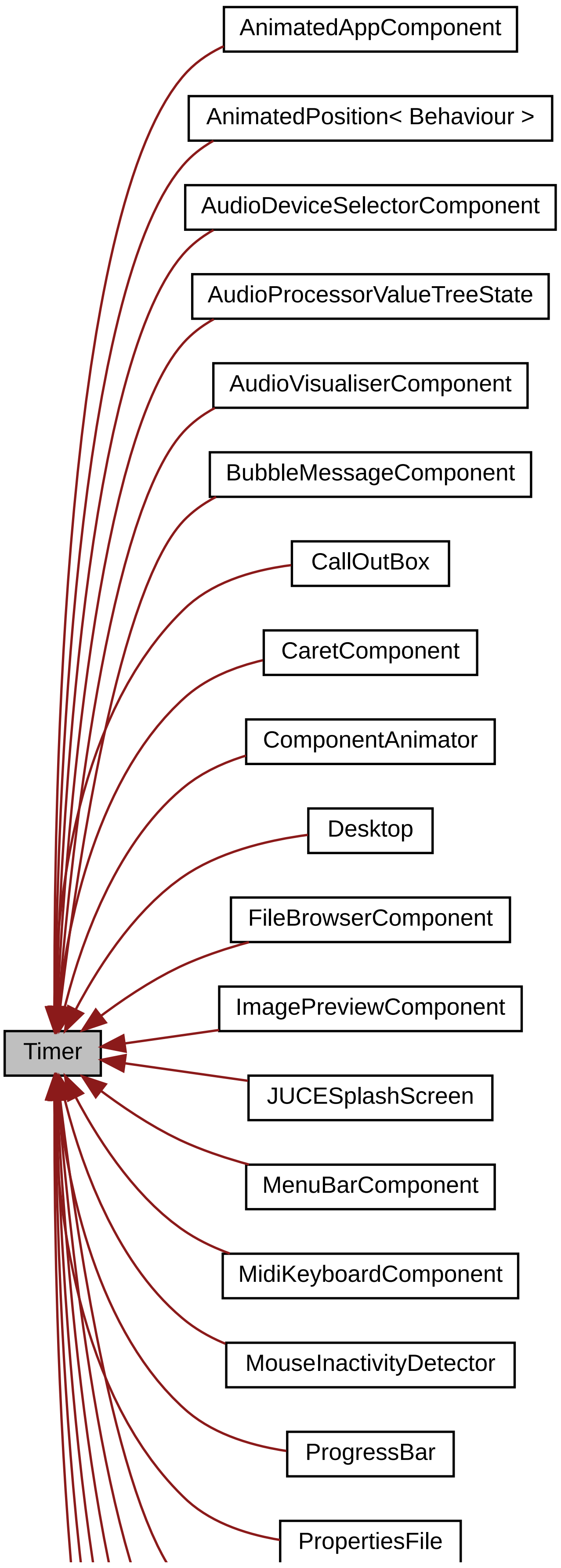
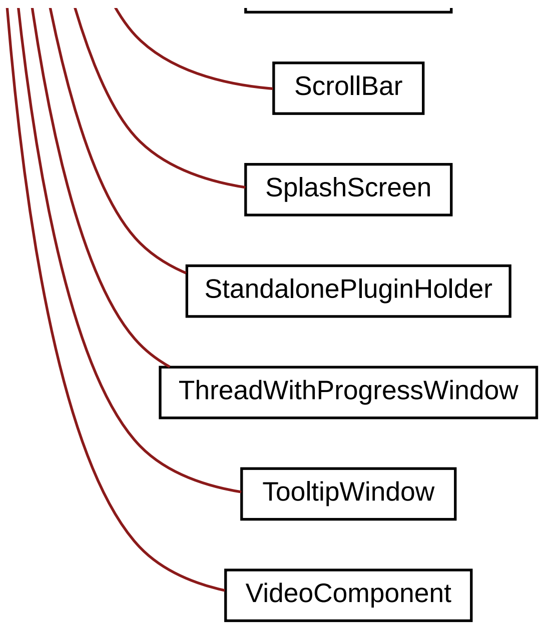
Timer Class Reference abstract

juce_events » timers

Makes repeated callbacks to a virtual method at a specified time interval. More...

Inheritance diagram for Timer:





Public Member Functions

virtual	**~Timer** () Destructor. More...
virtual void	**timerCallback** ()=0 The user-defined callback routine that actually gets called periodically. More...
void	**startTimer** (**int** intervalInMilliseconds) noexcept Starts the timer and sets the length of interval required. More...
void	**startTimerHz** (**int** timerFrequencyHz) noexcept Starts the timer with an interval specified in Hertz. More...
void	**stopTimer** () noexcept Stops the timer. More...
bool	**isTimerRunning** () const noexcept Returns true if the timer is currently running. More...
int	**getTimerInterval** () const noexcept Returns the timer's interval. More...

Static Public Member Functions

static void **JUCE_CALLTYPE**	**callAfterDelay** (**int** milliseconds, std::function< void()> functionToCall) Invokes a lambda after a given number of milliseconds. More...
static void **JUCE_CALLTYPE**	**callPendingTimersSynchronously** () For internal use only: invokes any timers that need callbacks. More...

Protected Member Functions

	Timer () noexcept Creates a **Timer**. More...
	Timer (const **Timer** &) noexcept Creates a copy of another timer. More...

Detailed Description

Makes repeated callbacks to a virtual method at a specified time interval.

A Timer's **timerCallback()** method will be repeatedly called at a given interval. When you create a Timer object, it will do nothing until the startTimer() method is called, which will cause the message thread to start making callbacks at the specified interval, until stopTimer() is called or the object is deleted.

The time interval isn't guaranteed to be precise to any more than maybe 10-20ms, and the intervals may end up being much longer than requested if the system is busy. Because the callbacks are made by the main message thread, anything that blocks the message queue for a period of time will also prevent any timers from running until it can carry on.

If you need to have a single callback that is shared by multiple timers with different frequencies, then the MultiTimer class allows you to do that - its structure is very similar to the Timer class, but contains multiple timers internally, each one identified by an ID number.

See also

HighResolutionTimer, MultiTimer

Constructor & Destructor Documentation

◆ Timer() [1/2]

Timer::Timer ( ) protected noexcept

Creates a Timer.

When created, the timer is stopped, so use startTimer() to get it going.

◆ Timer() [2/2]

Timer::Timer (const Timer &) protected noexcept

Creates a copy of another timer.

Note that this timer won't be started, even if the one you're copying is running.

◆ ~Timer()

virtual Timer::~Timer ( ) `virtual`

Destructor.

Member Function Documentation

◆ timerCallback()

virtual void Timer::timerCallback ( ) `pure virtual`

The user-defined callback routine that actually gets called periodically.

It's perfectly ok to call **startTimer()** or **stopTimer()** from within this callback to change the subsequent intervals.

Implemented in **MidiKeyboardComponent**, **BubbleMessageComponent**, **AudioDeviceSelectorComponent**, and **ImagePreviewComponent**.

◆ startTimer()

void Timer::startTimer (int intervalInMilliseconds) `noexcept`

Starts the timer and sets the length of interval required.

If the timer is already started, this will reset it, so the time between calling this method and the next timer callback will not be less than the interval length passed in.

Parameters

intervalInMilliseconds	the interval to use (any value less than 1 will be rounded up to 1)

Referenced by **StandalonePluginHolder::init()**.

◆ startTimerHz()

void Timer::startTimerHz (int timerFrequencyHz) noexcept

Starts the timer with an interval specified in Hertz.

This is effectively the same as calling startTimer (1000 / timerFrequencyHz).

Referenced by AnimatedPosition< Behaviour >::endDrag(), and AnimatedPosition< Behaviour >::nudge().

◆ stopTimer()

void Timer::stopTimer ( ) noexcept

Stops the timer.

No more timer callbacks will be triggered after this method returns.

Note that if you call this from a background thread while the message-thread is already in the middle of your callback, then this method will cancel any future timer callbacks, but it will return without waiting for the current one to finish. The current callback will continue, possibly still running some of your timer code after this method has returned.

Referenced by AnimatedPosition< Behaviour >::beginDrag(), AnimatedPosition< Behaviour >::setPosition(), and StandalonePluginHolder::~StandalonePluginHolder().

◆ isTimerRunning()

bool Timer::isTimerRunning ( ) const noexcept

Returns true if the timer is currently running.

◆ getTimerInterval()

int Timer::getTimerInterval ( ) const noexcept

Returns the timer's interval.

Returns

the timer's interval in milliseconds if it's running, or 0 if it's not.

◆ callAfterDelay()

```
static void JUCE_CALLTYPE Timer::callAfterDelay ( int                      milliseconds,
                                                  std::function< void()>   functionToCall
                                                )                                          static
```

Invokes a lambda after a given number of milliseconds.

◆ callPendingTimersSynchronously()

```
static void JUCE_CALLTYPE Timer::callPendingTimersSynchronously ( )                        static
```

For internal use only: invokes any timers that need callbacks.

Don't call this unless you really know what you're doing!

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

- **juce_Timer.h**