Portable Systems Group

Windows NT Timer Specification

Author: *David N. Cutler*

Original Draft 1.0, May 12, 1989
Revision 1.1, July 15, 1989
Revision 1.2, August 8, 1989
Revision 1.3, January 6, 1990

1. Introduction

This specification describes the **Windows NT** *timer object* which is used to record the passage of time. A timer object is set to a specified time, and then expires when the time becomes due. When a timer object is set, its state is changed to Not-Signaled, and it is inserted in the timer queue according to its expiration time. When the timer expires, it is removed from the timer queue and its state is set to Signaled.

When a timer is set, an Asynchronous Procedure Call (**APC**) routine can optionally be specified. This routine is called asynchronously in the context of the establishing thread when the timer expires.

Waiting for a timer object causes the execution of the subject thread to be suspended until the timer attains a state of Signaled. Satisfying the Wait for a timer does not cause the state of the timer to change. Therefore, when a timer attains a Signaled state, an attempt is made to satisfy as many Waits as possible.

API's that support the timer object include:

NtCreateTimer - Create a timer object and open a handle to it
NtOpenTimer - Open a handle to existing timer object
NtCancelTimer - Cancel a timer object that is set to expire
NtQueryTimer - Get information about a timer object
NtSetTimer - Set a timer object to expire at a specified time

2. Create Timer Object

A timer object can be created and a handle opened for access to the object with the **NtCreateTimer** function:

NTSTATUS
NtCreateTimer (
OUT PHANDLE *TimerHandle*,
IN ULONG *DesiredAccess*,
IN POBJECT_ATTRIBUTES *ObjectAttributes* **OPTIONAL**
);

Parameters:

TimerHandle - A pointer to a variable that receives the timer object handle value.

DesiredAccess - The desired types of access to the timer object. The following type specific access flags can be specified in addition to the

STANDARD_ACCESS_REQUIRED flags described in the Object Management Specification.

DesiredAccess Flags:

TIMER_QUERY_STATE - Query access to the timer object is desired.

TIMER_MODIFY_STATE - Modify access (set and cancel) to the timer object is desired.

SYNCHRONIZE - Synchronization access (wait) to the timer object is desired.

TIMER_ALL_ACCESS - All possible types of access to the timer object are desired.

ObjectAttributes - An optional pointer to a structure that specifies the object atributes; refer to the Object Management Specification for details.

If the *OBJ_OPENIF* flag is specified, and a timer object with the specified name already exists, then a handle to the existing object is opened, provided the desired access types can be granted. Otherwise, a new timer object is created with an initial state of Not-Signaled and a handle is opened to the new timer object.

3. Open Timer Object

A handle can be opened to an existing timer object with the **NtOpenTimer** function:

```
NTSTATUS
NtOpenTimer (
    OUT PHANDLE TimerHandle,
    IN ULONG DesiredAccess,
    IN POBJECT_ATTRIBUTES ObjectAttributes
        );
```

Parameters:

TimerHandle - A pointer to a variable that receives the timer object handle value.

DesiredAccess - The desired types of access to the timer object. The following type specific access flags can be specified in addition to the *STANDARD_ACCESS_REQUIRED* flags described in the Object Management Specification.

DesiredAccess Flags:

TIMER_QUERY_STATE - Query access to the timer object is desired.

TIMER_MODIFY_STATE - Modify access (set and cancel) to the timer object is desired.

SYNCHRONIZE - Synchronization access (wait) to the timer object is desired.

TIMER_ALL_ACCESS - All possible types of access to the timer object are desired.

ObjectAttributes - A pointer to a structure that specifies the object atributes; refer to the Object Management Specification for details.

If the desired types of access can be granted, then a handle is opened to the specified timer object.

4. Cancel Timer

A timer can be cancelled with the **NtCancelTimer** function:

```
NTSTATUS
NtCancelTimer (
    IN HANDLE TimerHandle,
    OUT PBOOLEAN CurrentState OPTIONAL
    );
```

Parameters:

TimerHandle - An open handle to a timer object.

CurrentState - An optional pointer to a boolean variable that receives the current state of the timer object.

Canceling a timer object causes the timer to be removed from the timer queue if it is currently set, and returns the current state of the timer. If the current state of the timer object is Not-Signaled, then a value of **FALSE** is returned. Otherwise, the current state of the timer object is Signaled and a value of **TRUE** is returned.

Canceling a timer object that is not currently set to expire has no effect on the timer. Canceling a timer object also does not affect the state of the timer object.

5. Query Timer

The state of a timer can be queried with the **NtQueryTimer** function:

NTSTATUS
NtQueryTimer (
    IN HANDLE *TimerHandle*,
    IN TIMERINFOCLASS *TimerInformationClass*,
    OUT PVOID *TimerInformation*,
    IN ULONG *TimerInformationLength*,
    OUT PULONG *ReturnLength* **OPTIONAL**
    );

Parameters:

TimerHandle - An open handle to a timer object.

TimerInformationClass - The timer information class for which information is to be returned.

TimerInformation - A pointer to a buffer that will receive the specified information. The format and content of the buffer is dependent on the specified information class.

TimerInformation Format by Information Class:

TimerBasicInformation - Data type is *TIMERBASICINFO*.

TIMERBASICINFO Structure:

TIME *RemainingTime* - The amount of time remaining before the timer will expire.

BOOLEAN *TimerState* - The current state of the timer.

TimerInformationLength - Specifies the length in bytes of the timer information buffer.

ReturnLength - An optional pointer which, if specified, receives the number of bytes placed in the timer information buffer.

This function provides the capability to determine the state of a timer object and how much time remains before the timer will expire.

If the current state of the timer object is Not-Signaled, then a value of **FALSE** is returned. Otherwise the current state of the timer object is Signaled and a value of **TRUE** is returned.

The remaining time is returned as the difference between the expiration time of the timer and the current system time. If the timer has already expired, then a negative time is returned which represents the amount of time that has lapsed since the timer expired. Otherwise, a positive value is returned that represents the amount of time remaining before the timer will expire.

6. Set Timer

A timer can be set to expire at a specified time with the **NtSetTimer** function:

NTSTATUS
NtSetTimer (
    IN HANDLE *TimerHandle*,
    IN PTIME *DueTime*,
    IN PTIMER_APC_ROUTINE *TimerApcRoutine* **OPTIONAL**,
    IN PVOID *TimerContext* **OPTIONAL**,
    OUT PBOOLEAN *PreviousState* **OPTIONAL**
    );

Parameters:

TimerHandle - An open handle to a timer object.

DueTime - The absolute or relative time at which the timer is to expire.

TimerApcRoutine - An optional pointer to a function that is called asynchronously when the timer expires. If this parameter is not specified, then the *TimerContext* parameter is ignored.

TimerContext - A pointer to an arbitrary data structure that is passed to the function specified by the *TimerApcRoutine* parameter. This parameter is ignored if the *TimerApcRoutine* parameter is not specified.

PreviousState - An optional pointer to a boolean variable that receives the previous state of the timer.

The function specified by the *TimerApcRoutine* parameter has the following type definition:

```
typedef
VOID
(*PTIMER_APC_ROUTINE) (
    IN PVOID TimerContext,
    IN ULONG TimerLowValue,
    IN LONG TimerHighValue
    );
```

Parameters:

TimerContext - A pointer to an arbitrary data structure which was specified when the timer was set.

TimerLowValue - The low half of the timer expiration time.

TimerHighValue - The high half of the timer expiration time.

Setting a timer object causes the absolute expiration time to be computed, the state of the timer set to Not-Signaled, and the timer object to be inserted in the timer queue.

If the timer is already in the timer queue, then it is implicitly canceled before it is set to the new expiration time.

The expiration time of the timer object is specified as either the absolute time that the timer is to expire, or a time relative to the current system time. If the value of the *DueTime* parameter is negative, then the expiration time is relative. Otherwise, the expiration time is absolute.

If an Asynchronous Procedure Call (**APC**) routine is specified, then the respective procedure is called in the context of the subject thread when the timer expires. The subject thread is also the only thread that can cancel the timer.

When the timer expires, it is removed from the timer queue and its state is set to Signaled.

Revision History:

Original Draft 1.0, May 12, 1989

Revision 1.1, July 15, 1989

1. Change type name of timer **APC** routine.

2. Remove restriction that only the thread that set a timer with an **APC** routine could cancel the timer.

Revision 1.2, August 8, 1989

1. Change the output parameters of NtCancelTimer and NtSetTimer to be optional.

Revision 1.3, January 6, 1990

1. Change type name of object attributes parameter and refer to the Object Management Specification for the definition of this parameter.

2. Change the description of the desired access flags to include the standard rights, object specific rights, and generic rights.

3. Delete the handle flags and object name parameter from the **NtOpenTimer** service and replace with a pointer to an object attributes structure.