```
        IN ACCESS_MASK AccessMask,
        IN POBJECT_ATTRIBUTES ObjectAttnbutes
);
```

NtOpenSymbolicLinkObject opens an existing symbolic link object.

PARAMETERS

hSymbolicLink	Pointer to the variable that receives handle to the SymbolicLink object.
AccessMask	Type of access requested to the symbolic link object. This can be a combination of any of the following flags: SYMBOLIC_LINK_QUERY or SYMBOLIC_LINK_ALL_ACCESS
ObjectAttributes	Points to the OBJECT_ATTRIBUTES structure containing information about the symbolic link object to be opened, such as name, parent directory, objectflags, and so on.

RETURN VALUE

Returns STATUS_SUCCESS on success and an appropriate error code on failure.

COMMENTS

None.

EQUIVALENT WIN32 API

None.

```
NtQuerySymbolicLInkObject ,
NTSTATUS
    NtQuerySymbolicLinkObject(
        IN HANDLE hSymbolicLink,
        IN OUT PUNICODE_STRING ObjectName,
        OUT PULONG BytesReturned
):
```

NtQuerySymbolicLinkObject returns the object referred to by a given symbolic link object.

PARAMETERS

hSymbolicLink	Handle to the symbolic link object returned using NtOpenSymbolicLinkObject or NtCreateSymbolicLinkObject.

ObjectName	Pointer to the initialized Unicode string. The object name referred to by the given symbolic link is returned. The buffer for the Unicode string must already be allocated.
BytesReturned	Pointer to the variable that returns the number of bytes copied into the buffer pointed to by ObjectName.

RETURN VALUE

Returns STATUS_SUCCESS on success and an appropriate error code on failure.

COMMENTS

The user-mode API call QueryDosDevice enables the query symbolic link object only under the object directory named "\??", whereas this system service enables you to query any symbolic link in the object name space provided you have permission.

EQUIVALENT WIN32 API

QueryDosDevice (limited support)

NtQueryObject

```
NTSTATUS
    NtQueryObject(
        IN HANDLE hObject,
        IN OBJECT_INFORMATION_CLASS InfoClass,
        OUT PVOID Buffer,
        IN ULONG BufferSize,
        OUT PULONG BytesRetumed
);
```

NtQueryObject returns different kinds of information about the object based on the InfoClass.

PARAMETERS

hObject	Handle to the object.
InfoClass	Type of information to be retrieved. This can take values from Oto 4.
Buffer	Pointer to the buffer that receives information about the object.
BufferSize	Size of the buffer in bytes pointed to by Buffer.
BytesReturned	Pointer to the variable that receives the number of bytes copied into the Buffer.

RETURN VALUE

Returns STATUS_SUCCESS on success and an appropriate error code on failure.

COMMENTS

Different information is returned based upon the InfoClass parameter. Here is the layout of Buffer based on the InfoClass:

InfoClass=0

```
typedef struct ObjectBasicInfo_t {
      char Unknown1[8];
      ULONG HandleCount;
      ULONG  ReferenceCount;
      ULONG  PagedQuota;
      ULONG  NonPagedQuota;
      char Unknown2[32];
} OBJECT_BASIC_INFO, *POBJECT_BASIC_INFO:
```

InfoClass=l Variable length structure based on the actual length of the object name.

```
typedef struct ObjectNameInfo_t (
      UNICODE_STRING ObjectName;
      WCHAR ObjectNameBuffer[l];
} OBJECT_NAME_INFO, *POBJECT_NAME_INFO;
```

InfoClass=2 Variable length structure based on the actual length of the object type.

```
typedef struct ObjectTypeInfo_t {
      UNICODE_STRING ObjectTypeName;
      char Unknown[0x58];
      WCHAR ObjectTypeNameBuffer[1];
1 OBJECT_TYPE_INFO, *POBJECT_TYPE_INFO;
```

InfoClass=3 Variable length structure based on the number of object types and actual length of each object type.

```
typedef struct ObjectAllTypeInfo_t {
      ULONG   NumberOfObjectTypes;
      OBJECT_TYPE_INFO ObjectsTypeInfo[l];
} OBJECT_ALL_TYPES_INFO, *POBJECT_ALL_TYPES_INFO;
```

InfoClass=4

```
typedef struct ObjectProtectionInfo_t {
      BOOLEAN blnhent;
```

```
    BOOLEAN  bProtectHandle;
} OBJECT_PROTECTION_INFO, *POBJECT_PROTECTION_INFO;
```

EQUIVALENT WIN32 API
GetHandleInformation (limited support)

NtSetInformationObject

```
NTSTATUS
    NtSetInformationObject(
        IN HANDLE hObject,
        IN OBJECT_INFORMATION_CLASS InfoClass,
        IN PVOID Buffer,
        IN ULONG BufferSize
);
```

NtSetInformationObject changes the attributes of the object based on the InfoClass.

PARAMETERS

hObject	Handle to the object.
InfoClass	Type of information to be set. This should be 4.
Buffer	Pointer to the buffer that contains the information about the object. Buffer should be in the OBJECT_PROTECTION_INFO structure format.
BufferS ize	Size of the buffer in bytes pointed to by Buffer.

RETURN VALUE
Returns STATUS_SUCCESS on success and an appropriate error code on failure.

COMMENTS
None.

EQUIVALENT WIN32 API
None.

NtCreateEvent

```
NTSTATUS
    NtCreateEvent(
        OUT PHANDLE hEvent,
        IN ACCESS_MASK AccessMask,
        IN POBJECT_ATTRIBUTES ObjectAttributes,
```

```
    IN EVENT_TYPE EventType.
    IN BOOLEAN bInitialState

);
```

NtCreateEvent creates a new event object.

PARAMETERS

hEvent	Pointer to the variable that receives handle to the event object.
AccessMask	Type of access requested to the event object. This can be a combination of any of the following flags: EVENT_QUERY_STATE, EVENT_MODIFY_STATE, and EVENT_ALL_ACCESS.
ObjectAttributes	Points to the OBJECT_ATTRIBUTES structure containing the information about the event object to be created, such as name, parent directory, objectflags, and so on.
EventType	Type of the event. This can be either of NotificationEvent, SynchronizationEvent. EventType is enumerated data type defined in NTDEF.H file in DDK.
blnitialState	Specifies whether the initial state of the event will be signaled (TRUE) or not signaled (FALSE).

RETURN VALUE

Returns STATUS_SUCCESS on success and an appropriate error code on failure.

COMMENTS

When the EventType is NotificationEvent and the event is signaled, all the threads waiting on the event are released. The state of the event remains signaled unless someone explicitly calls NtResetEvent or NtClearEvent.

When the EventType is SynchronizationEvent and the event is signaled, only one thread waiting on the event is released. All other threads continue to wait and the state of the event is again reset back to nonsignaled.

EQUIVALENT WIN32 API

CreateEvent

NtOpenEvent

```
NTSTATUS
    NtOpenEvent(
```

```
        OUT PHANDLE hEvent,
        IN ACCESS_MASK DesiredAccess,
        IN POBJECT_ATTRIBUTES ObjectAttributes
);
```

NtOpenEvent opens a handle to the existing named event object.

PARAMETERS

hEvent	Pointer to the variable that receives handle to the event object.
DesiredAccess	Type of access requested to the event object. This can be a combination of any of the following flags: EVENT_QUERY_STATE, EVENT_MODIFY_STATE, and EVENT_ALL_ACCESS.
ObjectAttributes	Points to the OBJECT_ATrRIBUTES structure containing the information about the event object to be opened, such as name, parent directory, objectflags, and so on.

RETURN VALUE
Returns STATUS_SUCCESS on success and an appropriate error code on failure.

COMMENTS
None.

EQUIVALENT WIN32 API
OpenEvent

NtClearEvent

```
NTSTATUS
     NtClearEvent(
          IN HANDLE hEvent
);
```

NtClearEvent sets the state of the event object to nonsignaled.

PARAMETERS

hEvent	Handle to the event object returned using the NtCreateEvent or NtOpenEvent system service.

RETURN VALUE
Returns STATUS_SUCCESS on success and an appropriate error code on failure.

COMMENTS
None.

EQUIVALENT WIN32 API
None.

NtPulseEvent

```
NTSTATUS
    NtPulseEvent(
        IN HANDLE hEvent
        OUT OPTIONAL PULONG PreviousState);
```

NtPulseEvent sets the state of the event object to signaled, releases one or more threads waiting on the event, and sets the event back to nonsignaled.

PARAMETERS

hEvent	Handle to the event object returned using the NtCreateEvent i NtOpenEvent system service.
PreviousState	Pointer to the variable that receives the previous state of the event.

RETURN VALUE
Returns STATUS_SUCCESS on success and an appropriate error code on failure.

COMMENTS
If the event referred to by hEvent is a NoficationEvent, then all the threads waiting on the event are released. If the event is of the type SynchronizationEvent, then only one thread waiting on the event is released.

EQUIVALENT WIN32 API
PulseEvent

NtQueryEvent

```
NTSTATUS
    NtQueryEvenU
        IN HANDLE hEvent,
        IN EVENT_INFO_CLASS InfoClass,
        OUT PVOID EventlnfoBuffer,
```

```
        IN ULONG EventInfoBufferSize,
        OUT PULONG BytesCopied
);
```

NtQueryEvent gets the information about the event object.

PARAMETERS

hEvent	Handle to the event object returned using the NtCreateEvent or NtOpenEvent system service.
InfoClass	Information class requested for the event object. This should be 0.
EventlnfoBuffer	Pointer to the buffer that receives information about the event object.
EventlnfoBufferSize	Size of the buffer (in bytes) pointed to by EventlnfoBuffer.
BytesCopied	Pointer to the variable that receives the number of bytes copied into EventlnfoBuffer.

RETURN VALUE
Returns STATUS_SUCCESS on success and an appropriate error code on failure.

COMMENTS
The buffer pointed to by EventlnfoBufferSize must be at least 8 bytes. The information in the buffer is organized as follows:

```
typedef struct EventInfo_t {
     EVENT_TYPE EventType.
     LONG EventState,
} EVENTINFO, *PEVENTINFO;
```

EQUIVALENT WIN32 API
None.

```
NtResetEvent
NTSTATUS
     NtResetEvent(
          IN HANDLE hEvent,
          OUT OPTIONAL PULONG PreviousState

);
```

NtResetEvent sets the state of the event object to nonsignaled and returns the previous state of the event in PreviousState.

PARAMETERS

hEvent	Handle to the event object returned using the NtCreateEvent or NtOpenEvent system service.
PreviousState	Pointer to the variable that receives the previous state of the event.

RETURN VALUE
Returns STATUS_SUCCESS on success and an appropriate error code on failure.

COMMENTS
None.

EQUIVALENT WIN32 API
ResetEvent (does not return the previous state of the event)

NtSetEvent

```
NTSTATUS
    NtSetEvent(
        IN HANDLE hEvent,
        OUT OPTIONAL PULONG PreviousState,
```

NtSetEvent sets the state of the event object to signaled and returns the previous state of the event in PreviousState.

PARAMETERS

hEvent	Handle to the event object returned using the NtCreateEvent i NtOpenEvent system service.
PreviousState	Pointer to the variable that receives the previous state of the event.

RETURN VALUE
Returns STATUS_SUCCESS on success and an appropriate error code on failure.

COMMENTS
None.

EQUIVALENT WIN32 API
SetEvent (does not return the previous state of the event)

NtCreateEventPair

```
NTSTATUS
     NtCreateEventPair(
          OUT PHANDLE hEventPair.
          IN ACCESS_MASK AccessMask,
          IN POBJECT_ATTRIBUTES ObjectAttn butes
);
```

NtCreateEventPair creates a new event-pair object.

PARAMETERS

hEventPair	Pointer to the variable that receives handle to the event-pair object.
AccessMask	Type of access requested to the event-pair object. This can be a combination of any of the following flags: EVENT_QUERY_STATE, EVENT_MODIFY_STATE, and EVENT_ALL_ACCESS.
ObjectAttributes	Points to the OBJECT_ATTRIBUTES structure containing the information about the event-pair object to be created, such as name, parent directory, objectflags, and so on.

RETURN VALUE
Returns STATUS_SUCCESS on success and an appropriate error code on failure.

COMMENTS
The event-pair object consists of two SynchronizationEvent type event objects. The event-pair objects are used by Windows NT 3.51 to implement Quick LPC. The first event is called the *low event* of the event-pair, and the second event is called the *high event* of the event pair.

EQUIVALENT WIN32 API
None.

NtOpenEventPair
NTSTATUS

```
NtOpenEventPai r(
     OUT PHANDLE hEventPair,
     IN ACCESS_MASK DesiredAccess,
     IN POBJECT_ATTRIBUTES ObjectAttnbutes
);
```

NtOpenEventPair opens a handle to an existing named event-pair object.

PARAMETERS

hEventPair	Pointer to the variable that receives handle to the event-pair object.
DesiredAccess	Type of access requested to the event-pair object. This can be a combination of any of the following flags: EVENT_QUERY_STATE, EVENT_MODIFY_STATE, and EVENT_ALL_ACCESS
ObjectAttributes	Points to the OBJECT_ATTRIBUTES structure containing the information about the event-pair object to be opened, such as name, parent directory, objectflags, and so on.

RETURN VALUE

Returns STATUS_SUCCESS on success and an appropriate error code on failure.

COMMENTS

None.

EQUIVALENT WIN32 API

None.

NtSetHighEventPair

```
NTSTATUS
     NtSetHighEventPair(
          IN HANDLE hEventPair
);
```

NtSetHighEventPair sets the high event of the event-pair to signaled state.

PARAMETERS

hEventPair	Handle to the event-pair.

RETURN VALUE
Returns STATUS_SUCCESS on success and an appropriate error code on failure.

COMMENTS
None.

EQUIVALENT WIN32 API
None.

NtSetHighWaitLowEventPair
NTSTATUS
NtSetHighWaitLowEventPair(
IN HANDLE hEventPair
);

NtSetHighWaitLowEventPair sets the high event of the event-pair to signaled state and waits for the low event of the event pair to get signaled.

PARAMETERS

hEventPair Handle to the event-pair.

RETURN VALUE
Returns STATUS_SUCCESS on success and an appropriate error code on failure.

COMMENTS
None.

EQUIVALENT WIN32 API
None.

NtSetLowEventPair
NTSTATUS
NtSetLowEventPair(
IN HANDLE hEventPair
);

NtSetLowEventPair sets the low event of the event-pair to signaled state.

PARAMETERS

hEventPair Handle to the event-pair.

RETURN VALUE
Returns STATUS_SUCCESS on success and an appropriate error code on failure.

COMMENTS
None.

EQUIVALENT WIN32 API
None.

NtSetLowWaitHighEventPair

```
NTSTATUS
     NtSetLowWaitHighEventPair(
          IN HANDLE hEventPair
);
```

NtSetLowWaitHighEventPair sets the low event of the event-pair to signaled ' state and waits for the high event of the event pair to get signaled.

PARAMETERS

hEventPair Handle to the event-pair.

RETURN VALUE
Returns STATUS_SUCCESS on success and an appropriate error code on failure.

COMMENTS
None.

EQUIVALENT WIN32 API
None.

NtCreateMutant

```
NTSTATUS
     NtCreateMutant(
          OUT PHANDLE hMutex,
          IN ACCESS_MASK AccessMask,
          IN POBJECT_ATTRIBUTES ObjectAttributes,
          IN BOOLEAN blnitialState
);
```

NtCreateMutant creates a new mutex object.

PARAMETERS

hMutex	Pointer to the variable that receives handle to the mutex object.
AccessMask	Type of access requested to the mutex object.
ObjectAttributes	Points to the OBJECT_ATTRIBUTES structure containing the information about the mutex object to be created, such as name, parent directory, objectflags, and so on.
blnitialState	Initial state of the mutex: signaled (TRUE), not signaled (FALSE).

RETURN VALUE
Returns STATUS_SUCCESS on success and an appropriate error code on failure.

COMMENTS
None.

EQUIVALENT WIN32 API
CreateMutex

NtOpenMutant

```
NTSTATUS
     NtOpenMutant(
          OUT PHANDLE hMutex,
          IN ACCESS_MASK DesiredAccess,
          IN POBJECT_ATTRIBUTES ObjectAttributes
);
```

NtOpenMutant opens handle to an existing named mutex object.

PARAMETERS

hMutex	Pointer to the variable that receives handle to the mutex object.
DesiredAccess	Type of access requested to the mutex object.
ObjectAttributes	Points to the OBJECT_ATTRIBUTES structure containing the information about the mutex object to be opened, such as name, parent directory, objectflags, and so on.

RETURN VALUE
Returns STATUS_SUCCESS on success and an appropriate error code on failure.

COMMENTS
None.

EQUIVALENT WIN32 API
OpenMutex

NtQueryMutant

```
NTSTATUS
    NtQueryMutant(
        IN HANDLE hMutex,
        IN MUTEX_INFO_CLASS InfoClass,
        OUT PVOID MutexInfoBuffer,
        IN ULONG MutexInfoBufferSize,
        OUT PULONG BytesCopied
```

NtQueryMutant queries the state information about the mutex object.

PARAMETERS

hMutex	Handle to the mutex object returned using the NtCreateMutant or NtOpenMutant system service.
InfoClass	Information class requested for the mutex object. This should be 0.
MutexInfoBuffer	Pointer to the buffer that receives information about the mutex object.
MutexInfoBufferSize	Size of the buffer (in bytes) pointed to by MutexInfoBuffer.
BytesCopied	Pointer to the variable that receives the number of bytes copied into MutexInfoBuffer.

RETURN VALUE
Returns STATUS_SUCCESS on success and an appropriate error code on failure.

COMMENTS
The buffer pointed to by MutexInfoBufferSize must be at least 8 bytes. The information in the buffer is organized as follows:

```
typedef struct MutexInfo_t (
     LONG MutexState;
     BOOLEAN   bOwnedByCallingThread;
     BOOLEAN bAbandoned;
     USHORT Unused:
] MUTEXINFO, *PMUTEXINFO;
```

If the calling thread of this system service owns the mutex, then bOwnedByCallingThread will be TRUE. MutexState is TRUE if the mutex object is signaled; otherwise, it is FALSE. bAbandoned is TRUE if the thread owning the mutex died without releasing the mutex.

EQUIVALENT WIN32 API
None.

NtReleaseMutant

```
NTSTATUS
     NtReleaseMutant(
          IN HANDLE hMutex,
          OUT OPTIONAL PULONG bSignalled , -
):
```

NtReleaseMutant releases an owned mutex object.

PARAMETERS

hMutex	Handle to the mutex object returned using the NtCreateMutant or NtOpenMutant system service.
bSingalled	Pointer to the variable that receives whether the mutex was set to signaled state as a result of system service. The returned value is 0 if the mutex is set to signaled state.

RETURN VALUE
Returns STATUS_SUCCESS on success and an appropriate error code on failure.

COMMENTS
A mutex can be owned by only one thread at a time. However, the same thread can acquire the same mutex multiple times. If the thread acquires the same mutex multiple times, then it has to call NtReleaseMutant that many times to set the mutex to signaled. The bSignalled variable receives whether the mutex was set to signaled.

EQUIVALENT WIN32 API
ReleaseMutex

```
NtCreateSemaphore
NTSTATUS
     NtCreateSemaphore(
          OUT PHANDLE hSemaphore,
          IN ACCESS_MASK AccessMask,
          IN POBJECT_ATTRIBUTES ObjectAttnbutes,
          IN ULONG InitialCount,
          IN ULONG MaxCount

);
```

NtCreateSemaphore creates a new semaphore object.

PARAMETERS

hSemaphore	Pointer to the variable that receives handle to the semaphore object.
AccessMask	Type of access requested to the semaphore object.
ObjectAttributes	Points to the OBJECT_ATTRIBUTES structure containing the information about the semaphore object to be created, such as name, parent directory, objectflags, and so on.
InitialCount	Initial count of the semaphore.
MaxCount	Maximum count of the semaphore

RETURN VALUE

Returns STATUS_SUCCESS on success and an appropriate error code on failure.

COMMENTS

None.

EQUIVALENT WIN32 API

CreateSemaphore

```
NtOpenSemaphore
NTSTATUS
     NtOpenSemaphore(
          OUT PHANDLE hSemaphore,
          IN ACCESS_MASK DesiredAccess,
          IN POBJECT_ATTRIBUTES ObjectAttnbutes
);
```

NtOpenSemaphore opens handle to an existing named semaphore object.

PARAMETERS

hSemaphore	Pointer to the variable that receives handle to the semaphore object.
DesiredAccess	Type of access requested to the semaphore object.
ObjectAttributes	Points to the OBJECT_ATTRIBUTES structure containing the information about the semaphore object to be opened, such as name, parent directory, objectflags, and so on.

RETURN VALUE
Returns STATUS_SUCCESS on success and an appropriate error code on failure.

COMMENTS
None.

EQUIVALENT WIN32 API
OpenSemaphore

NtQuerySemaphore

```
NTSTATUS
    NtQuerySemaphore(
        IN HANDLE hSemaphore,
        INSEMAPHORE_INFO_CLASSInfoClass,
        OUT PVOID SemaphoreInfoBuffer,
        IN ULONG SemaphoreInfoBufferSize,
        OUT PULONG BytesCopied
);
```

NtQuerySemaphore queries the information about the Semaphore object.

PARAMETERS

hSemaphore	Handle to the semaphore object returned using the NtCreateSemaphore or NtOpenSemaphore system service.
InfoClass	Information class requested for the semaphore object. This should be 0.
SemaphorelnfoBuffer	Pointer to the buffer that receives information about the semaphore object.
SemaphorelnfoBufferSize	Size of the buffer (in bytes) pointed to by SemaphorelnfoBuffer.

BytesCopied	Pointer to the variable that receives the number of bytes copied into SemaphoreInfoBuffer.

RETURN VALUE
Returns STATUS_SUCCESS on success and an appropriate error code on failure.

COMMENTS
The buffer pointed to by SemaphoreInfoBufferSize must be at least 8 bytes. The information in the buffer is organized as follows:

```
typedef struct SemaphoreInfo_t (
     ULONG CurrentCount;
     ULONG MaxCount;
) SEMAPHORE_INFO, *PSEMAPHORE_INFO;
```

EQUIVALENT WIN32 API
None.

```
NtReleaseSemaphore
NTSTATUS
     NtRel easeSemaphore(
          IN HANDLE hSemaphore,
          IN ULONG ReleaseCount,
          OUT PULONG PreviousCount  , ,
          ):
```

NtReleaseSemaphore releases semaphore object ReleaseCount times.

PARAMETERS

hSemaphore	Handle to the semaphore object returned using the NtCreateSemaphore or NtOpenSemaphore system service.
ReleaseCount	Number of times to release semaphore.
PreviousCount	Pointer to the variable that receives the previous count of the semaphore.

RETURN VALUE
Returns STATUS_SUCCESS on success and an appropriate error code on failure.

COMMENTS
None.

EQUIVALENT WIN32 API
ReleaseSemaphore

```
NtSignalAndWaitForSirigleObject(
NTSTATUS
     NtSignalAndWaitForSingleObject(
          IN HANDLE hSignalObject,
          IN  HANDLE  hWaitObject,
          IN BOOLEAN bAlertable,
          IN PLARGE_INTEGER Timeout
);
```

NtSignalAndWaitForSingleObject releases the object specified by hSignalObject and waits on the object specified by hWaitObject.

PARAMETERS

hSignalObject	Handle to the object to be released. This should be a handle to a mutex or event, or a semaphore object.
hWaitObject	Handle to the object to be acquired (wait on). This should be handle to a mutex or event, or a semaphore object.
bAlertable	Flag specifying whether the wait is alertable.
Timeout	Pointer to the variable that contains the timeout for wait.

RETURN VALUE
Returns STATUS_SUCCESS on success and an appropriate error code on failure.

COMMENTS
None.

EQUIVALENT WIN32 API
None.

```
NtWai tForMulti pieObjects(
NTSTATUS
     NtWaitForMultipleObjects(
          IN ULONG nObjects,
          IN HANDLE *ObjectHandleArray,
          IN WAIT_TYPE WaitType,
```

```
        IN BOOLEAN bAlertable,
        IN PLARGE_INTEGER Timeout
);
```

NtWaitForMultipleObjects waits on multiple objects to be released (set to signaled).

PARAMETERS

nObjects	Number of object handles in the array pointed to by ObjectHandleArray.
ObjectHandleArray	Pointer to the array that contains handle to the objects to wait on.
WaitType	Type of wait. This could be WaitAll or WaitAny.
bAlertable	Flag specifying whether the wait is alertable.
Timeout	Pointer to the variable that contains the timeout for wait.

RETURN VALUE

Returns STATUS_SUCCESS on success and an appropriate error code on failure.

COMMENTS

When WaitAll is specified, the function returns only when all the objects referred by handles in ObjectHandleArray are in signaled state, whereas when WaitAny is specified, the function returns when any one of the objects referred by handles in ObjectHandleArray is in signaled state. The wait is abandoned when Timeout occurs, irrespective of WaitType.

EQUIVALENT WIN32 API

WaitForMultipIeObjects

```
NtWaitForSingleObject(
NTSTATUS
     NtWaitForSingleObject(
          IN HANDLE hObject,
          IN BOOLEAN bAlertable,
          IN PLARGE_INTEGER Timeout
):
```

NtWaitForSingleObject waits on the object to be released (set to signaled).

PARAMETERS

hObject	Handle to the object to wait on.
bAlertable	Flag specifying whether the wait is alertable.
Timeout	Pointer to the variable that contains the timeout for wait.

RETURN VALUE
Returns STATUS_SUCCESS on success and an appropriate error code on failure.

COMMENTS
None.

EQUIVALENT WIN32 API
WaitForSingleObject

```
NtCancelTimer(
NTSTATUS
     NtCancelTimer(
          IN HANDLE hTimer,
          OUT PBOOLEAN pbState
);
```

NtCancelTimer cancels the timer and removes it from the system timer queue.

PARAMETERS

hTimer	Handle to the timer object.
pbState	Pointer to the variable that receives the state of the timer at the time of cancellation. Receives TRUE if the timer is signaled and FALSE if the timer is not signaled.

RETURN VALUE
Returns STATUS_SUCCESS on success and an appropriate error code on failure.

COMMENTS
The Equivalent Win32 API function does not return the state of the timer object, whereas the system service does return the state.

EQUIVALENT WIN32 API
CancelTimer

TimerlnfoBuffer	Pointer to the buffer that receives information about the timer object.
TimerlnfoBufferSize	Size of the buffer (in bytes) pointed to by TimerlnfoBuffer.
BytesCopied	Pointer to the variable that receives the number of bytes copied into TimerlnfoBuffer.

RETURN VALUE

Returns STATUS_SUCCESS on success and an appropriate error code on failure.

COMMENTS

The buffer pointed to by TimerlnfoBuffer must be exactly 16 bytes. The information in the buffer is organized as follows:

```
typedef struct TimerInfo_t {
      LARGE_INTEGER DueTime;
      ULONG TimerState;
      ULONG  Unused;
) TIMERINFO, *PTIMERINFO;
```

TimerState member is 0 if the timer is in not signaled state and 1 if the timer is in signaled state. DueTime is the time remaining for the timer to expire.

EQUIVALENT WIN32 API

None.

NtQueryTimerResolution

```
NTSTATUS
      NtQueryTimerResolution(
            OUT  PULONG  MaxResolution,
            OUT  PULONG  MinResolution.
            OUT  PULONG  SystemResolution
```

NtQueryTimerResolution gets the information about the granularity of the clock interrupt.

PARAMETERS

MaxResolution	Pointer to the variable that receives the maximum resolution (in nanoseconds) supported on the processor.

PARAMETERS

hObject	Handle to the object to wait on.
bAlertable	Flag specifying whether the wait is alertable.
Timeout	Pointer to the variable that contains the timeout for wait.

RETURN VALUE
Returns STATUS_SUCCESS on success and an appropriate error code on failure.

COMMENTS
None.

EQUIVALENT WIN32 API
WaitForSingleObject

NtCancelTimeK

```
NTSTATUS
    NtCancelTimer(
        IN HANDLE hTimer,
        OUT PBOOLEAN pbState
```

NtCancelTimer cancels the timer and removes it from the system timer queue.

PARAMETERS

hTimer	Handle to the timer object.
pbState	Pointer to the variable that receives the state of the timer at the time of cancellation. Receives TRUE if the timer is signaled and FALSE if the timer is not signaled.

RETURN VALUE
Returns STATUS_SUCCESS on success and an appropriate error code on failure.

COMMENTS
The Equivalent Win32 API function does not return the state of the timer object, whereas the system service does return the state.

EQUIVALENT WIN32 API
CancelTimer

NtCreateTimer

```
NTSTATUS
     NtCreateTimer(
          OUT PHANDLE phTimer,
          IN ACCESS_MASK AccessMask.
          IN POBJECT_ATTRIBUTES ObjectAttributes,
          IN TIMER_TYPE TimerType
);
```

NtCreateTimer creates a new timer object.

PARAMETERS

phTimer	Pointer to the variable that receives handle to the timer object.
AccessMask	Type of access requested to the timer object.
ObjectAttributes	Points to the OBJECT_ATTRIBUTES structure containing the information about the timer object to be created, such as name, parent directory, objectflags, and so on.
TimerType	Type of the timer. It can be either NotificationTimer or SynchronizationTimer.

RETURN VALUE

Returns STATUS_SUCCESS on success and an appropriate error code on failure.

COMMENTS

When a timer of type NotificationTimer is signaled, all threads waiting on the timer object are released and the timer remains in signaled state. When a timer of type SynchronizationTimer is signaled, only one thread waiting on the timer object is released. All other threads waiting for the same timer object continue to wait, and the timer object is again set back to not signaled. The Equivalent Win32 API of this system service enables you to create timers of SynchronizationTimer type *only*, whereas the system service enables you to create both types of timers.

EQUIVALENT WIN32 API

CreateWaitableTimer

NtOpenTimer

```
NTSTATUS                                   r,
     NtOpenTimer(
          OUT PHANDLE phTimer,
          IN ACCESS_MASK AccessMask,
```

```
        IN POBJECT_ATTRIBUTES ObjectAttributes
);
```

NtOpenTimer opens handle to an existing named timer object.

PARAMETERS

phTimer	Pointer to the variable that receives handle to the timer object.
AccessMask	Type of access requested to the timer object.
ObjectAttributes	Points to the OBJECT_ATTRIBUTES structure containing the information about the timer object to be opened, such as name, parent directory, objectflags, and so on.

RETURN VALUE
Returns STATUS_SUCCESS on success and an appropriate error code on failure.

COMMENTS
None.

EQUIVALENT WIN32 API
OpenWaitableTimer

NtQueryTimer

```
NTSTATUS
    NtQueryTimer(
        IN HANDLE hTimer,
        IN TIMER_INFO_CLASS InfoClass.
        OUT PVOID TimerInfoBuffer,
        IN ULONG TimerInfoBufferSize,
        OUT PULONG BytesCopied
);
```

NtQueryTimer queries the state information about the timer object.

PARAMETERS

hTimer	Handle to the timer object.
InfoClass	Information class requested for the timer object. This should be 0.

MinResolution	Pointer to the variable that receives the minimum resolution (in nanoseconds) supported on the processor.
SystemResolution	Pointer to the variable that receives resolution (in nanoseconds) currently used by the system.

RETURN VALUE
Returns STATUS_SUCCESS on success and an appropriate error code on failure.

COMMENTS
None.

EQUIVALENT **WIN32 API**
None.

NtSetTimer
NTSTATUS

```
NtSetTimer(
      IN HANDLE hTimer,
      IN PLARGE_INTEGER pDueTime,
      IN PTIMERAPCROUTINE pfnCompletionRoutine OPTIONAL,
      IN DWORD pfnCompletionRoutineArg OPTIONAL,
      IN BOOLEAN bResume,
      IN LONG Period.
      OUT PBOOLEAN bTimerState
```

NtSetTimer activates the timer specified by hTimer.

PARAMETERS

hTimer	Handle to the timer object.
DueTime	Time at which the timer will be set to signaled. Positive values indicate absolute time. Negative values represent time relative to the current system time. The due time is specified in terms of 100ns units.
pfnCompletionRoutine	Pointer to the function that will be called when the timer expires. The completion routine should be defined according to following prototype. This parameter is optional and can be NULL.

```
VOID
```

```
(APIENTRY *PTIMERAPCROUTINE)(
   LPVOID lpArgToCompletionRoutine,
   DWORD dwTimerLowValue,
   DWORD dwTimerHighValue
);
```

pfnCompletionRoutineArg	Optional argument that will be passed to pfnCompletionRoutine. This parameter can be NULL.
bResume	Flag specifying whether to set the system in suspended power conservation mode when the timer expires. This parameter is ignored if the platform does not support this feature.
Period	Specifies the time in milliseconds by which the timer will be reactivated once the timer elapses. If this parameter is 0, the timer is signaled only once.
bTimerState	Pointer to the variable that receives the present state of the timer (TRUE for signaled and FALSE for not signaled).

RETURN VALUE
Returns STATUS_SUCCESS on success and an appropriate error code on failure.

COMMENTS
The Equivalent Win32 API of the call does not return the present state of the timer, whereas the system service does return this information.

EQUIVALENT WIN32 API
SetWaitableTimer

```
NtSetTimerResolution
NTSTATUS
     NtSetTimerResolution(
          IN ULONG NewResolution,
          IN BOOLEAN bSet,
          OUT PULONG pResolutionSet
)f
```

NtSetTimerResolution changes the timer resolution for the clock interrupt.

PARAMETERS

NewResolution	Newly requested resolution for the timer in units of 100ns. The acceptable values for X86 platforms are between 1 and 10 milliseconds.
bSet	Flag specifying whether to set new resolution (TRUE) or to restore previously set resolution (FALSE). The NewResolution parameter is ignored if this parameter is FALSE.
pResolutionSet	Pointer to the variable that receives the timer resolution set by the system.

RETURN VALUE
Returns STATUS_SUCCESS on success and an appropriate error code on failure.

COMMENTS
None.

EQUIVALENT WIN32 API
None.

NtQueryPerformanceCounter

```
NTSTATUS
    NtQueryPerformanceCounter(
        OUT PLARGE_INTEGER pPerformanceCount.
        OUTPLARGE_INTEGERpFrequency                    v  -
):
```

NtQueryPerformanceCounter retrieves the current value and frequency of the high-resolution performance counter if it exists.

PARAMETERS

pPerformanceCount	Pointer to the variable that receives the current performance counter value. If the hardware does not support a high-resolution performance counter, the value will be set to 0.
pFrequency	Pointer to the variable that receives the frequency of the high-resolution performance counter per second. If the hardware does not support a high-resolution performance counter, the value will be set to 0.

RETURN VALUE
Returns STATUS_SUCCESS on success and an appropriate error code on failure.

COMMENTS
None.

EQUIVALENT WIN32 API
QueryPerformanceCounter, QueryPerformanceFrequency

```
NtQuerySytemTime
NTSTATUS
      NtQuerySystemTime(
            OUT PLARGE_INTEGER pSystemTime
);
```

NtQuerySystemTime retrieves the number of 100 nanosecond intervals elapsed since January 1, 1601.

PARAMETERS

pSystemTime	Pointer to the variable that receives the number of 100 nanosecond intervals elapsed since January 1, 1601. The time is expressed as GMT.

RETURN VALUE
Returns STATUS_SUCCESS on success and an appropriate error code on failure.

COMMENTS
None.

EQUIVALENT WIN32 API
GetSystemTime

```
NtSetSytemTime
NTSTATUS
      NtSetSystemTime(
            IN PLARGE_INTEGER pNewSystemTime,
            OUT  PLARGE_INTEGER pOldsystemTime  OPTIONAL
);
```

NtSetSystemTime sets the system time.

PARAMETERS

pNewSystemTime	Pointer to the variable that contains the system time expressed in GMT.
pOldSystemTime	Pointer to the variable that receives the present system time in GMT. This parameter is optional.

RETURN VALUE
Returns STATUS_SUCCESS on success and an appropriate error code on failure.

COMMENTS
None.

EQUIVALENT WIN32 API
SetSystemTime

```
NtGetTickCount
ULONG
     NtGetTickCount(
);
```

NtGetTickCount returns the number of milliseconds that have elapsed since Windows started.

PARAMETERS
None.

RETURN VALUE
Returns the milliseconds elapsed.

COMMENTS
None.

EQUIVALENT WIN32 API
GetTickCount

```
NtAddAtom
NTSTATUS
     NtAddAtom(
          IN PWCHAR pStnng,
          IN ULONG pStringLength,
          OUT PATOM pAtom
):
```