Portable Systems Group

Windows NT Event - Semaphore Specification

Author: *Lou Perazzoli*

Original Draft 1.0, January 5, 1989
Revision 1.3, May 11, 1989
Revision 1.4, August 8, 1989
Revision 1.5, October 23, 1989
Revision 1.6, December 1, 1989
Revision 1.7, January 3, 1990
Revision 1.8, January 23, 1990

Copyright (c) Microsoft Corporation - Use subject to the Windows Research Kernel License

Copyright (c) Microsoft Corporation - Use subject to the Windows Research Kernel License

1. Introduction

This specification describes the **Windows NT** event and semaphore objects and the wait services. A definition and an explanation of operation is given for each API. No attempt has been made, however, to fully explain all error conditions and their consequences.

The APIs described include:

NtCreateEvent - create event and open handle
NtOpenEvent - open handle to existing event
NtSetEvent - set event to Signal state
NtResetEvent - set event to Not-Signaled state
NtPulseEvent - set / reset event state atomically
NtQueryEvent - get information about event
NtCreateSemaphore - create semaphore and open handle
NtOpenSemaphore - open handle to existing semaphore
NtReleaseSemaphore - release semaphore
NtQuerySemaphore - get information about semaphore
tDelayExecution - delay execution for the specified time
NtClose - close an object handle

2. Event Objects

There are two types of event objects, *notification events* and *synchronization events*. Notification event objects provide a mechanism for notification. Notification events are either *Signaled* (TRUE) or *Not-Signaled* (FALSE). An event may be set multiple times, yet the state remains Signaled. Notification events provides no ownership capability. If multiple threads are waiting on a notification event, then when the event becomes Signaled, **all** threads waiting for the event are made "runnable". A notification event becomes Not-Signaled only when explicitly reset.

Synchronization event objects have the property that when the event is set, the event attains a state of Signaled, which releases a single thread currently waiting on the event, and then the event immediately attains a state of Not-Signaled. If there are no threads waiting on the event, the state of the event remains Signaled. This allows threads to "synchronize" on the signaling of the event. Like notification events, synchronization events provide no ownership capability.

A synchronization event attains a state of Not-Signaled when explicitly reset or when the first wait operation is satisfied on the event. Note that any time an event attains a state of Not-Signaled, the event count for the state of the event is set to zero.

Copyright (c) Microsoft Corporation - Use subject to the Windows Research Kernel License

2.1 Create Event Object

An event object is created and a handle opened for access to the object with the **NtCreateEvent** function:

NTSTATUS
NtCreateEvent (
    OUT PHANDLE *EventHandle,*
    IN ULONG *DesiredAccess,*
    IN POBJECT_ATTRIBUTES *ObjectAttributes* **OPTIONAL**,
    IN EVENT_TYPE *EventType,*
    IN BOOLEAN *InitialState*
    );

Parameters:

EventHandle - A pointer to a variable that receives the event object handle value.

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

DesiredAccess Flags

EVENT_QUERY_STATE - Query access to the event is desired.

EVENT_MODIFY_STATE - Modify state access (set and reset) to the event is desired.

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

ObjectAttributes - An optional pointer to a structure that specifies the object's attributes. Refer to the *Object Management Specification* for details.

EventType - The type of event object to be created. One of *NotificationEvent* or *SynchronizationEvent*.

InitialState - The initial state of the event object, one of *TRUE* or *FALSE*. If the *InitialState* is specified as TRUE, the event's current state value is set to one, otherwise it is set to zero.

The **NtCreateEvent** function creates an event object with the specified initial state. If an event is in the Signaled state (TRUE), a wait operation on the event does not

Copyright (c) Microsoft Corporation - Use subject to the Windows Research Kernel License

block. If the event is in the Not-Signaled state (FALSE), a wait operation on the event blocks until the specified event attains a state of Signaled, the timeout value is exceeded, or an alert is delivered.

2.2 Open Event Object

A handle can be opened to an existing event object with the **NtOpenEvent** function:

NTSTATUS
NtOpenEvent *(*
    OUT PHANDLE *EventHandle,*
    IN ULONG *DesiredAccess,*
    IN POBJECT_ATTRIBUTES *ObjectAttributes,*
    );

Parameters:

EventHandle - A pointer to a variable that receives the value of the event object handle value.

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

DesiredAccess Flags

EVENT_QUERY_STATE - Query access to the event is desired.

EVENT_MODIFY_STATE - Modify state access (set and reset) to the event is desired.

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

ObjectAttributes - A pointer to a structure that specifies the object's attributes. Refer to the *Object Management Specification* for details.

2.3 Set Event

An event can be set to the signaled state (*TRUE*) with the **NtSetEvent** function:

Copyright (c) Microsoft Corporation - Use subject to the Windows Research Kernel License

NTSTATUS
NtSetEvent *(*
IN HANDLE *EventHandle,*
OUT PLONG *PreviousState* **OPTIONAL**
);

Parameters:

EventHandle - An open handle to an event object.

PreviousState - An optional pointer to a variable that receives the previous state of the event. Zero is Not-Signaled, non-zero is Signaled. The value indicates the number of times the event has been set since the last reset.

Setting the event causes the event to attain a state of Signaled, which releases all threads currently waiting on the event. Any threads which issue a wait operation on the event do not block and continue to execute. It also increments the event count for the state of the event.

2.4 Reset Event

The state of an event is set to the Not-Signaled state (*FALSE*) using the **NtResetEvent** function:

NTSTATUS
NtResetEvent *(*
IN HANDLE *EventHandle,*
OUT PLONG *PreviousState* **OPTIONAL**
);

Parameters:

EventHandle - An open handle to an event object.

PreviousState - An optional pointer to a variable that receives the previous state of the event. Zero is Not-Signaled, non-zero is Signaled. The value indicates the number of times the event has been set since the last reset.

Once the event attains a state of Not-Signaled, any threads which wait on the event block, awaiting the event to become Signaled. The reset event service sets the event count to zero for the state of the event.

Copyright (c) Microsoft Corporation - Use subject to the Windows Research Kernel License

2.5 Pulse Event

An event can be set to the Signaled state and reset to the Not-Signaled state atomically with the **NtPulseEvent** function:

NTSTATUS
NtPulseEvent *(*
    IN HANDLE *EventHandle,*
    OUT PLONG *PreviousState* **OPTIONAL**
    );

Parameters:

EventHandle - An open handle to an event object.

PreviousState - An optional pointer to a variable that receives the previous state of the event. Zero is Not-Signaled, non-zero is Signaled. The value indicates the number of times the event has been set since the last reset.

Pulsing the event causes the event to attain a state of Signaled, which releases all threads currently waiting on the event, and then attain a state of Not-Signaled. The pulse event service sets the event count to zero for the state of the event.

2.6 Query Event

The state of an event can be queried with the **NtQueryEvent** function:

NTSTATUS
NtQueryEvent *(*
    IN HANDLE *EventHandle,*
    IN EVENT_INFORMATION_CLASS *EventInformationClass,*
    OUT PVOID *EventInformation,*
    IN ULONG *EventInformationLength,*
    OUT PULONG *ReturnLength* **OPTIONAL**
    );
Parameters:

EventHandle - An open handle to an event object.

EventInformationClass - The event information class about which to retrieve information.

EventInformation - A pointer to a buffer that receives the specified information. The format and content of the buffer depend on the specified event class.

Copyright (c) Microsoft Corporation - Use subject to the Windows Research Kernel License

EventInformation Format by Information Class:

EventBasicInformation - Data type is *EVENT_BASIC_INFORMATION*.

EVENT_BASIC_INFORMATION Structure

EVENT_TYPE *EventType* - The type of the event.

LONG *EventState* - The current state of the event.

EventInformationLength - Specifies the length in bytes of the event information buffer.

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

This function provides the capability to determine the state and granted access of an event object.

3. Semaphore Objects

Semaphore objects provide a mechanism for resource gates. When a semaphore is created, it is provided an initial count and maximum count. When a thread waits on a semaphore, if the current count is greater than zero, then the current count is decremented and the thread continues to execute. If the current count is zero, the thread blocks until the count becomes greater than zero. When a thread releases a semaphore, the current count is augmented. Semaphores do not provide ownership; multiple threads can be waiting and releasing the same semaphore.

3.1 Create Semaphore Object

A semaphore object is created and a handle opened for access to the object with the **NtCreateSemaphore** function:

Copyright (c) Microsoft Corporation - Use subject to the Windows Research Kernel License

NTSTATUS
NtCreateSemaphore (
OUT PHANDLE *SemaphoreHandle,*
IN ULONG *DesiredAccess,*
IN POBJECT_ATTRIBUTES *ObjectAttributes* **OPTIONAL**,
IN LONG *InitialCount,*
IN LONG *MaximumCount*
);

Parameters:

SemaphoreHandle - A pointer to a variable that receives the value of the semaphore object handle.

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

DesiredAccess Flags

SEMAPHORE_QUERY_STATE - Query access to the semaphore is desired.

SEMAPHORE_MODIFY_STATE - Modify state access (release) to the semaphore is desired.

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

ObjectAttributes - An optional pointer to a structure that specifies the object's attributes. Refer to the *Object Management Specification* for details.

InitialCount - The initial count for the semaphore, this value must be positive and less than or equal to the maximum count.

MaximumCount - The maximum count for the semaphore, this value must be greater than zero..

The **NtCreateSemaphore** function causes a semaphore object to be created which contains the specified initial and maximum counts.

3.2 Open Semaphore Object

A handle can be opened to an existing semaphore object with the **NtOpenSemaphore** function:

Copyright (c) Microsoft Corporation - Use subject to the Windows Research Kernel License

NTSTATUS
NtOpenSemaphore *(*
OUT PHANDLE *SemaphoreHandle,*
IN ULONG *DesiredAccess,*
IN POBJECT_ATTRIBUTES *ObjectAttributes*
);

Parameters:

SemaphoreHandle - A pointer to a variable that receives the semaphore object handle value.

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

DesiredAccess Flags

SEMAPHORE_QUERY_STATE - Query access to the semaphore is desired.

SEMAPHORE_MODIFY_STATE - Modify state access (release) to the semaphore is desired.

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

ObjectAttributes - A pointer to a structure that specifies the object's attributes. Refer to the *Object Management Specification* for details.

3.3 Release Semaphore Object

A semaphore object can be released with the **NtReleaseSemaphore** function:

NTSTATUS
NtReleaseSemaphore *(*
IN HANDLE *SemaphoreHandle,*
IN LONG *ReleaseCount,*
OUT PLONG *PreviousCount* **OPTIONAL**
);

Parameters:

SemaphoreHandle - An open handle to a semaphore object.

Copyright (c) Microsoft Corporation - Use subject to the Windows Research Kernel License

ReleaseCount - The release count for the semaphore. The count must be greater than zero and less than the maximum value specified for the semaphore.

PreviousCount - An optional pointer to a variable that receives the previous count for the semaphore.

When the semaphore is released, the current count of the semaphore is incremented by the *ReleaseCount*. Any threads that are waiting for the semaphore are examined to see if the current semaphore value is sufficient to satisfy their wait.

If the value specified by *ReleaseCount* would cause the maximum count for the semaphore to be exceeded, then the count for the semaphore is not affected and an error status is returned.

3.4 Query Semaphore

The state of a semaphore can be queried with the **NtQuerySemaphore** function:

NTSTATUS
NtQuerySemaphore *(*
IN HANDLE *SemaphoreHandle,*
IN SEMAPHORE_INFORMATION_CLASS *SemaphoreInformationClass,*
OUT PVOID *SemaphoreInformation,*
IN ULONG *SemaphoreInformationLength,*
OUT PULONG *ReturnLength* **OPTIONAL**
);

Parameters:

SemaphoreHandle - An open handle to a semaphore object.

SemaphoreInformationClass - The semaphore information class about which to retrieve information.

SemaphoreInformation - A pointer to a buffer which receives the specified information. The format and content of the buffer depend on the specified semaphore class.

SemaphoreInformation Format by Information Class:

SemaphoreBasicInformation - Data type is *SEMAPHORE_BASIC_INFORMATION*.

SEMAPHORE_BASIC_INFORMATION Structure

Copyright (c) Microsoft Corporation - Use subject to the Windows Research Kernel License

LONG *CurrentCount* - The current count of the semaphore.

LONG *MaximumCount* - The maximum count that may be obtained by the semaphore.

SemaphoreInformationLength - Specifies the length in bytes of the semaphore information buffer.

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

This function provides the capability to determine the state and granted access of a semaphore object

4.0 Delay Execution

The execution of the current thread can be delayed for a specified interval of time with the **NtDelayExecution** function:

NTSTATUS
NtDelayExecution (
IN BOOLEAN *Alertable,*
IN PTIME *DelayInterval*
);

Parameters:

Alertable - A boolean value that specifies whether the wait is alertable.

DelayInterval - The absolute or relative time over which the wait is to occur.

The **NtDelayExecution** function causes the current thread to enter a waiting state until the specified interval of time has passed. If *Alertable* is specified as *TRUE*, the wait service completes and a condition of *STATUS_ALERTED* is raised. If an **APC** is delivered while the thread is waiting alertable, the **APC** is invoked and the wait operation re-executed.

Revision History:

Original Draft 1.0, January 5, 1989

Revision 1.2, March 12, 1989

Copyright (c) Microsoft Corporation - Use subject to the Windows Research Kernel License

1. Removed Muxwait object and Mutex object.

Revision 1.3, May 11, 1989

1. Added wait for multiple objects.
2. Added NtDelayExecution

Revision 1.4, August 8, 1989

1. Make return parameters for PreviousState and CurrentState optional.

Revision 1.5, October 23, 1989

1. Changed EventName/SemaphoreName in OBJA structure to ObjectName.
2. Added description of notification and synchronization events.
3. Changed *PreviousState* to return a count that indicates the number of times the event was set since the last reset.
4. Added the *EventType* to the query event call.
5. Changed wait services to describe the abandoned state.

Revision 1.6, December 1, 1989

1. Changed desciption of NtCreateSemaphore, NtCreateEvent, NtOpenSemaphore and NtOpenEvent to use OBJECT_ATTRIBUTES and reference Object Management Specification for detials.
2. Changed PULONG to PLONG for PreviousState argument in NtSetEvent, NtResetEvent, and NtPulseEvent.

Copyright (c) Microsoft Corporation - Use subject to the Windows Research Kernel License

Revision 1.7, January 3, 1990

1. Clarified the behavior of sychronzation events and the state of the event count.

2. Changed desired access flags for NtCreateEvent, NtOpenEvent, NtCreateSemaphore, and NtOpenSemaphore.

3. Removed NtWait description. This is now in the Object Management Specification.

Revision 1.8, January 23, 1990

1. Changed NtReleaseSemaphore to return a failure if the ReleaseCount is greater than the maximum count.

2. Changed NtReleaseSemaphore to require the ReleaseCount to be greater than 0.

Copyright (c) Microsoft Corporation - Use subject to the Windows Research Kernel License