime_steps(1,this%maxtimes,incl_pause=pauze)

    end function getelapsedtime_total
    !++++++++++++++++++++++++++++++++++++++++++++
    pure function getelapsedtime_steps(this,Tstart,Tend,INCL_PAUSE) Result(sec)
    use, intrinsic :: iso_fortran_env
        class(ttimer), intent(in) :: this
        integer, intent(in) :: tstart, tend
        logical, intent(in), optional :: incl_pause
        integer, parameter :: dp = real64
        real(dp) :: sec

        type(ttime) :: elap, tmp
        integer :: t1, t2, nr

    if ((tstart<1).or.(tend<1).or.(tstart>this%ntimes).or.(tend>this%ntimes)) then
        sec=-1.0
        return
    end if

    if (tstart>tend) then
        t1=tend
        t2=tstart
    else
        t1=tstart
        t2=tend
    end if

    elap=this%timestamps(t2)-this%timestamps(t1)
    sec=elap%GetTimeSeconds()
    if (present(incl_pause)) then
        if (incl_pause) then ! pauses should be excluded, so we subtract the again
            do nr=1,this%ntimes-1
                if (.not.this%timedInterval(nr)) then
                    tmp=this%timestamps(nr+1)-this%timestamps(nr)
                    sec=sec-tmp%GetTimeSeconds()
                end if

            end do
        end if
    end if

    end function getelapsedtime_steps
    !++++++++++++++++++++++++++++++++++++++++++++++
    pure function getelapsedtimestring(this,TSTART, TSTOP, INCL_PAUSE, FMT) Result(str)
    use, intrinsic :: iso_fortran_env
        class(ttimer), intent(in) :: this
        integer, intent(in), optional :: tstart, tstop
```

```fortran
        logical, intent(in), optional :: incl_pause
        character(len=*), intent(in), optional :: fmt
        character(len=50) :: str

        integer, parameter :: dp = real64
        real(dp) :: sec
        integer :: t1, t2, nr, hour, day, minute
        character(len=4) :: opt
        character(len=255) :: line

        t1=1
        t2=this%ntimes
        if(present(tstart)) then
            if ((tstart>0).and.(tstart<=this%ntimes)) t1=tstart
        end if
        if(present(tstop)) then
            if ((tstop>0).and.(tstop<=this%ntimes)) t2=tstop
        end if
        if (t1>t2) then
            nr=t1
            t1=t2
            t2=nr
        end if

        sec=this%GetElapsedTime(t1,t2,incl_pause)
        !now transform to a string
        opt='sec'
        if (present(fmt)) opt=trim(adjustl(fmt))

        select case(trim(adjustl(opt)))
            case ('sec')
                write(line,'(F30.3,A)') sec," secs "
            case ('hour')
                write(line,'(F30.3,A)') sec/3600.0_dp," hours "
            case ('day')
                write(line,'(F30.3,A)') sec/86400.0_dp," days "
            case ('hms')
                hour=floor(sec/3600.0_dp)
                sec=sec-(hour*3600.0_dp)
                minute=floor(sec/60.0_dp)
                sec=sec-(minute*60.0_dp)
                write(line,'(I0,A,I2,A,F6.3)') hour," h ",minute," min ",sec," secs "
            case ('dhms')
                day=floor(sec/86400.0_dp)
                sec=sec-(day*864000_dp)
                hour=floor(sec/3600.0_dp)
                sec=sec-(hour*3600.0_dp)
                minute=floor(sec/60.0_dp)
                sec=sec-(minute*60.0_dp)
                write(line,'(I0,A,2(I2,A),F6.3)') day," days ",hour," h ",minute," min ",sec," secs "
        end select
        write(str,'(3A)') " ",trim(adjustl(line))," "

    end function getelapsedtimestring
    !++++++++++++++++++++++++++++++++++++++++++++++
    subroutine printelapsedtimereport(this, message, Tstart, Tstop, UN, INCL_PAUSE)
    use, intrinsic :: iso_fortran_env
        class(ttimer), intent(inout) :: this
        character(len=*), intent(in) :: message
        integer, intent(in) :: Tstart, Tstop
        integer, intent(in) :: UN
        logical, intent(in), optional :: INCL_PAUSE

        character(len=50) :: line
        integer, parameter :: dp = real64
        real(dp) :: sec
        integer :: ih, im

        write(un,"(2A)",advance='NO') trim(message)," : "
        line=this%GetElapsedTimeString(tstart,tstop,incl_pause,'sec')
        write(un,"(A)") trim(adjustl(line))
        sec=this%GetElapsedTime(tstart,tstop,incl_pause)

        ih=floor(sec/3600.0_dp)
        sec=sec-dble(ih)*3600.0_dp
        im=floor(sec/60.0_dp)
        sec=sec-dble(im)*60.0_dp

        write (un,'(I8,A)') ih," hours"
        write (un,'(I8,A)') im," minutes"
        write (un,'(F8.3,A)') sec," seconds"

    end subroutine printelapsedtimereport
    !++++++++++++++++++++++++++++++++++++++++++++++
    pure subroutine copy(this,from)
        class(ttimer), intent(inout) :: this
        class(ttimer), intent(in) :: from

    integer :: nr

    this%maxtimes=from%maxtimes
    this%ntimes=from%ntimes
    this%paused=from%paused
    this%running=from%running
    this%stopped=from%stopped
    allocate(this%timedInterval(1:this%maxtimes))
    this%timedInterval(1:this%maxtimes)=from%timedInterval(1:this%maxtimes)
    allocate(this%timestamps(1:this%maxtimes))
    do nr=1, this%ntimes
        this%timestamps(nr)=from%timestamps(nr)
    end do

    end subroutine copy
    !++++++++++++++++++++++++++++++++++++++++++++
    subroutine destructor(this)
        Type(ttimer) :: this

    if (allocated(this%timestamps)) deallocate(this%timestamps)
    if (allocated(this%timedInterval)) deallocate(this%timedInterval)

    end subroutine destructor


end module timerclass
```

Index