Summary of Kernel-Mode Support Routines

1.1.1 Hardware Configuration

IoGetDeviceProperty
Retrieves device setup information from the registry. Use this routine, rather than accessing the registry directly, to insulate a driver from differences across platforms and from possible changes in the registry structure.

IoReportDetectedDevice
Reports a nonPnP device to the PnP Manager.

IoReportResourceForDetection
Claims hardware resources in the configuration registry for a legacy device. This routine is for drivers that detect legacy hardware which cannot be enumerated by PnP.

IoGetDmaAdapter
Returns a pointer to the DMA adapter structure that represents either the DMA channel to which a device is connected or the driver's busmaster adapter.

IoGetConfigurationInformation
Returns a pointer to the I/O Manager's configuration information structure, which indicates the number of disk, floppy, CD-ROM, tape, SCSI HBAs, serial, and parallel device objects that have already been named by previously loaded drivers, as well as whether certain address ranges have been claimed by "AT" disk-type drivers.

HalExamineMBR
Returns data from the master boot record (MBR) of a disk.

IoReadPartitionTable
Returns a list of partitions on a disk with a given sector size.

IoInvalidateDeviceRelations
Notifies the PnP manager that the relations for a device have changed. The types of device relations include bus relations, ejection relations, removal relations, and the target device relation.

IoInvalidateDeviceState
Notifies the PnP manager that some aspect of the PnP state of a device has changed. In response, the PnP Manager sends an IRP_MN_QUERY_PNP_DEVICE_STATE to the device stack.

IoRegisterPlugPlayNotification
Registers a driver callback routine to be called when a PnP event of the specified category occurs.

IoUnregisterPlugPlayNotification
Removes the registration of a driver's callback routine for a PnP event.

IoRequestDeviceEject
Notifies the PnP Manager that the device eject button was pressed. This routine reports a request for device eject, not media eject.

IoReportTargetDeviceChange
Notifies the PnP Manager that a custom event has occurred on a device. The PnP Manager sends notification of the event to drivers that registered for notification on the device.

1.1.2 Registry

IoGetDeviceProperty
Retrieves device setup information from the registry. Use this routine, rather than accessing the registry directly, to insulate a driver from differences across platforms and from possible changes in the registry structure.

IoOpenDeviceInterfaceRegistryKey
Returns a handle to a registry key for storing information about a particular device interface.

IoOpenDeviceRegistryKey
Returns a handle to a device-specific or a driver-specific registry key for a particular device instance.

IoRegisterDeviceInterface
Registers device functionality (a device interface) that a driver will enable for use by applications or other system components. The I/O Manager creates a registry key for the device interface. Drivers can access persistent storage under this key using **IoOpenDeviceInterfaceRegistryKey**.

IoSetDeviceInterfaceState
Enables or disables a previously registered device interface. Applications and other system components can open only interfaces that are enabled.

RtlCheckRegistryKey
Returns STATUS_SUCCESS if a key exists in the registry along the given relative path.

RtlCreateRegistryKey
Adds a key object in the registry along the given relative path.

RtlQueryRegistryValues
Gives the driver-supplied QueryRegistry callback (read only) access to the entries for the specified value name along the specified relative path in the registry after the QueryRegistry routine is given control.

RtlWriteRegistryValue
Writes caller-supplied data into the registry along the specified relative path at the given value name.

RtlDeleteRegistryValue
Removes the specified value name (and the associated value entries) from the registry along the given relative path.

InitializeObjectAttributes
Sets up a parameter of type OBJECT_ATTRIBUTES for a subsequent call to a **ZwCreate***Xxx* or **ZwOpen***Xxx* routine.

ZwCreateKey
Creates a new key in the registry with the given object's attributes, allowed access, and creation options (such as whether the key is created again when the system is booted). Alternatively, opens an existing key and returns a handle for the key object.

ZwOpenKey
Returns a handle for a key in the registry given the object's attributes (which must include a name for the key) and the desired access to the object.

ZwQueryKey
Returns information about the class of a key, and the number and sizes of its subkeys. This information includes, for example, the length of subkey names and the size of value entries.

ZwEnumerateKey
Returns the specified information about the subkeys of an opened key in the registry.

ZwEnumerateValueKey
Returns the specified information about the value entry, as selected by a zero-based index, of an opened key in the registry.

ZwQueryValueKey
Returns the value entry, as selected by a czero-based index, for an opened key in the registry.

ZwSetValueKey
Replaces (or creates) a value entry for an opened key in the registry.

ZwFlushKey
Forces changes made by **ZwCreateKey** or **ZwSetValueKey** for the opened key object to be written to disk.

ZwDeleteKey
Removes a key and its value entries from the registry as soon as the key is closed.

ZwClose
Releases the handle for an opened object, causing the handle to become invalid and decrementing the reference count of the object handle.

1.1.3 Standard Driver Routines

IoRegisterDriverReinitialization
Sets up the driver-supplied Reinitialize routine, together with its context, so that the Reinitialize routine is called after each subsequently loaded driver's **DriverEntry** routine returns control.

IoConnectInterrupt
Registers an ISR and sets up interrupt objects using values supplied in the PnP IRP_MN_START_DEVICE request. Returns a pointer to a set of interrupt objects that must be passed, along with the driver's SynchCritSection entry point, to **KeSynchronizeExecution**.

IoDisconnectInterrupt
Releases a driver's interrupt objects.

IoInitializeDpcRequest
Associates a driver-supplied DpcForIsr routine with a given device object, so that the DpcForIsr can complete interrupt-driven I/O operations.

KeInitializeDpc
Initializes a DPC object, setting up a driver-supplied CustomDpc routine that can be called with a given context.

KeInitializeTimer
Initializes a notification timer object to the Not-Signaled state.

KeInitializeTimerEx
Initializes a notification or synchronization timer object to the Not-Signaled state.

IoInitializeTimer
Associates a timer with the given device object and registers a driver-supplied IoTimer routine for the device object.

MmLockPagableCodeSection
Locks a set of driver routines marked with a special compiler directive into system space. This operation can occur during driver initialization but usually occurs in the driver's DispatchCreate routine.

MmLockPagableDataSection
Locks a named data section, which is marked with a special compiler directive, into system space if that data is used infrequently, predictably, and at an IRQL less than DISPATCH_LEVEL.

MmLockPagableSectionByHandle
Locks a pageable section into system memory using a handle returned from **MmLockPagableCodeSection** or **MmLockPagableDataSection**.

MmUnlockPagableImageSection
Releases a set of driver routines or a set of data that was locked into nonpaged system space when the driver is no longer processing IRPs.

MmPageEntireDriver
Allows a driver to page out all of its code and data, regardless of the attributes of the various sections in the driver's image.

MmResetDriverPaging
Resets a driver's pageable status to that specified by the sections which make up the driver's image.

1.1.4 Objects and Resources

IoCreateDevice
Initializes a device object, which represents a physical, virtual, or logical device for which the driver is being loaded into the system. Then it allocates space for the driver-defined device extension associated with the device object.

IoDeleteDevice
Removes a device object from the system when the underlying device is removed from the system.

IoGetDeviceObjectPointer
Requests access to a named device object and returns a pointer that device object if the requested access is granted. Also returns a pointer to the file object referenced by the named device object. In effect, this routine establishes a connection between the caller and the next-lower-level driver.

IoAttachDeviceToDeviceStack
Attaches the caller's device object to the highest device object in a chain of drivers and returns a pointer to the previously highest device object. I/O requests bound for the target device are routed first to the caller.

IoGetAttachedDeviceReference
Returns a pointer to the highest level device object in a driver stack and increments the reference count on that object.

IoDetachDevice
Releases an attachment between the caller's device object and a target driver's device object.

IoAllocateDriverObjectExtension
Allocates a per-driver context area with a given unique identifier.

IoGetDriverObjectExtension
Retrieves a previously allocated per-driver context area.

IoRegisterDeviceInterface
Registers device functionality (a device interface) that a driver will enable for use by applications or other system components. The I/O Manager creates a registry key for the device interface. Drivers can access persistent storage under this key using **IoOpenDeviceInterfaceRegistryKey**.

IoIsWdmVersionAvailable
Checks whether a given WDM version is supported by the operating system.

IoDeleteSymbolicLink
Releases a symbolic link between a device object name and a user-visible name.

IoAssignArcName
Sets up a symbolic link between a named device object (such as a tape, floppy, or CD-ROM) and the corresponding ARC name for the device.

IoDeassignArcName
Releases the symbolic link created by calling **IoAssignArcName**.

IoSetShareAccess
Sets the access allowed to a given file object that represents a device. (Only highest-level drivers can call this routine.)

IoConnectInterrupt
Registers a driver's ISR according to the parameters supplied in the IRP_MN_START_DEVICE request. Returns a pointer to a set of allocated, initialized, and connected interrupt objects that is used as an argument to **KeSynchronizeExecution**.

IoDisconnectInterrupt
Releases a driver's interrupt objects when the driver unloads.

IoReadPartitionTable
Returns a list of partitions on a disk with a given sector size.

IoSetPartitionInformation
Sets the partition type and number for a (disk) partition.

IoWritePartitionTable
Writes partition tables for a disk, given the device object that represents the disk, the sector size, and a pointer to a buffer containing the drive layout structure.

IoCreateController
Initializes a controller object that represents a physical device controller which is shared by two or more similar devices that have the same driver, and specifies the size of the controller extension.

IoDeleteController
Removes a controller object from the system.

KeInitializeSpinLock
Initializes a variable of type KSPIN_LOCK.

KeInitializeDpc
Initializes a DPC object, setting up a driver-supplied CustomDpc routine that can be called with a given context.

KeInitializeTimer
Initializes a notification timer object to the Not-Signaled state.

KeInitializeTimerEx
Initializes a notification or synchronization timer object to the Not-Signaled state.

KeInitializeEvent
Initializes an event object as a synchronization (single waiter) or notification (multiple waiters) type event and sets up its initial state (Signaled or Not-Signaled).

ExInitializeFastMutex
Initializes a fast mutex variable that is used to synchronize mutually exclusive access to a shared resource by a set of threads.

KeInitializeMutex
Initializes a mutex object at a given level number as set to the Signaled state.

KeInitializeSemaphore
Initializes a semaphore object to a given count and specifies an upper bound for the count.

IoCreateNotificationEvent
Initializes a named notification event to be used to synchronize access between two or more components. Notification events are not automatically reset.

IoCreateSynchronizationEvent

Initializes a named synchronization event to be used to serialize access to hardware between two otherwise unrelated drivers.

PsCreateSystemThread
Creates a kernel-mode thread that is associated with a given process object or with the default system process. Returns a handle for the thread.

PsTerminateSystemThread
Terminates the current thread and satisfies as many waits as possible for the current thread object.

KeSetBasePriorityThread
Sets up the run-time priority, relative to the system process, for a driver-created thread.

KeSetPriorityThread
Sets up the run-time priority for a driver-created thread with a real-time priority attribute.

MmIsThisAnNtAsSystem
Returns TRUE if the current platform is a server, indicating that more resources are likely to be necessary to process I/O requests than if the machine were a client.

MmQuerySystemSize
Returns an estimate (small, medium, or large) of the amount of memory available on the current platform.

ExInitializeNPagedLookasideList
Initializes a lookaside list of nonpaged memory. After a successful initialization, fixed-size blocks can be allocated from and freed to the lookaside list.

ExInitializePagedLookasideList
Initializes a lookaside list of paged memory. After a successful initialization, fixed-size blocks can be allocated from and freed to the lookaside list.

ExInitializeResourceLite
Initializes a resource, for which the caller provides the storage, to be used for synchronization by a set of threads.

ExReinitializeResourceLite
Reinitializes an existing resource variable.

ExDeleteResourceLite
Deletes a caller-initialized resource from the system's resource list.

ObReferenceObjectByHandle
Returns a pointer to the object body and handle information (attributes and granted access rights), given the handle for an object, the object's type, and a mask. Specifies the desired access to the object and the preferred access mode. A successful call increments the reference count for the object.

ObReferenceObjectByPointer
Increments the reference count for an object so the caller can ensure that the object is not removed from the system while the caller is using it.

ObReferenceObject
Increments the reference count for an object, given a pointer to the object.

ObDereferenceObject
Releases a reference to an object (decrements the reference count), given a pointer to the object body.

RtlInitString
Initializes a counted string in a buffer.

RtlInitAnsiString
Initializes a counted ANSI string in a buffer.

RtlInitUnicodeString
Initializes a counted Unicode string in a buffer.

InitializeObjectAttributes
Initializes a parameter of type OBJECT_ATTRIBUTES for a subsequent call to a **ZwCreate***Xxx* or **ZwOpen***Xxx* routine.

ZwCreateDirectoryObject
Creates or opens a directory object with a specified set of object attributes and requests one or more types of access for the caller. Returns a handle for the directory object.

ZwCreateFile
Creates or opens a file object that represents a physical, logical, or virtual device, a directory, a data file, or a volume. Returns a handle for the file object.

ZwCreateKey
Creates or opens a key object in the registry and returns a handle for the key object.

ZwDeleteKey
Deletes an existing, open key in the registry after the last handle for the key is closed.

ZwMakeTemporaryObject
Resets the "permanent" attribute of an opened object, so that the object and its name can be deleted when the reference count for the object becomes zero.

ZwClose
Releases the handle for an opened object, causing the handle to become invalid, and decrements the reference count of the object handle.

PsGetVersion
Indicates whether the driver is running on a free or checked build of Windows NT/Windows 2000, and optionally supplies information about the operating system version and build number.

ObGetObjectSecurity
Returns a buffered security descriptor for a given object.

ObReleaseObjectSecurity
Releases the security descriptor returned by **ObGetObjectSecurity**.

1.1.5 Initializing Driver-Managed Queues

KeInitializeSpinLock
Initializes a variable of type KSPIN_LOCK. An initialized spin lock is a required parameter to the **Ex..InterlockedList** routines.

InitializeListHead
Sets up a queue header for a driver's internal queue, given a pointer to driver-supplied storage for the queue header and queue.

ExInitializeSListHead
Sets up the queue header for a sequenced, interlocked, singly-linked list.

KeInitializeDeviceQueue
Initializes a device queue object to a Not Busy state, setting up an associated spin lock for multiprocessor-safe access to device queue entries.

1.2.1 Processing IRPs

IoGetCurrentIrpStackLocation
Returns a pointer to the caller's I/O stack location in a given IRP.

IoGetNextIrpStackLocation
Returns a pointer to the next-lower-level driver's I/O stack location in a given IRP.

IoCopyCurrentIrpStackLocationToNext
Copies the IRP stack parameters from the current stack location to the stack location of the next-lower driver and allows the current driver to set an I/O completion routine.

IoSkipCurrentIrpStackLocation
Copies the IRP stack parameters from the current stack location to the stack location of the next-lower driver and does not allow the current driver to set an I/O completion routine.

IoGetRelatedDeviceObject
Returns a pointer to the device object represented by a given file object.

IoGetFunctionCodeFromCtlCode
Returns the value of the function field within a given IOCTL_*XXX* or FSCTL_*XXX*.

IoSetCompletionRoutine
Registers a driver-supplied IoCompletion routine for an IRP, so the IoCompletion routine is called when the next-lower-level driver has completed the requested operation in one or more of the following ways: successfully, with an error, or by canceling the IRP.

IoCallDriver
Sends an IRP to a lower-level driver.

PoCallDriver
Sends an IRP with major function code IRP_MJ_POWER to the next-lower driver.

IoMarkIrpPending

Marks a given IRP indicating that STATUS_PENDING was returned because further processing is required by another driver routine or by a lower-level driver.

IoStartPacket
Calls the driver's StartIo routine with the given IRP for the given device object or inserts the IRP into the device queue if the device is already busy, specifying whether the IRP is cancelable.

IoAcquireCancelSpinLock
Synchronizes cancelable state transitions for IRPs in a multiprocessor-safe manner.

IoSetCancelRoutine
Sets or clears the Cancel routine in an IRP. Setting a Cancel routine makes an IRP cancelable.

IoReleaseCancelSpinLock
Releases the cancel spin lock when the driver has changed the cancelable state of an IRP or releases the cancel spin lock from the driver's Cancel routine.

IoCancelIrp
Marks an IRP as canceled.

IoReadPartitionTable
Returns a list of partitions on a disk with a given sector size.

IoSetPartitionInformation
Sets the partition type and number for a (disk) partition.

IoWritePartitionTable
Writes partition tables for a disk, given the device object representing the disk, the sector size, and a pointer to a buffer containing the drive geometry.

IoAllocateErrorLogEntry
Allocates and initializes an error log packet; returns a pointer so that the caller can supply error-log data and call **IoWriteErrorLogEntry** with the packet.

IoWriteErrorLogEntry
Queues a previously allocated and filled-in error log packet to the system error logging thread.

IoIsErrorUserInduced
Returns a Boolean value indicating whether an I/O request failed due to one of the following conditions: STATUS_IO_TIMEOUT, STATUS_DEVICE_NOT_READY, STATUS_UNRECOGNIZED_MEDIA, STATUS_VERIFY_REQUIRED, STATUS_WRONG_VOLUME, STATUS_MEDIA_WRITE_PROTECTED, or STATUS_NO_MEDIA_IN_DEVICE. If the result is TRUE, a removable-media driver must call **IoSetHardErrorOrVerifyDevice** before completing the IRP.

IoSetHardErrorOrVerifyDevice
Supplies the device object for which the given IRP was failed due to a user-induced error, such as supplying the incorrect media for the requested operation or changing the media before the requested operation was completed. A file system driver uses the associated device object to notify the user, who can then correct the error or retry the operation.

IoGetDeviceToVerify
Returns a pointer to the device object, representing a removable-media device that is the target of the given thread's I/O request. Useful only to file systems or other highest-level drivers.

IoRaiseHardError
Notifies the user that the given IRP was failed on the given device object for an optional VPB, so that the user can correct the error or retry the operation.

IoRaiseInformationalHardError
Notifies the user of an error, providing an I/O error status and an optional string supplying more information.

ExRaiseStatus
Raises an error status and causes a caller-supplied structured exception handler to be called. Useful only to highest-level drivers that supply exception handlers, in particular to file systems.

IoStartNextPacket
Dequeues the next IRP for a given device object, specifies whether the IRP is cancelable, and calls the driver's StartIo routine.

IoStartNextPacketByKey
Dequeues the next IRP for a device object according to a specified sort-key value, specifies whether the IRP is cancelable, and calls the driver's StartIo routine.

IoCompleteRequest
Completes an I/O request, giving a priority boost to the original caller and returning a given IRP to the I/O system for disposal: either to call any IoCompletion routines supplied by higher-level drivers, or to return status to the original requestor of the operation.

IoGetCurrentProcess
Returns a pointer to the current process. Useful only to highest-level drivers.

IoGetInitialStack
Returns the initial base address of the current thread's stack. Useful only to highest-level drivers.

IoGetRemainingStackSize
Returns the amount of available stack space. Useful only to highest-level drivers.

IoGetStackLimits
Returns the boundaries of the current thread's stack frame. Useful only to highest-level drivers.

1.2.2 Driver-Allocated IRPs

IoBuildAsynchronousFsdRequest
Allocates and sets up an IRP that specifies a major function code (IRP_MJ_PNP, IRP_MJ_READ, IRP_MJ_WRITE, IRP_MJ_SHUTDOWN, or IRP_MJ_FLUSH_BUFFERS) with a pointer to:

- The lower driver's device object on which the I/O should occur
- A pointer to a buffer which will contain the data to be read or which contains the data to be written
- The length of the buffer in bytes
- The starting offset on the media
- The I/O status block where the called driver can return status information and the caller's IoCompletion routine can access it.

Returns a pointer to the IRP so the caller can set any necessary minor function code and set up its IoCompletion routine before sending the IRP to the target driver.

IoBuildSynchronousFsdRequest
Allocates and sets up an IRP specifying a major function code (IRP_MJ_PNP, IRP_MJ_READ, IRP_MJ_WRITE, IRP_MJ_SHUTDOWN, or IRP_MJ_FLUSH_BUFFERS) with a pointer to:

- The lower driver's device object on which the I/O should occur;
- A buffer which will contain the data to be read or which contains the data to be written
- The length of the buffer in bytes,
- The starting offset on the media;
- An event object to be set to the Signaled state when the requested operation completes
- The I/O status block where the called driver can return status information and the caller's IoCompletion routine can access it.

Returns a pointer to the IRP so the caller can set any necessary minor function code and set up its IoCompletion routine before sending the IRP to the target driver.

IoBuildDeviceIoControlRequest
Allocates and sets up an IRP specifying a major function code (either IRP_MJ_INTERNAL_DEVICE_CONTROL or IRP_MJ_DEVICE_CONTROL) with an optional input or output buffer; a pointer to the lower driver's device object; an event to be set to the Signaled state when the requested operation completes; and an I/O status block to be set by the driver that receives the IRP. Returns a pointer to the IRP so the caller can set the appropriate IOCTL_*XXX* before sending the IRP to the next-lower-level driver.

PoRequestPowerIrp
Allocates and initializes an IRP with major function code IRP_MJ_POWER and then sends the IRP to the top-level driver in the device stack for the specified device object.

IoSizeOfIrp
Returns the size in bytes required for an IRP with a given count of I/O stack locations.

IoAllocateIrp

Allocates an IRP, given the number of I/O stack locations (optionally, for the caller, but at least one for each driver layered under the caller) and whether to charge quota against the caller. Returns a pointer to an IRP in nonpaged system space if successful; otherwise, returns NULL.

IoInitializeIrp
Initializes an IRP, given a pointer to an already allocated IRP, its length in bytes, and its number of I/O stack locations.

IoSetNextIrpStackLocation
Sets the current IRP stack location to the caller's location in an IRP. The stack location must have been allocated by a preceding call to **IoAllocateIrp** that specified a stack-size argument large enough to give the caller its own stack location.

IoAllocateMdl
Allocates an MDL large enough to map the starting address and length supplied by the caller; optionally associates the MDL with a given IRP.

IoBuildPartialMdl
Builds an MDL for the specified starting virtual address and length in bytes from a given source MDL. Drivers that split large transfer requests into a number of smaller transfers can call this routine.

IoFreeMdl
Releases a given MDL allocated by the caller.

IoMakeAssociatedIrp
Allocates and initializes an IRP to be associated with a master IRP sent to the highest-level driver, allowing the driver to "split" the original request and send associated IRPs on to lower-level drivers or to the device.

IoSetCompletionRoutine
Registers a driver-supplied IoCompletion routine with a given IRP, so that the IoCompletion routine is called when lower-level drivers have completed the request. The IoCompletion routine lets the caller release the IRP it allocated with **IoAllocateIrp** or **IoBuildAsynchronousFsdRequest**; to release any other resources it allocated to set up an IRP for lower drivers; and to perform any I/O completion processing necessary.

IoCallDriver
Sends an IRP to a lower-level driver.

IoFreeIrp
Releases an IRP that was allocated by the caller.

IoReuseIrp
Reinitializes for reuse an IRP that was previously allocated by **IoAllocateIrp**.

1.2.3 File Objects

InitializeObjectAttributes
Initializes a parameter of type OBJECT_ATTRIBUTES for a subsequent call to a **ZwCreate***Xxx* or **ZwOpen***Xxx* routine.

ZwCreateFile
Creates or opens a file object representing a physical, logical, or virtual device, a directory, a data file, or a volume.

ZwQueryInformationFile
Returns information about the state or attributes of an open file.

IoGetFileObjectGenericMapping
Returns information about the mapping between generic access rights and specific access rights for file objects.

ZwReadFile
Returns data from an open file.

ZwSetInformationFile
Changes information about the state or attributes of an open file.

ZwWriteFile
Transfers data to an open file.

ZwClose
Releases the handle for an opened object, causing the handle to become invalid and decrementing the reference count of the object handle.

1.3.1 Driver Routines and I/O Objects

KeSynchronizeExecution
Synchronizes the execution of a driver-supplied SynchCritSection routine with that of the ISR associated with a set of interrupt objects, given a pointer to the interrupt objects.

IoRequestDpc
Queues a driver-supplied DpcForIsr routine to complete interrupt-driven I/O processing at a lower IRQL.

KeInsertQueueDpc
Queues a DPC to be executed as soon as the IRQL of a processor drops below DISPATCH_LEVEL; returns FALSE if the DPC object is already queued.

KeRemoveQueueDpc
Removes a given DPC object from the DPC queue; returns FALSE if the object is not in the queue.

KeSetImportanceDpc
Controls how a particular DCP is queued and, to some degree, how soon the DPC routine is run.

KeSetTargetProcessorDpc
Controls on which processor a particular DCP subsequently will be queued.

AllocateAdapterChannel
Connects a device object to an adapter object and calls a driver-supplied AdapterControl routine to carry out an I/O operation through the system DMA controller or a busmaster adapter as soon as the appropriate DMA channel and any necessary map registers are available. (This routine reserves exclusive access to a DMA channel and map registers for the specified device.)

FreeAdapterChannel
Releases an adapter object, representing a system DMA channel, and optionally releases map registers, if any were allocated.

FreeMapRegisters
Releases a set of map registers that were saved from a call to **AllocateAdapterChannel**, after the registers have been used by **IoMapTransfer** and the busmaster DMA transfer is complete.

IoAllocateController
Connects a device object to a controller object and calls a driver-supplied ControllerControl routine to carry out an I/O operation on the device controller as soon as the controller is not busy. (This routine reserves exclusive access to the hardware controller for the specified device.)

IoFreeController
Releases a controller object, provided that all device operations queued to the controller for the current IRP have completed.

IoStartTimer
Enables the timer for a given device object and calls the driver-supplied IoTimer routine once per second thereafter.

IoStopTimer
Disables the timer for a given device object so that the driver-supplied IoTimer routine is not called unless the driver re-enables the timer.

KeSetTimer
Sets the absolute or relative interval at which a timer object will be set to the Signaled state and optionally supplies a timer DPC to be executed after the interval expires.

KeSetTimerEx
Sets the absolute or relative interval at which a timer object will be set to the Signaled state, optionally supplies a timer DPC to be executed when the interval expires, and optionally supplies a recurring interval for the timer.

KeCancelTimer
Cancels a timer object before the interval passed to **KeSetTimer** expires; dequeues a timer DPC before the timer interval, if any was set, expires.

KeReadStateTimer
Returns whether a given timer object is set to the Signaled state.

IoStartPacket
Calls the driver's StartIo routine with the given IRP for the given device object or inserts the IRP into the device queue if the device is already busy, specifying whether the IRP is cancelable.

IoStartNextPacket

Dequeues the next IRP for a given device object, specifying whether the IRP is cancelable, and calls the driver's StartIo routine.

IoStartNextPacketByKey
Dequeues the next IRP, according to the specified sort-key value, for a given device object. Specifies whether the IRP is cancelable and calls the driver's StartIo routine.

IoSetCompletionRoutine
Registers a driver-supplied IoCompletion routine with a given IRP, so the IoCompletion routine is called when the next-lower-level driver has completed the requested operation in one or more of the following ways: successfully, with an error, or by cancelling the IRP.

IoSetCancelRoutine
Sets or clears the Cancel routine in an IRP. Setting a Cancel routine makes an IRP cancelable.

KeStallExecutionProcessor
Stalls the caller (a device driver) for a given interval on the current processor.

ExAcquireResourceExclusiveLite
Acquires an initialized resource for exclusive access by the calling thread and optionally waits for the resource to be acquired.

ExTryToAcquireResourceExclusiveLite
Acquires a given resource for exclusive access immediately or returns FALSE.

ExAcquireResourceSharedLite
Acquires an initialized resource for shared access by the calling thread and optionally waits for the resource to be acquired.

ExAcquireSharedStarveExclusive
Acquires a given resource for shared access without waiting for any pending attempts to acquire exclusive access to the same resource.

ExAcquireSharedWaitForExclusive
Acquires a given resource for shared access, optionally waiting for any pending exclusive waiters to acquire and release the resource first.

ExReleaseResourceForThreadLite
Releases a given resource that was acquired by the given thread.

ZwReadFile
Reads data from an open file. If the caller opened the file object with certain parameters, the caller can wait on the file handle for completion of the I/O.

ZwWriteFile
Writes data to an open file. If the caller opened the file object with certain parameters, the caller can wait on the file handle for completion of the I/O.

1.3.2 IRQL

KeRaiseIrql
Raises the hardware priority to a given IRQL value, thereby masking off interrupts of equivalent or lower IRQL on the current processor.

KeRaiseIrqlToDpcLevel
Raises the hardware priority to IRQL DISPATCH_LEVEL, thereby masking off interrupts of equivalent or lower IRQL on the current processor.

KeLowerIrql
Restores the IRQL on the current processor to its original value.

KeGetCurrentIrql
Returns the current hardware priority IRQL value.

1.3.3 Spin Locks and Interlocks

IoAcquireCancelSpinLock
Synchronizes cancelable state transitions for IRPs in a multiprocessor-safe manner.

IoSetCancelRoutine
Sets or clears the Cancel routine in an IRP during a cancelable state transition. Setting a Cancel routine makes an IRP cancelable.

IoReleaseCancelSpinLock
Releases the cancel spin lock when the driver has changed the cancelable state of an IRP or releases the cancel spin lock from the driver's Cancel routine.

KeInitializeSpinLock
Initializes a variable of type KSPIN_LOCK, used to synchronize access to data shared among nonISR routines. An initialized spin lock also is a required parameter to the **ExInterlocked***Xxx* routines.

KeAcquireSpinLock
Acquires a spin lock so the caller can synchronize access to shared data safely on multiprocessor platforms.

KeReleaseSpinLock
Releases a spin lock that was acquired by calling **KeAcquireSpinLock** and restores the original IRQL at which the caller was running.

KeAcquireSpinLockAtDpcLevel
Acquires a spin lock, provided that the caller is already running at IRQL DISPATCH_LEVEL.

KeReleaseSpinLockFromDpcLevel
Releases a spin lock that was acquired by calling **KeAcquireSpinLockAtDpcLevel**.

ExInterlocked..List
Insert and remove IRPs in a driver-managed internal queue, which is protected by an initialized spin lock for which the driver provides the storage.

Ke..DeviceQueue
Insert and remove IRPs in a driver-allocated and managed internal device queue object, which is protected by a built-in spin lock.

ExInterlockedAddUlong
Adds a value to a variable of type ULONG as an atomic operation, using a spin lock to ensure multiprocessor-safe access to the variable; returns the value of the variable before the call occurred.

ExInterlockedAddLargeInteger
Adds a value to a variable of type LARGE_INTEGER as an atomic operation, using a spin lock to ensure multiprocessor-safe access to the variable; returns the value of the variable before the call occurred.

InterlockedIncrement
Increments a variable of type LONG as an atomic operation. The sign of the return value is the sign of the result of the operation.

InterlockedDecrement
Decrements a variable of type LONG as an atomic operation. The sign of the return value is the sign of the result of the operation.

InterlockedExchange
Sets a variable of type LONG to a specified value as an atomic operation; returns the value of the variable before the call occurred.

InterlockedExchangeAdd
Adds a value to a given integer variable as an atomic operation; returns the value of the variable before the call occurred.

InterlockedCompareExchange
Compares the values referenced by two pointers. If the values are equal, resets one of the values to a caller-supplied value in an atomic operation.

InterlockedCompareExchangePointer
Compares the pointers referenced by two pointers. If the pointer values are equal, resets one of the values to a caller-supplied value in an atomic operation.

ExInterlockedCompareExchange64
Compares one integer variable to another and, if they are equal, resets the first variable to a caller-supplied ULONGLONG-type value as an atomic operation.

KeGetCurrentProcessorNumber
Returns the current processor number when debugging spin lock usage in SMP machines.

1.3.4 Timers

IoInitializeTimer

Associates a timer with the given device object and registers a driver-supplied IoTimer routine for the device object.

IoStartTimer
Enables the timer for a given device object and calls the driver-supplied IoTimer routine once every second.

IoStopTimer
Disables the timer for a given device object so the driver-supplied IoTimer routine is not called unless the driver re-enables the timer.

KeInitializeDpc
Initializes a DPC object and sets up a driver-supplied CustomTimerDpc routine that can be called with a given context.

KeInitializeTimer
Initializes a notification timer object to the Not-Signaled state.

KeInitializeTimerEx
Initializes a notification or synchronization timer object to the Not-Signaled state.

KeSetTimer
Sets the absolute or relative interval at which a timer object will be set to the Signaled state; optionally supplies a timer DPC to be executed when the interval expires.

KeSetTimerEx
Sets the absolute or relative interval at which a timer object will be set to the Signaled state; optionally supplies a timer DPC to be executed when the interval expires; and optionally supplies a recurring interval for the timer.

KeCancelTimer
Cancels a timer object before the interval passed to **KeSetTimer** expires; dequeues a timer DPC before the timer interval, if any was set, expires.

KeReadStateTimer
Returns TRUE if a given timer object is set to the Signaled state.

KeQuerySystemTime
Returns the current system time.

KeQueryTickCount
Returns the number of interval-timer interrupts that have occurred since the system was booted.

KeQueryTimeIncrement
Returns the number of 100-nanosecond units that are added to the system time at each interval-timer interrupt.

KeQueryInterruptTime
Returns the current value of the system interrupt-time count in 100-nanosecond units.

KeQueryPerformanceCounter
Returns the system performance counter value in hertz.

1.3.5 Driver Threads, Dispatcher Objects, and Resources

KeDelayExecutionThread
Puts the current thread into an alertable or nonalertable wait state for a given interval.

ExInitializeResourceLite
Initializes a resource, for which the caller provides the storage, to be used for synchronization by a set of threads (shared readers, exclusive writers).

ExReinitializeResourceLite
Reinitializes an existing resource variable.

ExAcquireResourceExclusiveLite
Acquires an initialized resource for exclusive access by the calling thread and optionally waits for the resource to be acquired.

ExTryToAcquireResourceExclusiveLite
Either acquires a given resource for exclusive access immediately, or returns FALSE.

ExAcquireResourceSharedLite
Acquires an initialized resource for shared access by the calling thread and optionally waits for the resource to be acquired.

ExAcquireSharedStarveExclusive
Acquires a given resource for shared access without waiting for any pending attempts to acquire exclusive access to the same resource.

ExAcquireSharedWaitForExclusive
Acquires a given resource for shared access, optionally waiting for any pending exclusive waiters to acquire and release the resource first.

ExIsResourceAcquiredExclusiveLite
Returns whether the calling thread has exclusive access to a given resource.

ExIsResourceAcquiredSharedLite
Returns how many times the calling thread has acquired shared access to a given resource.

ExGetExclusiveWaiterCount
Returns the number of threads currently waiting to acquire a given resource for exclusive access.

ExGetSharedWaiterCount
Returns the number of threads currently waiting to acquire a given resource for shared access.

ExConvertExclusiveToSharedLite
Converts a given resource from acquired for exclusive access to acquired for shared access.

ExGetCurrentResourceThread
Returns the thread ID of the current thread.

ExReleaseResourceForThreadLite
Releases a given resource that was acquired by the given thread.

ExDeleteResourceLite
Deletes a caller-initialized resource from the system's resource list.

IoQueueWorkItem
Queues an initialized work queue item so the driver-supplied routine will be called when a system worker thread is given control.

KeSetTimer
Sets the absolute or relative interval at which a timer object will be set to the Signaled state, and optionally supplies a timer DPC to be executed when the interval expires.

KeSetTimerEx
Sets the absolute or relative interval at which a timer object will be set to the Signaled state. Optionally supplies a timer DPC to be executed when the interval expires and a recurring interval for the timer.

KeCancelTimer
Cancels a timer object before the interval passed to **KeSetTimer** expires. Dequeues a timer DPC before the timer interval (if any) expires.

KeReadStateTimer
Returns TRUE if a given timer object is set to the Signaled state.

KeSetEvent
Returns the previous state of a given event object and sets the event (if not already Signaled) to the Signaled state.

KeClearEvent
Resets an event to the Not-Signaled state.

KeResetEvent
Returns the previous state of an event object and resets the event to the Not-Signaled state.

KeReadStateEvent
Returns the current state (nonzero for Signaled or zero for Not-Signaled) of a given event object.

ExAcquireFastMutex
Acquires an initialized fast mutex, possibly after putting the caller into a wait state until it is acquired, and gives the calling thread ownership with APCs disabled.

ExTryToAcquireFastMutex
Acquires the given fast mutex immediately for the caller with APCs disabled, or returns FALSE.

ExReleaseFastMutex
Releases ownership of a fast mutex that was acquired with **ExAcquireFastMutex** or **ExTryToAcquireFastMutex**.

ExAcquireFastMutexUnsafe
Acquires an initialized fast mutex, possibly after putting the caller into a wait state until it is acquired.

ExReleaseFastMutexUnsafe
Releases ownership of a fast mutex that was acquired with **ExAcquireFastMutexUnsafe**.

KeReleaseMutex
Releases a given mutex object, specifying whether the caller will call one of the **KeWait***Xxx* routines as soon as **KeReleaseMutex** returns the previous value of the mutex state (a zero for Signaled; otherwise, Not-Signaled).

KeReadStateMutex

Returns the current state (one for Signaled or any other value for Not-Signaled) of a given mutex object.

KeReleaseSemaphore
Releases a given semaphore object. Supplies a (run-time) priority boost for waiting threads if the release sets the semaphore to the Signaled state. Augments the semaphore count by a given value and specifies whether the caller will call one of the **KeWait***Xxx* routines as soon as **KeReleaseSemaphore** returns.

KeReadStateSemaphore
Returns the current state (zero for Not-Signaled or a positive value for Signaled) of a given semaphore object.

KeWaitForSingleObject
Puts the current thread into an alertable or nonalertable wait state until a given dispatcher object is set to the Signaled state or (optionally) until the wait times out.

KeWaitForMutexObject
Puts the current thread into an alertable or nonalertable wait state until a given mutex is set to the Signaled state or (optionally) until the wait times out.

KeWaitForMultipleObjects
Puts the current thread into an alertable or nonalertable wait state until any one or all of a number of dispatcher objects are set to the Signaled state or (optionally) until the wait times out.

PsGetCurrentThread
Returns a handle for the current thread.

KeGetCurrentThread
Returns a pointer to the opaque thread object that represents the current thread.

IoGetCurrentProcess
Returns a handle for the process of the current thread.

PsGetCurrentProcess
Returns a pointer to the process of the current thread.

KeEnterCriticalRegion
Temporarily disables the delivery of normal kernel APCs while a highest-level driver is running in the context of the user-mode thread that requested the current I/O operation. Special kernel-mode APCs are still delivered.

KeLeaveCriticalRegion
Re-enables, as soon as possible, the delivery of normal kernel-mode APCs that were disabled by a preceding call to **KeEnterCriticalRegion**.

KeSaveFloatingPointState
Saves the current thread's nonvolatile floating-point context so that the caller can carry out its own floating-point operations.

KeRestoreFloatingPointState
Restores the previous nonvolatile floating-point context that was saved with **KeSaveFloatingPointState**.

ZwSetInformationThread
Sets the priority of a given thread for which the caller has a handle.

PsGetCurrentProcessId
Returns the system-assigned identifier of the current process.

PsGetCurrentThreadId
Returns the system-assigned identifier of the current thread.

PsSetCreateProcessNotifyRoutine
Registers a highest level driver's callback that is subsequently notified whenever a new process is created or existing process deleted.

PsSetCreateThreadNotifyRoutine
Registers a highest level driver's callback that is subsequently notified whenever a new thread is created or an existing thread is deleted.

PsSetLoadImageNotifyRoutine
Registers a callback routine for a highest level system-profiling driver. The callback is subsequently notified whenever a new image is loaded for execution.

1.4.1 Buffer Management

ExAllocatePool
Allocates (optionally cache-aligned) memory from paged or nonpaged system space.

ExAllocatePoolWithQuota
Allocates pool memory charging quota against the original requestor of the I/O operation. (Only highest-level drivers can call this routine.)

ExAllocatePoolWithTag
Allocates (optionally cache-aligned) tagged memory from paged or nonpaged system space. The caller-supplied tag is put into any crash dump of memory that occurs.

ExAllocatePoolWithQuotaTag
Allocates tagged pool memory charging quota against the original requestor of the I/O operation. The caller-supplied tag is put into any crash dump of memory that occurs. Only highest-level drivers can call this routine.

ExFreePool
Releases memory to paged or nonpaged system space.

ExInitializeNPagedLookasideList
Initializes a lookaside list of nonpaged memory. After successful initialization of the list, fixed-size blocks can be allocated from, and freed to, the lookaside list.

ExAllocateFromNPagedLookasideList
Removes the first entry from the specified lookaside list in nonpaged memory. If the lookaside list is empty, allocates an entry from nonpaged pool.

ExFreeToNPagedLookasideList
Returns an entry to the specified lookaside list in nonpaged memory. If the list has reached its maximum size, returns the entry to nonpaged pool.

ExDeleteNPagedLookasideList
Deletes a nonpaged lookaside list.

ExInitializePagedLookasideList
Initializes a lookaside list of paged memory. After successful initialization of the list, fixed-size blocks can be allocated from and freed to the lookaside list.

ExAllocateFromPagedLookasideList
Removes the first entry from the specified lookaside list in paged memory. If the lookaside list is empty, allocates an entry from paged pool.

ExFreeToPagedLookasideList
Returns an entry to the specified lookaside list in paged memory. If the list has reached its maximum size, returns the entry to paged pool.

ExDeletePagedLookasideList
Deletes a paged lookaside list.

MmQuerySystemSize
Returns an estimate (small, medium, or large) of the amount of memory available on the current platform.

MmIsThisAnNtAsSystem
Returns TRUE if the machine is running as a Windows NT/Windows 2000 server. If this routine returns TRUE, the caller is likely to require more resources to process I/O requests, and the machine is a server so it is likely to have more resources available.

1.4.2 Long-Term Internal Driver Buffers

MmAllocateContiguousMemory
Allocates a range of physically contiguous, cache-aligned memory in nonpaged pool.

MmFreeContiguousMemory
Releases a range of physically contiguous memory when the driver unloads.

MmAllocateNonCachedMemory
Allocates a virtual address range of noncached and cache-aligned memory in nonpaged system space (pool).

MmFreeNonCachedMemory
Releases a virtual address range of noncached memory in nonpaged system space when the driver unloads.

AllocateCommonBuffer
Allocates and maps a logically contiguous region of memory that is simultaneously accessible both from the processor and from a device, given access to an adapter object, the requested length of the

memory region to allocate, and access to variables where the starting logical and virtual addresses of the allocated region are returned. Returns TRUE if the requested length was allocated. Can be used for continuous busmaster DMA or for system DMA using the autoinitialize mode of a system DMA controller.

FreeCommonBuffer
Releases an allocated common buffer and unmaps it, given access to the adapter object, the length, and the starting logical and virtual addresses of the region to be freed when the driver unloads. Arguments must match those passed in the call to **AllocateCommonBuffer**.

1.4.3 Buffered Data and Buffer Initialization

RtlCompareMemory
Compares data, given pointers to caller-supplied buffers and the length in bytes for the comparison. Returns the number of bytes that are equal.

RtlCopyMemory
Copies the data from one caller-supplied buffer to another, given pointers to both buffers and the length in bytes to be copied.

RtlMoveMemory
Copies the data from one caller-supplied memory range to another, given pointers to the base of both ranges and the length in bytes to be copied.

RtlFillMemory
Fills a caller-supplied buffer with the specified UCHAR value, given a pointer to the buffer and the length in bytes to be filled.

RtlZeroMemory
Fills a buffer with zeros, given a pointer to the caller-supplied buffer and the length in bytes to be filled.

RtlStoreUshort
Stores a USHORT value at a given address, avoiding alignment faults.

RtlRetrieveUshort
Retrieves a USHORT value at a given address, avoiding alignment faults, and stores the value at a given address, that is assumed to be aligned.

RtlStoreUlong
Stores a ULONG value at a given address, avoiding alignment faults.

RtlRetrieveUlong
Retrieves a ULONG value at a given address, avoiding alignment faults, and stores the value at a given address, that is assumed to be aligned.

1.4.4 Address Mappings and MDLs

MmGetPhysicalAddress
Returns the corresponding physical address for a given valid virtual address.

MmGetMdlVirtualAddress
Returns a (possibly invalid) virtual address for a buffer described by a given MDL; the returned address, used as an index to a physical address entry in the MDL, can be input to **MapTransfer** for drivers that use DMA.

MmGetSystemAddressForMdl
Returns a system-space virtual address that maps the physical pages described by a given MDL for drivers whose devices must use PIO. If no virtual address exists, one is assigned. If none are available, a bug check is issued. Windows 2000 drivers should use **MmGetSystemAddressForMdlSafe** instead.

MmGetSystemAddressForMdlSafe
Returns a system-space virtual address that maps the physical pages described by a given MDL for drivers whose devices must use PIO. If no virtual address exists, one is assigned.

MmBuildMdlForNonPagedPool
Fills in the corresponding physical addresses of a given MDL that specifies a range of virtual addresses in nonpaged pool.

MmGetMdlByteCount
Returns the length in bytes of the buffer mapped by a given MDL.

MmGetMdlByteOffset
Returns the byte offset within a page of the buffer described by a given MDL.

MmMapLockedPages
Maps already locked physical pages, described by a given MDL, to a returned virtual address range.

MmUnmapLockedPages
Releases a mapping set up by **MmMapLockedPages**.

MmIsAddressValid
Returns whether a page fault will occur if a read or write operation is done at the given virtual address.

MmSizeOfMdl
Returns the number of bytes required for an MDL describing the buffer specified by the given virtual address and length in bytes.

MmCreateMdl
Allocates and initializes an MDL describing a buffer specified by the given virtual address and length in bytes; returns a pointer to the MDL.

MmPrepareMdlForReuse
Reinitializes a caller-created MDL for reuse.

MmInitializeMdl
Initializes a caller-created MDL to describe a buffer specified by the given virtual address and length in bytes.

MmMapIoSpace
Maps a physical address range to a cached or noncached virtual address range in nonpaged system space.

MmUnmapIoSpace
Unmaps a virtual address range from a physical address range.

MmProbeAndLockPages
Probes the pages specified in an MDL for a particular kind of access, makes the pages resident, and locks them in memory; returns the MDL updated with corresponding physical addresses. (Usually, only highest-level drivers call this routine.)

MmUnlockPages
Unlocks the previously probed and locked pages specified in an MDL.

IoAllocateMdl
Allocates an MDL large enough to map the starting address and length supplied by the caller; optionally associates the MDL with a given IRP.

IoBuildPartialMdl
Builds an MDL for the specified starting virtual address and length in bytes from a given source MDL. Drivers that split large transfer requests into a number of smaller transfers can call this routine.

IoFreeMdl
Releases a given MDL allocated by the caller.

1.4.5 Buffer and MDL Management

ADDRESS_AND_SIZE_TO_SPAN_PAGES
Returns the number of pages required to contain a given virtual address and size in bytes.

BYTE_OFFSET
Returns the byte offset of a given virtual address within the page.

BYTES_TO_PAGES
Returns the number of pages necessary to contain a given number of bytes.

PAGE_ALIGN
Returns the page-aligned virtual address for the page that contains a given virtual address.

ROUND_TO_PAGES
Rounds a given size in bytes up to a page-size multiple.

1.4.6 Device Memory Access

For the following, *XXX*_**REGISTER**_*XXX* indicates device memory that is mapped onto system space, while *XXX*_**PORT**_*XXX* indicates device memory in I/O space.

READ_PORT_UCHAR
Reads a UCHAR value from the given I/O port address.

READ_PORT_USHORT
Reads a USHORT value from the given I/O port address.

READ_PORT_ULONG
Reads a ULONG value from the given I/O port address.

READ_PORT_BUFFER_UCHAR
Reads a given count of UCHAR values from the given I/O port into a given buffer.

READ_PORT_BUFFER_USHORT
Reads a given count of USHORT values from the given I/O port into a given buffer.

READ_PORT_BUFFER_ULONG
Reads a given count of ULONG values from the given I/O port into a given buffer.

WRITE_PORT_UCHAR
Writes a given UCHAR value to the given I/O port address.

WRITE_PORT_USHORT
Writes a given USHORT value to the given I/O port address.

WRITE_PORT_ULONG
Writes a given ULONG value to the given I/O port address.

WRITE_PORT_BUFFER_UCHAR
Writes a given count of UCHAR values from a given buffer to the given I/O port.

WRITE_PORT_BUFFER_USHORT
Writes a given count of USHORT values from a given buffer to the given I/O port.

WRITE_PORT_BUFFER_ULONG
Writes a given count of ULONG values from a given buffer to the given I/O port.

READ_REGISTER_UCHAR
Reads a UCHAR value from the given register address in memory space.

READ_REGISTER_USHORT
Reads a USHORT value from the given register address in memory space.

READ_REGISTER_ULONG
Reads a ULONG value from the given register address in memory space.

READ_REGISTER_BUFFER_UCHAR
Reads a given count of UCHAR values from the given register address into the given buffer.

READ_REGISTER_BUFFER_USHORT
Reads a given count of USHORT values from the given register address into the given buffer.

READ_REGISTER_BUFFER_ULONG
Reads a given count of ULONG values from the given register address into the given buffer.

WRITE_REGISTER_UCHAR
Writes a given UCHAR value to the given register address in memory space.

WRITE_REGISTER_USHORT
Writes a given USHORT value to the given register address in memory space.

WRITE_REGISTER_ULONG
Writes a given ULONG value to the given register address in memory space.

WRITE_REGISTER_BUFFER_UCHAR
Writes a given count of UCHAR values from a given buffer to the given register address.

WRITE_REGISTER_BUFFER_USHORT
Writes a given count of USHORT values from a given buffer to the given register address.

WRITE_REGISTER_BUFFER_ULONG
Writes a given count of ULONG values from a given buffer to the given register address.

1.4.7 Pageable Drivers

MmLockPagableCodeSection
Locks a set of driver routines marked with a special compiler directive into system space.

MmLockPagableDataSection
Locks data marked with a special compiler directive into system space, when that data is accessed infrequently, predictably, and at an IRQL less than DISPATCH_LEVEL.

MmLockPagableSectionByHandle
Locks a pageable section into system memory using a handle returned from **MmLockPagableCodeSection** or **MmLockPagableDataSection**.

MmUnlockPagableImageSection
Releases a section that was previously locked into system space when the driver is no longer processing IRPs, or when the contents of the section is no longer required.

MmPageEntireDriver
Lets a driver page all of its code and data regardless of the attributes of the various sections in the driver's image.

MmResetDriverPaging
Resets a driver's pageable status to that specified by the sections which make up the driver's image.

1.4.8 Sections and Views

InitializeObjectAttributes
Sets up a parameter of type OBJECT_ATTRIBUTES for a subsequent call to a **ZwCreate***Xxx* or **ZwOpen***Xxx* routine.

ZwOpenSection
Obtains a handle for an existing section, provided that the requested access can be allowed.

ZwMapViewOfSection
Maps a view of an open section into the virtual address space of a process. Returns an offset into the section (base of the mapped view) and the size mapped.

ZwUnMapViewOfSection
Releases a mapped view in the virtual address space of a process.

1.5 DMA

IoGetDmaAdapter
Returns a pointer to an adapter object that represents either the DMA channel to which the driver's device is connected or the driver's busmaster adapter. Also returns the maximum number of map registers the driver can specify for each DMA transfer.

MmGetMdlVirtualAddress
Returns the base virtual address of a buffer described by a given MDL. The returned address, used as an index to a physical address entry in the MDL, can be input to **MapTransfer**.

MmGetSystemAddressForMdlSafe
Returns a nonpaged system-space virtual address for the base of the memory area described by an MDL. It maps the physical pages described by the MDL into system space, if they are not already mapped to system space. WDM drivers should use **MmGetSystemAddressForMdl** instead.

ADDRESS_AND_SIZE_TO_SPAN_PAGES
Returns the number of pages spanned by the virtual range defined by a virtual address and a length in bytes. A driver can use this macro to determine whether a transfer request must be split into partial transfers.

AllocateAdapterChannel
Reserves exclusive access to a DMA channel and map registers for a device. When the channel and registers are available, this routine calls a driver-supplied AdapterControl routine to carry out an I/O operation through either the system DMA controller or a busmaster adapter.

AllocateCommonBuffer
Allocates and maps a logically contiguous region of memory that is simultaneously accessible from both the processor and a device. This routine returns TRUE if the requested length was allocated.

FlushAdapterBuffers
Forces any data remaining in either a busmaster adapter's or the system DMA controller's internal buffers to be written into memory or to the device.

FreeAdapterChannel

Releases an adapter object that represents a system DMA channel, and optionally releases any allocated map registers.

FreeCommonBuffer
Releases and unmaps a previously allocated common buffer. Arguments must match those passed in an earlier call to **AllocateCommonBuffer**.

FreeMapRegisters
Releases a set of map registers that were saved from a call to **AllocateAdapterChannel**. A driver calls this routine after using the registers in one or more calls to **MapTransfer**, flushing the cache by calling **FlushAdapterBuffers**, and completing the busmaster DMA transfer.

GetDmaAlignment
Returns the buffer alignment requirements for a DMA controller or device.

GetScatterGatherList
Prepares the system for scatter/gather DMA for a device and calls a driver-supplied routine to carry out the I/O operation. For devices that support scatter/gather DMA, this routine combines the functionality of **AllocateAdapterChannel** and **MapTransfer**.

KeFlushIoBuffers
Flushes the memory region described by an MDL from all processors' caches into memory.

MapTransfer
Sets up map registers for an adapter object previously allocated by **AllocateAdapterChannel** to map a transfer from a locked-down buffer. Returns the logical address of the mapped region and, for busmaster devices that support scatter/gather, the number of bytes mapped.

PutDmaAdapter
Frees an adapter object previously allocated by **IoGetDmaAdapter**.

PutScatterGatherList
Frees map registers and scatter/gather list previously allocated by **GetScatterGatherList**.

ReadDmaCounter
Returns the number of bytes yet to be transferred during the current system DMA operation (in autoinitialize mode).

1.6 PIO

MmProbeAndLockPages
Probes the pages specified in an MDL for a particular kind of access, makes the pages resident, and locks them in memory; returns the MDL updated with corresponding physical addresses.

MmGetSystemAddressForMdlSafe
Returns a system-space virtual address that maps the physical pages described by a given MDL for drivers whose devices must use PIO. If no virtual address exists, one is assigned. Windows 98 drivers should use MmGetSystemAddressForMdl instead.

KeFlushIoBuffers
Flushes the memory region described by a given MDL from all processors' caches into memory.

MmUnlockPages
Unlocks the previously probed and locked pages specified in an MDL.

MmMapIoSpace
Maps a physical address range to a cached or noncached virtual address range in nonpaged system space.

MmUnmapIoSpace
Unmaps a virtual address range from a physical address range.

1.7 Driver-Managed Queues

KeInitializeSpinLock
Initializes a variable of type KSPIN_LOCK. An initialized spin lock is a required parameter to the **Ex..InterlockedList** routines.

InitializeListHead
Sets up a queue header for a driver's internal queue, given a pointer to driver-supplied storage for the queue header and queue. An initialized queue header is a required parameter to the **ExInterlockedInsert/Remove..List** routines.

ExInterlockedInsertTailList
Inserts an entry at the tail of a doubly-linked list, using a spin lock to ensure multiprocessor-safe access to the list and atomic modification of the list links.

ExInterlockedInsertHeadList
Inserts an entry at the head of a doubly-linked list, using a spin lock to ensure multiprocessor-safe access to the list and atomic modification of the links in the list.

ExInterlockedRemoveHeadList
Removes an entry from the head of a doubly-linked list, using a spin lock to ensure multiprocessor-safe access to the list and atomic modification of the links in the list.

ExInterlockedPopEntryList
Removes an entry from the head of a singly-linked list as an atomic operation, using a spin lock to ensure multiprocessor-safe access to the list.

ExInterlockedPushEntryList
Inserts an entry at the head of a singly-linked list as an atomic operation, using a spin lock to ensure multiprocessor-safe access to the list.

IsListEmpty
Returns TRUE if a queue is empty. (This type of doubly-linked list is not protected by a spin lock, unless the caller explicitly manages synchronization to queued entries with an initialized spin lock for which the caller supplies the storage.)

InsertTailList
Queues an entry at the end of the list.

InsertHeadList
Queues an entry at the head of the list.

RemoveHeadList
Dequeues an entry at the head of the list.

RemoveTailList
Dequeues an entry at the end of the list.

RemoveEntryList
Returns whether a given entry is in the given list and dequeues the entry if it is.

PushEntryList
Inserts an entry into the queue. (This type of singly-linked list is not protected by a spin lock, unless the caller explicitly manages synchronization to queued entries with an initialized spin lock for which the caller supplies the storage.)

PopEntryList
Removes an entry from the queue.

ExInterlockedPopEntrySList
Removes an entry from the head of a sequenced, singly-linked list that was set up with **ExInitializeSListHead**

ExInterlockedPushEntrySList
Queues an entry at the head of a sequenced, singly-linked list that was set up with **ExInitializeSListHead**

ExQueryDepthSList
Returns the number of entries currently queued in a sequenced, singly-linked list.

ExInitializeNPagedLookasideList
Sets up a lookaside list, protected by a system-supplied spin lock, in nonpaged pool from which the driver can allocate and free blocks of a fixed size.

KeInitializeDeviceQueue
Initializes a device queue object to a not-busy state, setting up an associated spin lock for multiprocessor-safe access to device queue entries.

KeInsertDeviceQueue
Acquires the device queue spin lock and queues an entry to a device driver if the device queue is not empty; otherwise, inserts the entry at the tail of the device queue.

KeInsertByKeyDeviceQueue
Acquires the device queue spin lock and queues an entry to a device driver if the device queue is not empty; otherwise, inserts the entry into the queue according to the given sort-key value.

KeRemoveDeviceQueue
Removes an entry from the head of a given device queue.

KeRemoveByKeyDeviceQueue

Removes an entry, selected according to the specified sort-key value, from the given device queue.

KeRemoveEntryDeviceQueue
Determines whether a given entry is in the given device queue and, if so, dequeues the entry.

1.8 Driver System Threads

PsCreateSystemThread
Creates a kernel-mode thread associated with a given process object or with the default system process. Returns a handle for the thread.

PsTerminateSystemThread
Terminates the current thread and satisfies as many waits as possible for the current thread object.

PsGetCurrentThread
Returns a handle for the current thread.

KeGetCurrentThread
Returns a pointer to the opaque thread object that represents the current thread.

KeQueryPriorityThread
Returns the current priority of a given thread.

KeSetBasePriorityThread
Sets up the run-time priority, relative to the system process, for a driver-created thread.

KeSetPriorityThread
Sets up the run-time priority for a driver-created thread with a real-time priority attribute.

KeDelayExecutionThread
Puts the current thread into an alertable or nonalertable wait state for a given interval.

IoQueueWorkItem
Queues an initialized work queue item so the driver-supplied routine will be called when a system worker thread is given control.

ZwSetInformationThread
Sets the priority of a given thread for which the caller has a handle.

1.9 Strings

RtlInitString
Initializes the specified string in a buffer.

RtlInitAnsiString
Initializes the specified ANSI string in a buffer.

RtlInitUnicodeString
Initializes the specified Unicode string in a buffer.

RtlAnsiStringToUnicodeSize
Returns the size in bytes required to hold a Unicode version of a given buffered ANSI string.

RtlAnsiStringToUnicodeString
Converts a buffered ANSI string to a Unicode string, given a pointer to the source-string buffer and the address of caller-supplied storage for a pointer to the destination buffer. (This routine allocates a destination buffer if the caller does not supply the storage.) You can also use the string manipulation routines provided by a compiler to convert ANSI strings to Unicode.

RtlFreeUnicodeString
Releases a buffer containing a Unicode string, given a pointer to the buffer returned by **RtlAnsiStringToUnicodeString**.

RtlUnicodeStringToAnsiString
Converts a buffered Unicode string to an ANSI string, given a pointer to the source-string buffer and the address of caller-supplied storage for a pointer to the destination buffer. (This routine allocates a destination buffer if the caller does not supply the storage.)

RtlFreeAnsiString
Releases a buffer containing an ANSI string, given a pointer to the buffer returned by **RtlUnicodeStringToAnsiString**.

RtlAppendUnicodeStringToString
Concatenates a copy of a buffered Unicode string with a buffered Unicode string, given pointers to both buffers.

RtlAppendUnicodeToString
Concatenates a given input string with a buffered Unicode string, given a pointer to the buffer.

RtlCopyString
Copies the source string to the destination, given pointers to both buffers, or sets the length of the destination string (but not the length of the destination buffer) to zero if the optional pointer to the source-string buffer is NULL.

RtlCopyUnicodeString
Copies the source string to the destination, given pointers to both buffers, or sets the length of the destination string (but not the length of the destination buffer) to zero if the optional pointer to the source-string buffer is NULL.

RtlEqualString
Returns TRUE if the given ANSI alphabetic strings are equivalent.

RtlEqualUnicodeString
Returns TRUE if the given buffered strings are equivalent.

RtlCompareString
Compares two buffered, single-byte character strings and returns a signed value indicating whether they are equivalent or which is greater.

RtlCompareUnicodeString
Compares two buffered Unicode strings and returns a signed value indicating whether they are equivalent or which is greater.

RtlUpperString
Converts a copy of a buffered string to uppercase and stores the copy in a destination buffer.

RtlUpcaseUnicodeString
Converts a copy of a buffered Unicode string to uppercase and stores the copy in a destination buffer.

RtlIntegerToUnicodeString
Converts an unsigned integer value in the specified base to one or more Unicode characters in a buffer.

RtlUnicodeStringToInteger
RtlUnicodeStringToInteger converts the Unicode string representation of an integer into its integer equivalent.

1.10 Data Conversions

InterlockedExchange
Sets a variable of type LONG to a given value as an atomic operation; returns the original value of the variable.

RtlConvertLongToLargeInteger
Converts a given LONG value to a LARGE_INTEGER value.

RtlConvertUlongToLargeInteger
Converts a given ULONG value to a LARGE_INTEGER value.

RtlTimeFieldsToTime
Converts information in a TIME_FIELDS structure to system time.

RtlTimeToTimeFields
Converts a system time value into a buffered TIME_FIELDS value.

ExSystemTimeToLocalTime
Adds the time-zone bias for the current locale to GMT system time, converting it to local time.

ExLocalTimeToSystemTime
Subtracts the time-zone bias from the local time, converting it to GMT system time.

RtlAnsiStringToUnicodeString
Converts a buffered ANSI string to a Unicode string, given a pointer to the source-string buffer and the address of caller-supplied storage for a pointer to the destination buffer. (This routine allocates a destination buffer if the caller does not supply the storage.)

RtlUnicodeStringToAnsiString
Converts a buffered Unicode string to an ANSI string, given a pointer to the source-string buffer and the address of caller-supplied storage for a pointer to the destination buffer. (This routine allocates a destination buffer if the caller does not supply the storage.)

RtlUpperString
Converts a copy of a buffered string to uppercase and stores the copy in a destination buffer.

RtlUpcaseUnicodeString
Converts a copy of a buffered Unicode string to uppercase and stores the copy in a destination buffer.

RtlCharToInteger
Converts a single-byte character value into an integer in the specified base.

RtlIntegerToUnicodeString
Converts an unsigned integer value in the specified base to one or more Unicode characters in the given buffer.

RtlUnicodeStringToInteger
Converts a Unicode string representation of an integer into its integer equivalent.

1.11 Access to Driver-Managed Objects

ExCreateCallback
Creates or opens a callback object.

ExNotifyCallback
Calls the callback routines registered with a previously created or opened callback object.

ExRegisterCallback
Registers a callback routine with a previously created or opened callback object, so that the caller can be notified when conditions defined for the callback occur.

ExUnregisterCallback
Cancels the registration of a callback routine with a callback object.

IoRegisterDeviceInterface
Registers device functionality (a device interface) that a driver can enable for use by applications or other system components.

IoSetDeviceInterfaceState
Enables or disables a previously registered device interface. Applications and other system components can open only interfaces that are enabled.

IoGetDeviceInterfaceAlias
Returns the alias device interface of the specified interface class, if the alias exists. Device interfaces are considered aliases if they are exposed by the same underlying device and have identical interface reference strings, but are of different interface classes.

IoGetDeviceInterfaces
Returns a list of device interfaces of a particular device interface class (such as all devices on the system that support a HID interface).

IoGetFileObjectGenericMapping
Returns information about the mapping between generic access rights and specific access rights for file objects.

IoSetShareAccess
Sets the access allowed to a given file object representing a device. (Only highest-level drivers can call this routine.)

IoCheckShareAccess
Checks whether a request to open a file object specifies a desired access that is compatible with the current shared access permissions for the open file object. (Only highest-level drivers can call this routine.)

IoUpdateShareAccess
Modifies the current share-access permissions on the given file object. (Only highest-level drivers can call this routine.)

IoRemoveShareAccess
Restores the shared-access permissions on the given file object that were modified by a preceding call to **IoUpdateShareAccess**.

RtlLengthSecurityDescriptor
Returns the size in bytes of a given security descriptor.

RtlValidSecurityDescriptor
Returns whether a given security descriptor is valid.

RtlCreateSecurityDescriptor
Initializes a new security descriptor to an absolute format with default values (in effect, with no security constraints).

RtlSetDaclSecurityDescriptor
Sets the discretionary ACL information for a given security descriptor in absolute format.

SeAssignSecurity
Builds a security descriptor for a new object, given the security descriptor of its parent directory (if any) and an originally requested security for the object.

SeDeassignSecurity
Deallocates the memory associated with a security descriptor that was created with **SeAssignSecurity**.

SeValidSecurityDescriptor
Returns whether a given security descriptor is structurally valid.

SeAccessCheck
Returns a Boolean indicating whether the requested access rights can be granted to an object protected by a security descriptor and, possibly, a current owner.

SeSinglePrivilegeCheck
Returns a Boolean indicating whether the current thread has at least the given privilege level.

1.12 Error Handling

IoAllocateErrorLogEntry
Allocates and initializes an error log packet; returns a pointer so the caller can supply error-log data and call **IoWriteErrorLogEntry** with the packet.

IoWriteErrorLogEntry
Queues a previously allocated error log packet, filled in by the driver, to the system error logging thread.

IoIsErrorUserInduced
Returns a Boolean indicating whether an I/O request failed due to one of the following (user-correctable) conditions: STATUS_IO_TIMEOUT, STATUS_DEVICE_NOT_READY, STATUS_UNRECOGNIZED_MEDIA, STATUS_VERIFY_REQUIRED, STATUS_WRONG_VOLUME, STATUS_MEDIA_WRITE_PROTECTED, or STATUS_NO_MEDIA_IN_DEVICE. If the result is TRUE, a removable-media driver must call **IoSetHardErrorOrVerifyDevice** before completing the IRP.

IoSetHardErrorOrVerifyDevice
Supplies the device object for which the given IRP was failed due to a user-induced error, such as supplying the incorrect media for the requested operation or changing the media before the requested operation was completed. (A file system driver uses the associated device object to send a popup to the user; the user can then correct the error or retry the operation.)

IoSetThreadHardErrorMode
Enables or disables error reporting for the current thread using **IoRaiseHardError** or **IoRaiseInformationalHardError**.

IoGetDeviceToVerify
Returns a pointer to the device object, representing a removable-media device, that is the target of the given thread's I/O request. (This routine is useful only to file systems or other highest-level drivers.)

IoRaiseHardError
Causes a popup to be sent to the user indicating that the given IRP was failed on the given device object for an optional VPB, so that the user can correct the error or retry the operation.

IoRaiseInformationalHardError
Causes a popup to be sent to the user, showing an I/O error status and optional string supplying more information.

ExRaiseStatus
Raises an error status so that a caller-supplied structured exception handler is called. (This routine is useful only to highest-level drivers that supply exception handlers, in particular to file systems.)

KeBugCheckEx
Brings down the system in a controlled manner, displaying the bugcheck code and possibly more information, after the caller discovers an unrecoverable inconsistency that will corrupt the system unless it is brought down. After the system is brought down, this routine displays bug-check and

possibly other information. (This routine can be called when debugging under-development drivers. Otherwise, drivers should never call this routine when they can handle an error by failing an IRP and by calling **IoAllocateErrorLogEntry** and **IoWriteErrorLogEntry**.)

KeBugCheck

Brings down the system in a controlled manner when the caller discovers an unrecoverable inconsistency that will corrupt the system if the caller continues to run. **KeBugCheckEx** is preferable.

KeInitializeCallbackRecord

Initializes a bug-check callback record before a device driver calls **KeRegisterBugCheckCallback**.

KeRegisterBugCheckCallback

Registers the device driver's bug-check callback routine, that is called if a system bug check occurs. Such a driver-supplied routine saves driver-determined state information, such as the contents of device registers, that would not otherwise be written into the system crash-dump file.

KeDeregisterBugCheckCallback

Removes a device driver's callback routine from the set of registered bug-check callbacks.