



## Clocks

### **CLOCK\_1 (alias CLOCK\_SYSTEM)**

- real-time HAL extension
- resolution of 1 ms.
- threads in both the Win32 and RTSS environments

### **CLOCK\_2 (alias CLOCK\_FASTEST)**

- real-time HAL extension
- resolution of 1  $\mu$ S.
- the period is variable and can be set to 100, 200, 500, or 1000  $\mu$ S.

### **CLOCK\_3 — CLOCK\_3**

- real-time HAL extension
- resolution of 1  $\mu$ S.
- the period for timers scheduled on this clock is calculated through the number of Programmable Timer ticks passed in the period and one-tick interval 838.095 nanosecond. This clock has a precision of .001 nanosecond.



## Clocks

Specified Timer Period (unit: $\mu\text{S}$ )	<b>Ticks</b>	CLOCK_2 Period (unit: $\mu\text{S}$ )	CLOCK_3 Period (0.001 nanosecond)
100	119	100	99,733,305
200	239	200	200,304,705
500	596	500	499,504,620
1000	1193	1000	999,847,335



## Sleep(...)

**Sleep** suspends the current process for the specified time.

### Prototype

```
void Sleep(ULONG milliseconds)
```

### Parameters

*milliseconds* - The amount of time to sleep, expressed as milliseconds.

### Return Values

The function returns no value.



## RtSleepFt(...)

**RtSleepFt** suspends the current thread for the specified time.

### Prototype

```
void RtSleepFt(PLARGE_INTEGER pDuration)
```

### Parameters

*pDuration* - a pointer to a LARGE\_INTEGER structure indicating the amount of time to sleep, in 100ns units. *pDuration* must be less than or equal to one second, and must be greater than or equal to the minimum timer period of the system.

### Return Values

This function returns no value.

### Comments

An expiration interval of 0 yields the process to other equal priority runnable threads (if any).



## RtGetClockResolution(...)

**RtGetClockResolution** obtains the resolution of the specified clock.

### Prototype

```
BOOL RtGetClockResolution(CLOCK clock, PLARGE_INTEGER pResolution)
```

### Parameters

Clock - a clock identifier.  
pResolution - a pointer to a LARGE\_INTEGER structure in which to store the results.

### Return Values

The function returns TRUE if it completes successfully, or it returns FALSE if invalid parameters are specified.



## RtGetClockTime(...)

**RtGetClockTime** obtains the current value of the specified clock.

### Prototype

```
BOOL RtGetClockTime(CLOCK clock, PLARGE_INTEGER pTime)
```

### Parameters

**clock** - a clock identifier.

**pTime** - a pointer to a **LARGE\_INTEGER** structure in which to store the results.

### Return Values

The function returns **TRUE** if it completes successfully, or it returns **FALSE** if invalid parameters are specified.



## RtGetClockTimerPeriod(...)

**RtGetClockTimerPeriod** obtains the minimum timer period of the specified clock.

### Prototype

```
BOOL RtGetClockTimerPeriod(CLOCK clock, PLARGE_INTEGER pTime)
```

### Parameters

**clock** - a clock identifier.

**pTime** - a pointer to a LARGE\_INTEGER structure in which to store the results.

### Return Values

The function returns TRUE if it completes successfully, or it returns FALSE if invalid parameters are specified.



## RtSetClockTime(...)

**RtSetClockTime** sets the current value of the specified clock.

### Prototype

```
BOOL RtSetClockTime(CLOCK clock, PLARGE_INTEGER pTime)
```

### Parameters

**clock** - a clock identifier.

**pTime** - a pointer to a **LARGE\_INTEGER** structure specifying the new value for **clock**.

### Return Values

The function returns **TRUE** if it completes successfully, or it returns **FALSE** if invalid parameters are specified.



## RtCreateTimer(...)

**RtCreateTimer** creates a timer associated with the specified clock, and returns a handle to the timer.

### Prototype

```
HANDLE RtCreateTimer(  
    PSECURITY_ATTRIBUTES pThreadAttributes,  
    ULONG StackSize,  
    VOID (RTFCNDCL *Routine) (PVOID context),  
    PVOID Context,  
    ULONG Priority,  
    CLOCK Clock  
)
```



## RtCreateTimer(...)

### Parameters

- pThreadAttributes** - ignored by RTSS.
- StackSize** - the stack size for handler thread. Use a size of 0 for default.
- Routine** - a pointer to the routine to be run upon completion. The routine takes a single PVOID argument and returns VOID.
- Context** - the argument to the routine, cast as a PVOID.
- Priority** - the handler thread priority.
- Clock** - a clock identifier.

### Return Value

If successful, the function returns a non-zero handle to the timer; otherwise, it returns a NULL handle.



## RtCreateTimer(...)

```
/* *****  
 * Create periodical Timer  
 * ***** */  
hMain_timer = NULL;  
hMain_timer = RtCreateTimer(  
    NULL,          /* security_attributes are ignored */  
    0,             /* STACK size is default */  
    OnPCI1002Timer, /* pointer to a completion function */  
    NULL,         /* argument to a completion function */  
    RT_PRIORITY_MAX, /* priority */  
    CLOCK_2       /* H&L timer */  
);  
  
if (hMain_timer == NULL) {  
    sprintf(message, "ERROR\tIniPCI1002Timer::RtCreateTimer()\tCan not create 'Main_timer'.  
        Error code = %d. Application will be terminated.", GetLastError());  
    LOG_MESSAGE();  
    return FALSE;  
}  
sprintf(message, "OK\t\tPeriodic timer 'Main_timer' was created. Handler = 0x%X.", hMain_timer);  
LOG_MESSAGE();
```

---

```
/* *****  
 * OnPCI1002Timer(...)  
 *  
 * Description:  
 * MainTimer service routine  
 * ***** */  
void RTFCNDCL OnPCI1002Timer(PVOID context) {  
    sprintf(message, "OK\t\t'Main_timer' expired.");  
    LOG_MESSAGE();  
}
```



## RtSetTimerRelative(...)

**RtSetTimerRelative** sets the expiration time (relative) and repeat interval on the specified timer.

### Prototype

```
BOOL RtSetTimerRelative(  
    HANDLE hTimer,  
    PLARGE_INTEGER pExpiration,  
    PLARGE_INTEGER pInterval  
)
```



## RtSetTimerRelative(...)

### Parameters

#### **hTimer**

- an RTX-specific handle to the timer.

#### **pExpiration**

- a pointer to a LARGE\_INTEGER structure indicating the time (in 100 ns units) for the initial expiration of the timer. Expiration is calculated **relative** to the current value of the clock associated with the timer at creation.

#### **pInterval**

- a pointer to a LARGE\_INTEGER structure indicating the amount of time (in 100 ns units) between the first expiration and each successive expiration.

### Return Value

The function returns TRUE if it completes successfully, or it returns FALSE if invalid parameters are specified.



## RtSetTimerRelative(...)

```
/*  
*****  
* Set the timing period  
*****/  
/*  
Period is in 100 ns unit, thus 5 = 0.5us, 50 = 5us ...  
Period mus be ineegral multiplifier of the H&L period (default is 500 us)  
*/  
main_timer_period.QuadPart = (LONGLONG) (TSAMP * 10000000);  
if (! RtSetTimerRelative( hMain_timer, &main_timer_period, &main_timer_period) ) {  
    sprintf(message, "ERROR\tIniPCI1002Timer::RtSetTimerRelative()\tCan not set 'Main_timer' period,  
                Error = %d. Application will be terminated.", GetLastError());  
    LOG_MESSAGE();  
    return FALSE;  
}  
sprintf(message, "OK\t\t'Main_timer' period was set to %d us.", main_timer_period.QuadPart/10);  
LOG_MESSAGE();
```



## RtSetTimer(...)

**RtSetTimer** sets the expiration time (absolute) and repeat interval on the specified timer.

### Prototype

```
BOOL RtSetTimerRelative(  
    HANDLE hTimer,  
    PLARGE_INTEGER pExpiration,  
    PLARGE_INTEGER pInterval  
)
```



## RtSetTimer(...)

### Parameters

#### **hTimer**

- an RTX-specific handle to the timer.

#### **pExpiration**

- a pointer to a LARGE\_INTEGER structure indicating the **absolute** time (in 100 ns units) for the initial expiration of the timer. If the value of the expiration time is less than zero, the call is interpreted as a request to set the timer relative to current time on the associated clock.

#### **pInterval**

- a pointer to a LARGE\_INTEGER structure indicating the amount of time (in 100 ns units) between the first expiration and each successive expiration.

### Return Value

The function returns TRUE if it completes successfully, or it returns FALSE if invalid parameters are specified.



## RtGetTimer(...)

**RtGetTimer** returns the remaining relative time until the next expiration of the specified timer.

### Prototype

```
BOOL RtGetTimer(HANDLE hTimer, PLARGE_INTEGER pTimeRemaining)
```

### Parameters

**hTimer** - an RTX-specific handle to the timer.

**pTimeRemaining** - a pointer to a LARGE\_INTEGER structure in which to store the remaining time until next expiration.

### Return Value

The function returns TRUE if it completes successfully, or it returns FALSE if invalid parameters are specified.



## RtCancelTimer(...)

**RtCancelTimer** cancels the expiration of the indicated timer.

### Prototype

```
BOOL RtCancelTimer(HANDLE hTimer, PLARGE_INTEGER pTimeRemaining)
```

### Parameters

- hTimer** - an RTX-specific handle to the timer.
- pTimeRemaining** - a pointer to a **LARGE\_INTEGER** to store the time remaining on the canceled timer. If the pointer is non-NULL, the **LARGE\_INTEGER** will be written with the time (in 100 ns units) remaining on the timer at the time of cancellation. The time remaining is calculated relative to the current value of the clock associated with the time at creation.

### Return Values

The function returns **TRUE** if it completes successfully, or it returns **FALSE** if invalid parameters are specified.



## RtDeleteTimer(...)

**RtDeleteTimer** deletes the timer specified by the given handle.

### Prototype

```
BOOL RtDeleteTimer(HANDLE hTimer)
```

### Parameters

*hTimer* - an RTX-specific handle to the timer to be deleted.

### Return Values

The function returns TRUE if it completes successfully, or it returns FALSE if invalid parameters are specified.



## RtDeleteTimer(...)

```
/*
*****
* Delete Main_timer
*****
*/
if (hMain_timer != NULL) {
    if (RtDeleteTimer(hMain_timer)) {
        sprintf(message, "OK\t\t't'Main_timer' was deleted. Handler was closed.");
        LOG_MESSAGE();
    }
    else {
        sprintf(message, "ERROR\tTerminateApplication::RtDeleteTimer()\t't'Main_timer' was NOT deleted.");
        LOG_MESSAGE();
    }
}
else {
    sprintf(message, "ERROR\tTerminateApplication::hMain_timer\t't'Main_timer' does not exist!.
        This message has not be seen.");
    LOG_MESSAGE();
}
```