IBM

Operating System/2™ Technical Reference

Volume 2

Programming Family

First Edition (September 1987)

Preface

This book contains technical information pertaining to the IBM Operating System/2™ (OS/2™[1]).

This book covers topics for the system programmer, and application developer. The reader should be knowledgeable about operating systems and be proficient in one or more of the IBM Personal Computer programming languages. It is also assumed that you are familiar with the 80286 architecture.

This book contains six chapters that describe the OS/2 Application Programming Interface (API) Function Requests. The function requests are arranged in alphabetical order and reflect each call's purpose, calling sequence, parameter definitions, return codes, considerations and remarks.

Note: For a complete error code list refer to the Appendix at the back of this book.

[1] IBM Operating System/2™ and OS/2 are trademarks of International Business Machines Corporation

Related Publications

This book is intended to be used in conjunction with the:

IBM Operating System/2™ User's Reference
IBM Operating System/2™ Programmer's Guide

Other books related to OS/2 are:

IBM Personal Computer *AT®*[2] *Technical Reference*
IBM Personal Computer *XT™*[3] *Model 286 Technical Reference*
IBM Personal System/2™ Model 50/IBM Personal System/2™ Model 60 Technical Reference
IBM Personal System/2™ Model 80 Technical Reference
IBM Personal Computer *BIOS Technical Reference*
IBM Personal Computer *Macro Assembler/2*
iAPX 286 Programmer's Reference Manual including the *iAPX 286 Numeric Supplement* 210498, or the *iAPX 386 Programmer's Reference Manual* (IBSN 1-55512-022-9) Literature Department, Intel®[4] Corporation, 3065 Bowers Avenue, Santa Clara, CA. 95051

[2] Personal Computer AT® is a registered trademark of International Business Machines Corporation

[3] Personal Computer XT™ Model 286 and IBM Personal System/2™ are trademarks of International Business Machines Corporation

[4] Intel® is a registered trademark of Intel Corporation.

Contents

Chapter 1. IBM Operating System/2™ Function Requests

Applications built under IBM Operating System/2(OS/2) use a dynamic link mechanism (far CALL) to access all services. The call interface implementation is outlined below:

The OS/2 function requests must operate

1. for all memory models
2. for a wide range of IBM supported languages
3. for all dynamic link entries.

The stack passes call parameters when OS/2 service is requested, using a call-return interface. Before a call is issued, parameters are pushed onto the stack. The parameters are copied by the hardware, from the requestor's stack to the receiving program's stack. All OS/2 functions are invoked using the call-return interface. The following section explains how to use the call interface system.

How OS/2 Function Requests Work

A function call is declared EXTERNAL FAR when it is coded to a dynamically linked subroutine. The compiler generates a standard external reference. Library names that contain the dynamic link definition records are sent to the linker when the object module links. These records provide a correspondence between the called entry point and the module file that contains the routine being called.

OS/2 Function Request Format

The Function Request interfaces are set up in this book descriptively rather than in coding sequence example. The conventions described below are employed throughout this document.

Since all parameters are pushed onto the stack, there are several pseudo-instructions that describe these operations.

Operand	Description
PUSH	pushes various size items onto the stack. The data types are described below.
PUSH@	pushes the address of an item onto the stack. All addresses in these interfaces are composed of a 32-bit value, a 16-bit selector, and a 16-bit offset. These addresses point to data item types.
CALL	calls a function and accesses it via *FAR CALLS*. A parameter list contains several data items.
WORD	(2 bytes) is passed by value (pushed onto the stack) or by reference (the address is passed on the stack).
DWORD	(4 bytes) is passed by value or by reference.
ASCIIZ	is a null (00H) terminated character string, accessed by reference.
OTHER	pushes the address of a structure on to the stack and is accessed by reference.

Note: The OS/2 function calls in this book are described in mixed case for readability. The function call names are known to the system as upper case character strings. For example, the OS/2 function call "DosGetInfoSeg" is actually the external name "DOSGETINFOSEG"

If you are using a compiler which will generate mixed case external names, you may wish to code the OS/2 function calls in upper case.

Chapter 2. OS/2 DOS Calls

This chapter reflects the DOS Application Program Interface (API) only. For a complete list of all return codes, refer to the Appendix in the back of this book.

For information regarding other functional characteristics of the API, refer to the OS/2 Technical Reference, Volume 1.

DosAllocHuge – Allocate Huge Memory

Purpose

DosAllocHuge allocates multiple segments of memory.

Calling Sequence

```
EXTRN DosAllocHuge:FAR

PUSH   WORD    NumSeg          ;Number of 65536-byte segments
PUSH   WORD    Size            ;Number of bytes in last segment
PUSH@  WORD    Selector        ;Selector allocated (returned)
PUSH   WORD    MaxNumSeg       ;Max number of 65536-byte segments
PUSH   WORD    Flags           ;Allocation flags
CALL   DosAllocHuge
```

Where

NumSeg
is the number of 65536-byte segments requested.

Size
is the number of bytes requested in the last non-65536-byte segment. A value of zero indicates none.

Selector
is where the selector of the first segment allocated is returned.

MaxNumSeg
is the maximum number of 65536-byte segments an object occupies as a result of any subsequent DosReallocHuge, (see "DosReallocHuge – Change Huge Memory Size" on page 2-198). If MaxNumSeg is 0, OS/2 assumes this segment will never be increased by DosReallocHuge beyond its original size, though it may be decreased.

Flags
indicate sharing attributes of the allocated segment. The following bit values are defined below:

Bit 0 (0001b) =
segment is shareable through DosGiveSeg
Bit 1 (0010b) =
segment is shareable through DosGetSeg

Bit 2 (0100b) =
segment is discardable in low memory situations.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Family API Considerations

Some options operate differently in the DOS mode than they do in the OS/2 mode. Therefore, the following considerations apply to DosAllocHuge when coding in the DOS mode:

- Requested Size value is rounded up to the next paragraph.
- Selector is the actual segment address allocated.
- Flags must be set equal to zero.
- MaxNumSeg is ignored.

Remarks

The memory allocated by DosAllocHuge is movable and swappable. Increment the selector of the first segment to obtain a selector for a second segment. Use the increment added to the second selector to address the third segment, and so forth. Derive the increment by shifting the value "1" to the left by the amount returned as a shift count by DosGetHugeShift. For example:

- Assume DosAllocHuge is issued with NumSeg equal to three. The first segment allocated is at selector number 63.
- If DosGetHugeShift returns a shift count of four, shifting the value one by this amount results in an increment of 16
- Adding this increment to selector number 63 results in selector 79 as the second selector. Adding the increment to 79 yields selector 95 as the third selector.
- The three selector values (63, 79, and 95) are saved by the program and referenced for later use.

For any segments allocated as discardable, the data in the segments is lost when they are discarded in low memory situations. To reference the segments again the operator must reallocate them and recreate the data.

For huge segments, discarding is forced on all other segments in the huge allocation when one segment is discarded.

Memory allocated by DosAllocHuge is deallocated by DosFreeSeg. The selector returned by DosAllocHuge is passed to DosFreeSeg, which frees all the allocated memory.

Purpose

DosAllocSeg allocates a segment of memory to a requesting process.

Calling Sequence

```
EXTRN DosAllocSeg:FAR

PUSH  WORD Size        ;Number of bytes requested
PUSH@ WORD Selector    ;Selector allocated (returned)
PUSH  WORD Flags       ;Allocation flags
CALL  DosAllocSeg
```

Where

Size
is the number of bytes requested. The value specified must be less than or equal to 65535. A value of zero indicates 65536 bytes.

Selector
is where the selector of the allocated segment is returned.

Flags
indicate sharing attributes of the allocated segment. The following bit values are defined below:

Bit 0 (0001b) =
segment is shareable through DosGiveSeg
Bit 1 (0010b) =
segment is shareable through DosGetSeg
Bit 2 (0100b) =
segment is discardable in low memory situations.

Returns

IF AX = 0 then NO error

ELSE AX = error code

DosAllocSeg – Allocate Segment

Family API Considerations

Some options operate differently in the DOS mode than they do in the OS/2 mode. Therefore, the following restrictions apply to DosAllocSeg when coding in the DOS mode:

- Requested Size value is rounded up to the next paragraph.
- Selector is the actual segment address allocated.
- If Flag = one, an error is returned.

Remarks

The memory allocated by DosAllocSeg is movable and swappable. To allocate a segment of memory that can be discarded when not in use, use DosLockSeg and DosUnlockSeg in conjunction with AllocFlags bit two set. In low memory situations, a segment can be swapped if it is locked and if bit 2 is set. If the segment is unlocked, it may be discarded.

For any segments allocated as discardable, the data in the segments is lost when they are discarded in low memory situations. To reference the segments again the operator must reallocate them and recreate the data.

For huge segments, discarding is forced on all other segments in the huge allocation when one segment is discarded.

DosAllocShrSeg – Allocate Shared Segment

Purpose

DosAllocShrSeg allocates a shared memory segment to a process.

Calling Sequence

```
EXTRN DosAllocShrSeg:FAR

PUSH   WORD    Size          ;Number of bytes requested
PUSH@  ASCIIZ  Name          ;Name string
PUSH@  WORD    Selector      ;Selector allocated (returned)
CALL  DosAllocShrSeg
```

Where

Size

is the number of bytes requested. The value specified must be less than or equal to 65535. A value of 0 indicates 65536 bytes.

Name

is a symbolic name to be associated with the shared memory segment to be allocated. The name string that specifies the name for the shared memory segment must include the prefix \SHAREMEM\. For example \SHAREMEM\PUBLIC.DAT.

Selector

is where the selector of the allocated segment is returned.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

Memory allocated by DosAllocShrSeg can be moved and swapped. The selector returned to the issuing process by DosAllocShrSeg is the same as that returned to another process when it issues DosGetShrSeg to access the same segment.

DosAllocShrSeg – Allocate Shared Segment

The name assigned to the segment should be used by another process on its DosGetShrSeg call to access the same segment.

The maximum number of segments a process can define with DosAllocShrSeg or access with DosGetShrSeg is 30.

DosBeep – Generate Sound From Speaker

Purpose

DosBeep generates sound from the speaker.

Calling Sequence

```
EXTRN DosBeep:FAR

PUSH   WORD    Frequency     ;Hertz (Hz)
PUSH   WORD    Duration      ;Length of sound
CALL   DosBeep
```

Where

Frequency
is the tone in Hertz (cycles per second) in the range 37 to 32767.

Duration
is the length of the sound in milliseconds.

Returns

IF   AX = 0 then NO error

ELSE AX = error code

Remarks

DosBeep executes synchronously. An application program that invokes DosBeep waits until the specified number of milliseconds expire before it resumes execution.

DosBufReset – Commit File's Cache Buffers

Purpose

DosBufReset flushes a requesting process's cache buffers for a specific file handle.

Calling Sequence

```
EXTRN  DosBufReset:FAR

PUSH   WORD    FileHandle    ;File handle
CALL   DosBufReset
```

Where

FileHandle
is the file handle whose buffers are to be flushed. If FileHandle = FFFFH, all of the process's cache buffers that require file I/O are written out.

Returns

IF   AX = 0 then NO error

ELSE AX = error code

Remarks

Upon issuing DosBufReset, a file's directory entry is updated as if the file had been closed, however, the file remains in the open state.

To flush a requesting process's cache buffers using DosBufReset could require the user to mount and dismount a large number of removable volumes.

DosCaseMap – Perform Case Mapping

Purpose

DosCaseMap performs case mapping on a string of binary values that represent ASCII characters.

Calling Sequence

```
EXTRN DosCaseMap:FAR

PUSH   WORD    Length        ;Length of string to case map
PUSH@  OTHER   Structure     ;Input data structure
PUSH@  OTHER   BinaryString  ;Address of string of binary
CALL  DosCaseMap
```

Where

Length
is the length of the string of binary values to be case mapped.

Structure
is a two word input data structure where:

word 0
Country Code (0 - default country)
word 1
Code Page ID (0 - current process code page).

BinaryString
is a string of binary characters to be case mapped. They are case mapped in place so the results appear in BinaryString and the input is destroyed.

Returns

IF AX = 0 then NO error

ELSE AX = error code

DosCaseMap – Perform Case Mapping

Remarks

The case map information is taken from the country information file. See the COUNTRY statement in the *IBM Operating System/2™ User's Reference* for information on how to specify the country information file.

If the Country Code in Structure is 0, the case mapping is performed using the information for the country specified in the COUNTRY statement in CONFIG.SYS. If the Country Code in Structure is not zero, the case mapping is performed using the information for that country.

If the CodePageID in Structure is 0, the case mapping is performed using the information for the current process code page. Refer to "DosSetCp – Set Code Page" on page 2-222 and the CHCP command in the *IBM Operating System/2™ User's Reference* for information on setting the process code page. If the CodePageID in Structure is not 0, the case mapping is performed using the information for that code page.

The returned country dependent information may be for the default country and current process code page or for a specific country and code page.

Purpose

DosChDir defines the current directory for the requesting process.

Calling Sequence

```
EXTRN  DosChDir:FAR

PUSH@  ASCIIZ  DirName       ;Directory path name
PUSH   DWORD   0             ;Reserved (must be zero)
CALL   DosChDir
```

Where

DirName
is the directory path name. The string is limited to 64 characters.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

The directory path is not changed if any member of the path does not exist. The current directory changes only for the requesting process.

DosChgFilePtr – Change (Move) File Read/Write Pointer

Purpose

DosChgFilePtr moves the read/write pointer in accordance with the method specified.

Calling Sequence

```
EXTRN  DosChgFilePtr:FAR

PUSH   WORD    FileHandle    ;File handle
PUSH   DWORD   Distance      ;Distance to move in bytes
PUSH   WORD    MoveType      ;Method of moving (0, 1, 2)
PUSH@  DWORD   NewPointer    ;New Pointer Location
CALL   DosChgFilePtr
```

Where

FileHandle
is the handle returned by a previous DosOpen call.

Distance
is the distance (offset) to move in bytes.

MoveType
is the method of moving:

If value = 0
move pointer Distance bytes (offset) from the beginning of the file.

If value = 1
move pointer to the current location plus offset.

If value = 2
move pointer to the end-of-file plus offset. Use this method to determine a file's size.

NewPointer
is the area where the system returns the new pointer location.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

None

DosCLIAccess — Request CLI/STI Privilege

Purpose

DosCLIAccess requests I/O privilege for disabling and enabling interrupts. Access to ports must be granted via DosPortAccess.

Calling Sequence

```
EXTRN  DosCLIAccess:FAR

CALL   DosCLIAccess
```

Where

None

Returns

AX = 0

Remarks

Applications that only use CLI/STI in IOPL segments must request CLI/STI privilege from the operating system.

Applications that use IN/OUT instructions to I/O ports must request I/O privilege with DosPortAccess. (See "DosPortAccess — Request Port Access" on page 2-164 for more detail) Request for port access will also grant CLI/STI privilege from the operating system.

Purpose

DosClose closes a specific file handle.

Calling Sequence

```
EXTRN DosClose:FAR

PUSH   WORD    FileHandle    ;File handle
CALL   DosClose
```

Where

FileHandle

is the handle returned by a previous DosOpen or DosMakePipe call.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

Closing a file handle closes the file, updates the directory, and writes the file's internal buffers to the media.

When hard error popups are disabled, issue DosBufReset before issuing DosClose.

DosCloseQueue — Close Queue

Purpose

DosCloseQueue closes the queue in use by the requesting process.

Calling Sequence

```
EXTRN  DosCloseQueue:FAR

PUSH   WORD    QueueHandle   ;Handle of queue
CALL   DosCloseQueue
```

Where

QueueHandle
is the handle returned from a previous DosCreateQueue or DosOpenQueue call.

Returns

IF   AX = 0 then NO error

ELSE AX = error code

Remarks

When DosCloseQueue is issued, if the requesting process is the owner of the queue, all outstanding elements are purged. Other processes that have the queue open receive the QUEUE_DOES NOT_EXIST (invalid queue handle) return code on the next request.

For a writer of the queue using DosCloseQueue, access to the queue is terminated, but the queue is not affected.

Purpose

DosCloseSem closes a specific system semaphore.

Calling Sequence

```
EXTRN  DosCloseSem:FAR

PUSH   DWORD   SemHandle     ;Semaphore handle
CALL   DosCloseSem
```

Where

SemHandle
is the handle returned from a previous DosCreateSem or DosOpenSem call.

Returns

IF   AX = 0 then NO error

ELSE AX = error code

Remarks

When all processes using the semaphore issue DosCloseSem, the semaphore is deleted. If a process terminates with open semaphores, the system closes the semaphores. If any of the semaphores are owned by the current process, the first thread given the semaphore wakes with the ERROR_ SEM_OWNER_DIED. This indicates the owner of the semaphore ended without releasing it and the resources are in an indeterminate state.

DosCreateCSAlias – Create CS Alias

Purpose

DosCreateCSAlias creates a code segment alias descriptor for a data segment passed as input.

Calling Sequence

```
EXTRN  DosCreateCSAlias:FAR

PUSH   WORD    DataSelector  ;Data segment selector
PUSH@  WORD    CodeSelector  ;Code segment selector (returned)
CALL   DosCreateCSAlias
```

Where

DataSelector
is the data segment selector.

CodeSelector
is where the selector of the code segment alias descriptor is returned.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Family API Considerations

The returned selector is the segment address of the allocated memory. When the returned selector or the original selector is freed, OS/2 immediately deallocates the block of memory.

Remarks

Any selector valid for DS, ES or SS, can be used as the data segment selector passed as input to DosCreateCSAlias. However, its segment must be exclusively accessible by the process and not a huge segment. Shared memory segments and dynamically linked global data segments cannot be used as input for DosCreateCSAlias.

The code segment selector returned as output is valid for CS. If a valid procedure is stored in the segment that uses the data selector, the procedure can be called using the CS alias. The procedure is called from privilege level three or I/O privilege level.

Use DosFreeSeg to free a CS alias created with DosCreateCSAlias. Procedures in the segment continue to be referenced if the data selector for the aliased segment is passed to DosFreeSeg, however the CS alias selector is not affected.

DosCreateQueue – Create Queue

Purpose

DosCreateQueue creates a queue owned by a creating process.

Calling Sequence

```
EXTRN  DosCreateQueue:FAR

PUSH@  WORD    RWHandle       ;Queue handle (returned)
PUSH   WORD    QueuePrty      ;Ordering to use for elements
PUSH@  ASCIIZ  QueueName      ;Queue name string
CALL   DosCreateQueue
```

Where

RWHandle
is where the read/write handle of the queue is returned. The handle is used by the requestor on return.

QueuePrty
indicates the priority ordering algorithm to use for elements placed in the queue. The valid values and their meanings are:

0 = FIFO queue
1 = LIFO queue
2 = Priority queue (sender specifies priority zero to 15.)

QueueName
is the name of the queue. The name string that specifies the name for the queue must include \QUEUES\ as a path name. For example, \QUEUES\RETRIEVE\CONTROL.QUE is a valid queue name. The same name must be specified when calling DosOpenQueue for the process which adds elements to the queue.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

A queue must be created before it can be opened. The process that creates a queue owns the queue. When specifying a name for a queue, the *ASCIIZ* name string must include the prefix \QUEUES\\. Only the owner can access or remove the elements in a queue. Any process can open a queue and place data in it. When a queue is created, each process writing to it must open the queue through DosOpenQueue. When a process creates or opens a queue, any thread in that process can access the queue with equal authority. This provides the capability for multi-server queues.

A queue exists when it is created and ceases to exist when the owner closes it. If other processes have a queue open when the owner closes it, subsequent requests return with the "queue does not exist" return code.

DosCreateSem — Create System Semaphore

Purpose

DosCreateSem creates a system semaphore used by semaphore manipulation calls such as DosSemRequest, DosSemClear, DosSemSet, DosSemSetWait, DosSemWait, and DosMuxSemWait.

Calling Sequence

```
EXTRN  DosCreateSem:FAR

PUSH   WORD    NoExclusive   ;Indicate no exclusive ownership
PUSH@  DWORD   SemHandle     ;Semaphore handle (returned)
PUSH@  ASCIIZ  SemName       ;Semaphore name string
CALL   DosCreateSem
```

Where

NoExclusive
is an indicator that the owning process does not want exclusive ownership of the semaphore. Other processes can alter the state of the semaphore while it is owned.

If value = 0
the owning process has exclusive ownership of the semaphore.

If value = 1
exclusive ownership is not required.

SemHandle
is where the handle assigned the semaphore is returned.

SemName
is the name of the system semaphore. The name string that specifies the name for the semaphore must include \SEM\ as a path name. For example, \SEM\RETRIEVE\SIGNAL.SEM is a valid semaphore name. The same name must be specified when calling DosOpenSem for any other process that needs to access the same semaphore.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

DosCreateSem creates a system semaphore to control access to a serially reusable resource through multiple asynchronous threads. DosCreateSem opens and allows access to a semaphore. The NoExclusive operand allows an owned semaphore to be modified by a process other than the owner. When the system semaphore is exclusive this means the semaphore has a use count associated with it. When specifying the name for a system semaphore, the ASCIIZ name string must include the prefix *SEM*\.

DosCreateThread – Create Another Thread of Execution

Purpose

DosCreateThread creates an asynchronous thread of execution under the current process.

Calling Sequence

```
EXTRN  DosCreateThread:FAR

PUSH   DWORD   PgmAddress      ;Program address
PUSH@  WORD    ThreadIDWord    ;New thread ID (returned)
PUSH   DWORD   NewThreadStack  ;End of stack
                               ; for new thread
CALL   DosCreateThread
```

Where

PgmAddress
is the program to receive control under the new thread.

ThreadIDWord
is where the thread ID of the new thread is to be returned.

NewThreadStack
points to the end of the new thread's stack.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

DosCreateThread causes a far call to be generated by the system at PgmAddress. The new thread is identical to the requesting thread and can access all files and resources owned by the parent process. Operations of all threads within a process are identical. The only thread-specific information maintained is register contents, stack, and dispatching priority.

Within a given process, any thread can open a file or device and any other thread can subsequently issue read or writes to that handle. A similar case exists with pipes, queues and system semaphores.

DosCwait – Wait for Child Termination

Purpose

DosCwait places the current thread in a wait state until a child process terminates. When this occurs, the process ID and termination code of the ending process is returned.

Calling Sequence

```
EXTRN  DosCwait:FAR

PUSH   WORD    ActionCode     ;Execution options
PUSH   WORD    WaitOption     ;Wait options
PUSH@  DWORD   ReturnCodes    ;Termination Codes (returned)
PUSH@  WORD    ProcessIDWord  ;Process ID (returned)
PUSH   WORD    ProcessID      ;Process ID of process to wait for
CALL   DosCwait
```

Where

ActionCode
indicates the process of interest:

If value = 0
the current thread waits until the indicated process ends.

If value = 1
the current thread waits until the indicated process and all its child processes end.

WaitOption
indicates return if no child process ends:

If value = 0
the current thread waits if no child process ends or until there are no child processes outstanding.

If value = 1
the current thread does not wait for child processes to end.

ReturnCodes
is where the termination code and result code indicate the reason for the child's termination is returned.

The first word is a termination code. It is furnished by the system, and describes why the child terminated. The values returned and their meanings are:

0 = EXIT (normal exit)
1 = hard error abnormally end (or stop) execution
2 = trap operation
3 = unintercepted DosKillProcess.

The second word passes the ResultCode specified by the terminating process on its last DosExit call.

ProcessIDWord
is where the process ID of the ending process is returned.

ProcessID
is the ID of the terminating process being waited for:

If value = 0
the current thread waits until any child process ends.
If value ≠ 0
the current thread waits until the indicated process and all its child processes end.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

DosCwait waits for completion of a child process. If a child process starts other processes, DosCwait waits for the grandchild processes to complete before it returns. It does not report their status. If the indicated child process has multiple threads, the result code is returned on the last DosExit request.

If no child processes were started, DosCwait returns with an error. If no child processes terminate, DosCwait waits until one terminates before returning to the parent.

Check the process ID to verify which child a return code is from. To wait for all child processes and grandchild processes to end, issue

DosCwait – Wait for Child Termination

DosCwait (with ActionCode = 1, ProcessID = 0) repeatedly until the NO_CHILD_PROCESS_EXISTS return code is given.

If DosCwait is used to wait for a child process to end, use DosExecPgm to create the child process and indicate "AsyncTraceFlags=2".

Purpose

DosDelete removes a directory entry associated with a filename.

Calling Sequence

```
EXTRN  DosDelete:FAR

PUSH@  ASCIIZ  FileName      ;Filename path
PUSH   DWORD   0             ;Reserved (must be zero)
CALL   DosDelete
```

Where

FileName
is the name of the file to be deleted.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

Global filename characters are not allowed in any part of the ASCIIZ string. DosDelete cannot delete read-only files. To delete a read-only file, use DosSetFileMode to change the file's read-only attribute to 0, then delete the file.

DosDevConfig – Get Device Configuration

Purpose

DosDevConfig gets information about attached devices.

Calling Sequence

```
EXTRN DosDevConfig:FAR

PUSH@  OTHER   DeviceInfo     ;Returned information
PUSH   WORD    Item           ;Item number
PUSH   WORD    Parm           ;Reserved
CALL   DosDevConfig
```

Where

DeviceInfo
is where the requested information is returned.

Item
indicates what device information to return:

Item	Returned Device Information
0	Number of printers attached
1	Number of RS232 ports
2	Number of internal diskette drives
3	Presence of math coprocessor (where 0 = not present, 1 = present)
4	PC Submodel Type (where the return is the system submodel byte)
5	PC Model Type (where the return is the system model byte)
6	Display adapter type (where 0 = monochrome mode compatible, 1 = other).

Parm
is reserved and should be set to 0.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

The system model (function 5) and submodel (function 4) information is obtained from BIOS.

In addition, the number of devices attached in a PS/2 environment reflect only devices that are "awake". Devices that are "asleep" are not counted.

DosDevIOCtl – I/O Control for Devices

Purpose

DosDevIOCtl performs control functions on a device specified by an opened device handle.

Calling Sequence

```
EXTRN  DosDevIOCtl:FAR

PUSH@  OTHER   Data           ;Data area
PUSH@  OTHER   ParmList       ;Command arguments
PUSH   WORD    Function       ;Device function
PUSH   WORD    Category       ;Device category
PUSH   WORD    DevHandle      ;Specifies the device
CALL   DosDevIOCtl
```

Where

Data
is the data area.

ParmList
is a command-specific argument list.

Function
is the device-specific function code.

Category
is the device category.

DevHandle
is a device handle returned by DosOpen or a standard (open) device handle.

Returns

IF   AX = 0 then NO error

ELSE AX = error code

Family API Considerations

Level of support for DosDevIOCtl is identified by category and function call, with a noted restriction if it is not supported by DOS 3.3. Functions tend to be more restrictive in lower versions of DOS.

- Category 1 supported in Family API
- 41h Set Baud Rate
- 42h Set Line Control
- All other Category one functions not supported for DOS 3.3
- Category 2 not supported in the Family API
- Category 3 not supported in the Family API
- Category 4 not supported in the Family API
- Category 5 supported in the Family API
- 42h Set Frame Control (for IBM Graphics Printers only)
- 44h Get Infinite Retry (DOS 3.3 the function is in effect for duration of program only.)
- 46h Initialize Printer
- 62h Get Frame Control (not supported for DOS 3.3)
- 64h Get Infinite Retry
- 66h Get Printer Status
- Category 6 not supported by the Family API
- Category 7 not supported by the Family API
- Category 8 supported in Family API
- 00h Lock Drive (not supported below DOS 3.3)
- 01h Unlock Drive (not supported below DOS 3.3)
- 02h Redetermine Media (not supported below DOS 3.3)
- 03h Set Logical Map (not supported below DOS 3.3)
- 20h Block Removable (not supported below DOS 3.3)
- 21h Get Logical Map (not supported below DOS 3.3)
- 43h Set Device Parameters (not supported DOS 3.3)
- 44h Write Track (not supported DOS 3.3)
- 45h Format Track (not supported DOS 3.3)
- 63h Get Device Parameters (not supported DOS 3.3)
- 64h Read Track (not supported DOS 3.3)
- 65h Verify Track (not supported DOS 3.3)
- Category 9 reserved
- Category 10 (0Ah) not supported in the Family API
- Category 11 (0Bh) not supported in the Family API.

Remarks

Values returned in the range hex FF00 to FFFF are user dependent error codes. Values returned in the range hex FE00 to FEFF are device driver dependent error codes. Refer to the *IBM Operating System/2 Technical Reference, Volume 1* for complete listing of DevHlp calls.

Purpose

DosDupHandle returns a new file handle for an open file that refers to the same file at the same position.

Calling Sequence

```
EXTRN  DosDupHandle:FAR

PUSH   WORD    OldFileHandle  ;Existing file handle
PUSH@  WORD    NewFileHandle  ;New file handle (returned)
CALL   DosDupHandle
```

Where

OldFileHandle
is the current file handle.

NewFileHandle
is where the new file handle is returned. When DosDupHandle is called, if this word contains FFFFH, OS/2 allocates a new file handle and returns its value here. If this word contains any other value, OS/2 assumes this value is to be the new file handle.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

Duplicating the handle duplicates and ties all handle-specific information between OldFileHandle and NewFileHandle.

A file handle value other than FFFFH specified in NewFileHandle causes OS/2 to close the file handle before redefining it as the duplicate of OldFileHandle.

DosDupHandle — Duplicate File Handle

The values for NewFileHandle include:

FFFFH	= assign new handle
0000H	= standard input
0001H	= standard output
0002H	= standard error
nnnn	the handle of any currently open file.

Note: Avoid using other arbitrary values.

If the read/write pointer of either handle is moved by DosRead, DosWrite, or DosChgFilePtr, the pointer for all duplicated handles is also changed.

Issuing DosClose against a file handle does not affect the duplicate handle.

DosEnterCritSec – Enter Critical Section of Execution

Purpose
DosEnterCritSec disables thread switching for the current process.

Calling Sequence
```
EXTRN  DosEnterCritSec:FAR

CALL   DosEnterCritSec
```

Where
None

Returns
```
IF     AX = 0  then NO error

ELSE AX = error code
```

Remarks
When the current thread issues DosExitCritSec, other threads in a process resume normal dispatching.

If a signal occurs, the first thread begins execution to process the signal even though another thread in the process has a DosEnterCritSec active. Any processing the first thread does to satisfy the signal must not access the critical resource intended to be protected by the DosEnterCritSec.

A count of the number of outstanding DosEnterCritSec requests is maintained. The count is increase on DosEnterCritSec requests and decreased on DosExitCritSec requests. A DosExitCritSec request does not cause normal thread dispatching to restore while the count is greater than zero. This count is maintained in a word and if overflow is encountered, the count is set to the maximum value, an error is returned, and the operation does not perform.

DosEnterCritSec — Enter Critical Section of Execution

Once a DosEnterCritSec request has been made, no dynamic link calls should be made until the corresponding DosExitCritSec call has been completed.

DosErrClass – Classify Error Codes

Purpose
DosErrClass helps OS/2 applications respond to error codes (return codes) received from the OS/2 API.

Calling Sequence
```
EXTRN DosErrClass:FAR

PUSH   WORD    Code           ;Error code for analysis
PUSH@  WORD    Class          ;Error classification (returned)
PUSH@  WORD    Action         ;Recommended action (returned)
PUSH@  WORD    Locus          ;Error locus (returned)
CALL   DosErrClass
```

Where
Code
is the error code returned by an OS/2 function.

Class
is where the classification of that error is returned.

Action
is where the recommended action for that error is returned.

Locus
is where the origin of the error is returned.

Returns
AX = 0

Remarks
The input is the return code returned from another function call, and the output is a classification of the return and recommended action. Depending on the application, the recommended action could be followed, or a more specific application recovery could be performed.

When DosErrClass is called by a family application, it returns a valid error classification for returns that have occurred. The classifications of a given return code may not be the same for the Family API and the OS/2 mode applications.

The following values are returned in Class, Action, and Locus:

Class Definitions

Value	Mnemonic	Description
1	OUTRES	Out of resources
2	TEMPSIT	Temporary situation
3	AUTH	Authorization failed
4	INTRN	Internal error
5	HRDFAIL	Device hardware failure
6	SYSFAIL	System failure
7	APPERR	Probable application error
8	NOTFND	Item not located
9	BADFMT	Bad format for call/data
10	LOCKED	Resource/data locked
11	MEDIA	Incorrect media, CRC error
12	ALREADY	Resource/action already taken/done/exists
13	UNK	Unclassified
14	CAN'T	Can't perform requested action
15	TIME	Timeout

Action Definitions

Value	Mnemonic	Description
1	RETRY	Retry immediately
2	DLYRET	Delay and retry
3	USER	Bad user input - get new values
4	ABORT	Terminate in an orderly manner
5	PANIC	Terminate immediately
6	IGNORE	Ignore error
7	INTRET	Retry after user intervention

Locus Definitions

Value	Mnemonic	Description
1	UNK	Unknown
2	DISK	Random access device such as a disk
3	NET	Network
4	SERDEV	Serial device
5	MEM	Memory

DosError – Enable Hard Error Processing

Purpose

DosError allows a process to disable user notification (from OS/2) on hard errors and program exceptions.

Calling Sequence

```
EXTRN  DosError:FAR

PUSH   WORD    Flag          ;Action flag
CALL   DosError
```

Where

Flag

is a bit field defined as shown below. The unused high-order bits are reserved and must be 0.

```
xxxxxxxxxxxxxxx0  disable hard error popups
                    (fail requests)
xxxxxxxxxxxxxxx1  enable hard error popups
xxxxxxxxxxxxxx0x  enable exception popups
xxxxxxxxxxxxxx1x  disable exception popups
```

Returns

IF   AX = 0 then NO error

ELSE AX = error code

Family API Considerations

Some options operate differently in the DOS mode than they do in the OS/2 mode. Therefore, the following restriction applies to DosError when coding in the DOS mode:

For Flags, a value of 0000 will cause all subsequent INT 24's to be failed until a subsequent call with a value of 1 is issued.

Remarks

The default situation is both hard error pop-ups and exception pop-ups are enabled, if DosError is not issued. DosError allows an OS/2 process to disable user notification if a program (or untrapped numeric processor) exception occurs. If end user notification is disabled, and, if one of these exceptions occurs, the process is terminated.

DosExecPgm – Execute Program

Purpose

DosExecPgm allows a program to request another program to execute as a child process. The requestor's process continues to execute asynchronously to the new program.

Calling Sequence

```
EXTRN  DosExecPgm:FAR

PUSH@  OTHER   ObjNameBuf      ;Object name buffer (returned)
PUSH   WORD    ObjNameBufL     ;Length of object name buffer
PUSH   WORD    ExecFlags       ;Execute asynchronously/trace
PUSH@  ASCIIZ  ArgPointer      ;Argument string
PUSH@  ASCIIZ  EnvPointer      ;Environment string
PUSH@  DWORD   ReturnCodes     ;Termination codes (returned)
PUSH@  ASCIIZ  PgmPointer      ;Program filename
CALL   DosExecPgm
```

Where

ObjNameBuf

is a buffer where the name of the object that contributed to the failure of DosExecPgm is returned.

ObjNameBufL

is the length, in bytes, of the buffer described by ObjNameBuf.

ExecFlags

is an indicator that the requested program is to execute asynchronous to the requestor, with/without tracing, or in a session separate from the parent.

Value Definition

If value = 0

the program executes synchronously to the parent process. The termination code and result code is stored in the two-word structure ReturnCodes.

If value = 1
: the program executes asynchronously to the parent process. When the requested program terminates, its ResultCode is discarded. The Process ID is stored in the first word of the two-word structure ReturnCodes.

If value = 2
: the program executes asynchronously to the parent process. When the requested program terminates, its ResultCode is saved for interrogation by a DosCwait request. The Process ID is stored in the first word of the two-word structure ReturnCodes.

If value = 3
: the program executes under conditions for tracing. The parent process is the debugger and the child is the process to be debugged.

If value = 4
: the program executes asynchronously to the parent process detached from the parent process session. When a process is started as a detached process it is not affected if the parent process is stopped. The detached process is treated as an orphan of the parent process. A program executed with this option executes in the background. It should not require any input from the keyboard or output to the screen other than VioPopUps. It should not issue any VIO, KBD or MOU calls.

If value = 5
: the program is loaded into storage and made ready to execute, but is not placed into execution until the session manager thaws the process.

Some memory is consumed for uncollected result codes. Issue DosCwait to release this memory. If result codes are not collected, then ExecFlags = 0 or 1 should be used.

ArgPointer
is a set of argument strings passed to the target program. These strings represent "command parameters" for the program as opposed to the "environment parameters."

The convention used by CMD.EXE is that the first of these strings is the program name (as entered after the command prompt or found in a batch file) and the second string is the remaining characters from the command prompt.

If no argument string is passed to the program, push a double-word of 0s.

EnvPointer
is a block of text strings that are passed to the program. These strings convey configuration parameters and represent the combination of the current value of all "Set Symbols" for the current program.

When an indicated program gets control, it receives:

- A pointer to a copy of the environment
- A string of the fully qualified path of the filename of the program being started
- A copy of the argument strings passed to the target program.

A coded example of this follows:

```
eo:   ASCIIZ string 1   ; environment string 1
      ASCIIZ string 2   ; environment string 2
:
      ASCIIZ string n   ; environment string n
      Byte of 0
:
po:   ASCIIZ            ; string of filename
                        ; of program to run.
:
ao:   ASCIIZ            ; argument string 1
      ASCIIZ            ; argument string 2
      Byte of 0
```

The beginning of the environment segment is "eo" and "ao" is the offset of the first argument string in that segment. Register BX contains "ao" on entry to the target program. The environment strings have the form parameter = value. A 0 value for the address of EnvPointer causes the newly created process to inherit the unchanged parent's environment.

ReturnCodes

is a structure where the Process ID or termination code and the result code, indicating the reason for the child's termination are returned.

For an asynchronous request the first word is the process ID of the child process.

For a synchronous request the first word is a termination code furnished by the system that describes why the child terminated. The values returned and their meanings are:

0 = EXIT (normal exit)
1 = Hard error abort
2 = Trap operation
3 = Unintercepted DosKillProcess.

The second word is used to pass the ResultCode specified by the terminating process on its last DosExit call.

PgmPointer

is the name of the file that contains the program to be executed. When the environment is passed to the target program, this name is copied into "po" in the above environment description.

If the string appears to be a fully qualified path (for example: contains a : (colon)in the second position and/or contains a "\") the program is loaded from the indicated drive:directory. If neither of these are true, and this filename is not found in the current directory, each drive:directory specification in the path defined in the current program's environment is searched for this file. Any extension (.xxx) is acceptable for a program filename.

Returns

IF AX = 0 then NO error

ELSE AX = error code

DosExecPgm – Execute Program

Family API Considerations

Some options operate differently in the DOS mode than they do in the OS/2 mode. Therefore, the following restrictions apply to DosExecPgm when coding in the DOS mode:

- If ExecFlags = non-zero, DosExecPgm returns the following error code, ERROR_INVALID_DATA.
- This field must be set to 0. This value will not be related to the PID of the program being executed.

Remarks

The target program is located and loaded into storage if necessary. A process is created and executed for the target program. If asynchronous execution is not indicated, the requesting process waits pending completion of the target program

The new process is created with an address space separate from its parent, that is, a new LDT built for the process.

The new process inherits all the file handles and pipes from its parent without the same access rights. Files are inherited except those opened with no inheritance indicated. Pipes are inherited.

The parent process has control of the meanings of standard input, output, and error. The parent can write a series of records to a file, open the file as standard input, open a listing file as standard output, and execute a sort program that takes input from standard input and writes to standard output.

To test whether a program is running detached, use the following method. Issue a video call, (for example, VioGetAnsi). If the call is not issued within a video popup and the process is detached, the video call will return error code ERROR_VIO_DETACHED in AX.

Purpose

DosExit is issued when a thread completes executing. The current thread or process ends.

Calling Sequence

```
EXTRN  DosExit:FAR

PUSH   WORD    ActionCode    ;Indicates end thread or process
PUSH   WORD    ResultCode    ;Result Code to save for DosCwait
CALL   DosExit
```

Where

ActionCode
indicates whether to terminate the process and all its threads.

If value = 0
the current thread ends.
If value = 1
all threads in the process end.

ResultCode
is the program's completion code. It is passed to any thread that issues DosCwait for this process.

Returns

None

Family API Considerations

Some options operate differently in the DOS mode than they do in the OS/2 mode. Therefore, the following restrictions apply to DosExit when coding in the DOS mode:

There is no thread support in DOS 3.3, therefore DosExit exits the currently executing program.

If ActionCode = 0 this option is ignored. It is equivalent to an ActionCode = 1.

Remarks

If the ending thread is the last thread in a process, or if the request indicates to terminate all threads in the process, the process also terminates. All but one thread is terminated and that thread executes routines in the DosExitList list.

When this is completed, this thread and all other resources owned by this process are released. Since the system can start threads on behalf of an application, a request intended to terminate a process must specify ActionCode = 1 regardless of how many threads the application author believes to be executing.

Do not terminate thread 1 without terminating the process. When thread 1 ends, any monitors or signal processing routines set for this process will end. To prevent unpredictable results, specify action code = 1 when ending thread 1 to ensure the process ends.

DosExitCritSec – Exit Critical Section of Execution

Purpose
DosExitCritSec re-enables thread switching for the current process.

Calling Sequence

```
EXTRN  DosExitCritSec:FAR

CALL   DosExitCritSec
```

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks
DosExitCritSec executes after DosEnterCritSec and restores normal thread switching to the threads in a process.

A count of the number of outstanding DosEnterCritSec requests is maintained. The count is increased on DosEnterCritSec requests and decreased on DosExitCritSec requests. A DosExitCritSec request does not cause normal thread dispatching to restore while the count is greater than zero. This count is maintained in a word and if this word is decremented below zero (underflow), the count is set to zero, an error is returned, and the operation does not execute.

DosExitList – Routine List for Process Termination

Purpose

DosExitList maintains a list of routines that execute when the current process ends.

Calling Sequence

```
EXTRN  DosExitList:FAR

PUSH   WORD    Function       ;Function request code
PUSH   DWORD   RtnAddress     ;Routine address
CALL   DosExitList
```

Where

Function

indicates whether to add or remove a routine address from the list. The values and their meanings are:

1 = Add address to termination list
2 = Remove address from termination list
3 = Termination processing complete, transfer to next address on termination list.

RtnAddress

is the routine to be executed.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

DosExitList defines a routine that gets control when a process completes executing. Multiple routines can be defined to receive control when a process terminates. For each process, OS/2 maintains a list of routines. When the process terminates, it transfers control to each address on the list. If there are multiple addresses on the list, each get control in an indeterminate order.

DosExitList can be used in a library module to free resources or semaphores left busy when a client program ends. Before transferring control to the routines on the termination list, OS/2 resets the stack to its initial value. Transfer is by way of a JMP instruction. The routine must be in the address space of the terminating process. When complete the termination routine at that address issues DosExitList with Function = 3. Control is then transferred to another address on the exit list. When all addresses are serviced, the process completes exiting.

It is important that the exit routines be short and fail-safe. If a routine does not issue a DosExitList Function 3, the process stops, and OS/2 prevents termination.

When DosExitList routines execute, the process is in a state of partial termination. To insure good response there should be minimum delay in allowing termination to complete. All threads except the one executing the DosExitList routines are destroyed.

Most OS/2 system calls are valid in a DosExitList routine, however, certain functions such as DosCreateThread and DosExecPgm are not. When the exit list routine receives control, the first parameter on the stack (located at SS:SP+4) contains an indicator that explains why the process ended. The values provided are the same as those provided by the system as termination codes on DosCwait or DosExecPgm requests. The values returned and their meanings are:

0 = EXIT (normal exit).
1 = Hard error abort
2 = Trap operation
3 = Un-intercepted DosKillProcess.

DosFileLocks – File Lock Manager

Purpose

DosFileLocks locks and unlocks a range in an opened file.

Calling Sequence

```
EXTRN   DosFileLocks:FAR

PUSH    WORD     FileHandle      ;File handle
PUSH@   OTHER    UnLockRange     ;UnLock range
PUSH@   OTHER    LockRange       ;Lock range
CALL    DosFileLocks
```

Where

FileHandle
is the file handle.

UnLockRange
is the range to be unlocked, specified as a double-word pair.

The first double-word is the FileOffset where the locked area begins.

The second double-word is the RangeLength.

A null pointer to UnLockRange specifies unlocking is not required.

LockRange
is the range to be locked, specified as a double-word pair.

The first double-word is the FileOffset where the locked area begins.

The second double-word is the RangeLength.

A null pointer to LockRange specifies locking is not required.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

DosFileLocks provides a mechanism for excluding other processes read/write access to regions of a file. DosFileLocks is used when a file is opened using deny read or deny none sharing modes or when the file is opened for read/write and deny write sharing mode only. The locked regions can be anywhere in the logical file.

Locking beyond end-of-file is not an error. A region should only remain locked for a short time. Duplicating the handle duplicates access to the locked regions. Access to the locked regions is not duplicated across the DosExecPgm system call. The method for using locks is to lock the desired region and examine the error code.

If unlocking is specified, the function first unlocks the specified area using UnLockRange. After UnlockRange is processed, then the locking of a range (if specified via LockRange) is done.

A lock range must be cleared of any locked subranges or locked overlapping ranges. When a file with locks closes, the locks release in no defined order. When an open file containing an open file with locks terminates, the file closes and the locks release.

DosFindClose – Close Find Handle

Purpose

DosFindClose closes the association between a directory handle and a DosFindFirst or DosFindNext directory search function.

Calling Sequence

```
EXTRN  DosFindClose:FAR

PUSH   WORD    DirHandle      ;Directory search handle
CALL   DosFindClose
```

Where

DirHandle
is the handle previously associated with a DosFindFirst by the system, or used with a DosFindNext directory search function.

Returns

IF   AX = 0 then NO error

ELSE AX = error code

Remarks

When DosFindClose is issued, a subsequent DosFindNext call for the closed DirHandle will fail unless an intervening DosFindFirst has been issued specifying DirHandle.

Purpose

DosFindFirst finds the first set of filenames that match a given file specification.

Calling Sequence

```
EXTRN  DosFindFirst:FAR

PUSH@  ASCIIZ  FileName      ;File path name
PUSH@  WORD    DirHandle     ;Directory search handle
PUSH   WORD    Attribute     ;Search attribute
PUSH@  OTHER   ResultBuf     ;Result buffer
PUSH   WORD    ResultBufLen  ;Result buffer length
PUSH@  WORD    SearchCount   ;Number of entries to find
PUSH   DWORD   0             ;Reserved (must be 0)
CALL   DosFindFirst
```

Where

FileName

is the path name of the files to be found.

DirHandle

is the directory handle associated by the system with a specific request. A DirHandle value of 0001H is defined as always available. A DirHandle value of FFFFH allocates a handle to the user. The handle is returned by overwriting the FFFFH. Reuse of this DirHandle in another DosFindFirst closes the association with the previous DosFindFirst and opens a new association.

Attribute

is the attribute used to search for the file.

If Attribute is 0, normal file entries are found. Entries for subdirectories, hidden, and system files, are not returned.

If Attribute is set for hidden or system files, or directory entries, it is considered an inclusive search. All normal file entries plus all entries matching the specific attributes are returned. Set attribute to hidden + system + directory (all three bits on) to look at all directory entries except the volume label.

Attribute cannot specify the volume label. Volume labels are queried using DosQFsInfo.

ResultBuf
is where the information is returned. It contains one or more entires of the following format:

2 bytes -	File date of creation
2 bytes -	File time of creation
2 bytes -	File date of last access
2 bytes -	File time of last access
2 bytes -	File date of last write
2 bytes -	File time of last write
2 bytes -	File end of data (low word)
2 bytes -	File end of data (high word)
2 bytes -	file allocation (low word)
2 bytes -	file allocation (high word)
2 bytes -	File attribute
1 byte -	Length of of ASCIIZ name string
n bytes -	ASCIIZ name string.

This information is as accurate as the most recent DosClose or DosBufReset.

ResultBufLen
is the length of ResultBuf.

SearchCount
is the number of matching entries requested in ResultBuf. On return, this field contains the number of entries placed into ResultBuf.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Family API Considerations

Some options operate differently in the DOS mode than they do in OS/2 mode. Therefore, the following restrictions apply to DosFindFirst when coding in the DOS mode:

DirHandle must always equal one or FFFFH on the initial call to DosFindFirst. Subsequent calls to DosFindFirst must have a DirHandle of 1 unless a DosFindClose had been issued: in that case, one or FFFFH is allowed.

Remarks

DosFindNext uses the directory handle to repeat the related DosFindFirst.

To find all files that match a given pattern, issue DosFindFirst to find the first file. Repeatedly issue DosFindNext to find the next file (specify the DirHandle returned by DosFindFirst) until the return code indicates ERROR_NO_MORE_FILES is returned. Finally, issue DosFindClose to close the directory handle.

The time is mapped in the bits as follows:

```
15 14 13 12 11 10 9  8  7  6  5  4  3  2  1  0
h  h  h  h  h  m  m  m  m  m  m  x  x  x  x  x
```

Where:

hh	binary number of hours (0-23)
mm	binary number of minutes (0-59)
xx	binary number of two-second increments

Note: The time is stored with the least significant byte first.

The mm/dd/yy are mapped in the bits as follows:

```
15 14 13 12 11 10 9  8  7  6  5  4  3  2  1  0
y  y  y  y  y  y  y  m  m  m  m  d  d  d  d  d
```

Where:

mm	1-12
dd	1-31
yy	0-119 (1980-2099)

The date is stored with the least significant byte first.

The file name in FileName can contain global characters.

File date/time of creation and file date/time of last access are not supported in this release and are returned as zeros.

Purpose

DosFindNext locates the next set of directory entries that match the name specified in the previous DosFindFirst or DosFindNext call.

Calling Sequence

```
EXTRN  DosFindNext:FAR

PUSH   WORD    DirHandle     ;Directory handle
PUSH@  OTHER   ResultBuf     ;Result buffer
PUSH   WORD    ResultBufLen  ;Result buffer length
PUSH@  WORD    SearchCount   ;Number of entries to find
CALL   DosFindNext
```

Where

DirHandle
is the handle associated with a previous DosFindFirst or DosFindNext function call.

ResultBuf
is where the file system returns the results of the qualified directory search. The information returned is as accurate as the most recent DosClose or DosBufReset.

2 bytes -	File date of creation
2 bytes -	File time of creation
2 bytes -	File date of last access
2 bytes -	File time of last access
2 bytes -	File date of last write
2 bytes -	File time of last write
2 bytes -	File end of data (low word)
2 bytes -	File end of data (high word)
2 bytes -	File allocation (low word)
2 bytes -	File allocation (high word)
2 bytes -	File attribute
1 byte -	Length of ASCIIZ name string
n bytes -	ASCIIZ name string.

DosFindNext – Find Next Matching File

ResultBufLen
is the length of ResultBuf.

SearchCount
is where the number of matching entries requested in ResultBuf are located. The file system stores the number of entries actually returned.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Family API Considerations

Some options operate differently in the DOS mode than they do in OS/2 mode. Therefore, the following restriction applies to DosFindNext when coding in the DOS mode:

- DirHandle must always equal 1.

Remarks

An error code returns if no matching files are found.

For more information refer to "DosFindFirst – Find First Matching File" on page 2-59.

File date/time of creation and file date/time of last access are not supported in this release and are returned as zeros.

DosFlagProcess – Set Process External Event Flag

Purpose

DosFlagProcess allows one process to set an “external event” flag for another.

Calling Sequence

```
EXTRN  DosFlagProcess:FAR

PUSH   WORD    ProcessID     ;Process ID to flag
PUSH   WORD    ActionCode    ;Indicate to flag descendants
PUSH   WORD    Flagnum       ;Flag number
PUSH   WORD    Flagarg       ;Flag argument
CALL   DosFlagProcess
```

Where

ProcessID

is the ID of the process or root process of the process tree, for which the flag is to be set.

ActionCode

indicates whether to flag descendant processes in addition to the process indicated by ProcessID.

If value = 0
: the indicated process and all its descendants processes (except detached processes), will be flagged. The indicated process must be the current process, or must have been created by the current process as a non-detached process. If the indicated process terminates, its descendants are still flagged.

If value = 1
: only indicated process will be flagged. Any process can be specified.

Flagnum

is the number of the flag to be set:

0 = flag A
1 = flag B
2 = flag C

DosFlagProcess – Set Process External Event Flag

Flagarg
is an argument passed to indicated processes.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

User flags are signals whose action is defined by the user. By default, a user flag is ignored by a process. A process can use DosSetSigHandler to install a signal handler for the signal number corresponding to the user flag (SIGPFA is the signal number corresponding to user flag A etc.). The process is alerted, via the signal mechanism, when it has been flagged. The process will then be alerted, via the signal mechanism, when it has been flagged. A process can also specify that the flag action is to be ignored and that an error code is to be returned to the flagger.

Purpose

DosFreeModule frees the reference to a dynamic link module for a process. When the dynamic link module is no longer needed by any process, the module is freed from system memory.

Calling Sequence

```
EXTRN  DosFreeModule:FAR

PUSH   WORD    ModuleHandle  ;Module handle
CALL   DosFreeModule
```

Where

ModuleHandle
is the handle returned by DosLoadModule for the dynamic link module to be freed.

Returns

IF   AX = 0 then NO error

ELSE AX = error code

Remarks

The module identified by the handle must be loaded through DosLoadModule. An error is returned if the handle is invalid.

When a function completes, the module handle is no longer valid and is not used to reference the dynamic link module. Procedure entry addresses returned for this module are also no longer valid and cause a protection fault if they are invoked.

DosFreeSeg – Free Segment

Purpose

DosFreeSeg deallocates a memory segment.

Calling Sequence

```
EXTRN  DosFreeSeg:FAR

PUSH   WORD    Selector      ;Selector
CALL   DosFreeSeg
```

Where

Selector
is the selector of the segment to be freed.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Family API Considerations

Some options operate differently in the DOS mode than they do in OS/2 mode. Therefore, the following restriction applies to DosFreeSeg when coding in the DOS mode:

If DosFreeSeg is issued on a CSAliased segment it deallocates the associated memory. This is inconsistent with the OS/2 mode, because DosFreeSeg must be performed on both the original and CSAliased selectors.

Remarks

DosFreeSeg is designed to free memory segments: shared segments, unshared segments, memory allocated by DosAllocSeg or DosAllocHuge and CS alias created by DosCreateCSAlias.

The CS alias selector is not affected if the data selector for the aliased segment is passed to DosFreeSeg. Procedures in this segment can continue to be referenced.

DosFreeSeg decrements the reference count for shared segments. When the reference count is 0, the memory is deallocated.

DosGetCollate – Get Collate Table

Purpose

DosGetCollate obtains a collating sequence table (for characters 00H through FFH) that resides in the country information file. It is used by the SORT utility to sort text according to the collating sequence.

Calling Sequence

```
EXTRN DosGetCollate:FAR

PUSH   WORD    Length         ;Length of data area provided
PUSH@  OTHER   Structure      ;Input data structure
PUSH@  OTHER   MemoryBuffer   ;Data area to contain the
                              ; collate table
PUSH@  WORD    DataLength     ;Length of table
CALL  DosGetCollate
```

Where

Length
is the byte length of the data area (MemoryBuffer) provided by the caller. A length value of 256 is sufficient.

Structure
is a two word input data structure where:

word 0
Country Code (0 - default country)
word 1
Code Page ID (0 - current process code page).

MemoryBuffer
is the area where the collating table is returned. This memory area is provided by the caller. The size of the area is provided by the input parameter Length and should be at least 256 bytes. If it is too small to hold all the available information then as much information as possible is provided in the available space (in the order in which the data would appear). If the amount of returned data does not fill the memory area provided by the caller, the unaltered memory is set at 0. The buffer format for the returned information follows:

1 Byte = sort weight of ASCII (0)
1 Byte = sort weight of ASCII (1)

Additional values in collating order:

1 Byte = sort weight of ASCII (255)

DataLength
is where the length in bytes of the collate table is returned.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

The returned country dependent information may be for the default country and current process code page or for a specific country and code page. For more information "DosSetCp — Set Code Page" on page 2-222.

DosGetCp – Get Process Code Page

Purpose

DosGetCp allows a process to query the current process code page and the prepared system code pages.

Calling Sequence

```
EXTRN   DosGetCp:FAR

PUSH   WORD    Length         ;Length of list
PUSH@  OTHER   CodePageList   ;List (returned)
PUSH@  WORD    DataLength     ;Length of returned list
Call   DosGetCp
```

Where

Length
is the length in bytes (should be at least 2), of CodePageList.

CodePageList
is the area where the code page information is returned. If CodePageList length is too small to hold all the available information, then as much information as possible is provided in the available space. The format of the information returned in this list is:

1 word = Current process code page
N words = Other prepared system code pages.

DataLength
is the length in bytes of the data returned.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Family API Considerations

Some options operate differently in the DOS mode than they do in OS/2 mode. Therefore, the following restriction applies to DosGetCp when coding in the DOS mode:

The current process code page, and at most one prepared system code page is returned.

Remarks

If the process code page identifier was previously set by DosSetCp or inherited by a process, it is returned to the caller. The current process code page identifier is returned by a two byte input list.

DosGetCtryInfo – Get Country Information

Purpose

DosGetCtryInfo obtains country dependent formatting information that resides in the country information file.

Calling Sequence

```
EXTRN DosGetCtryInfo:FAR

PUSH   WORD    Length         ;Length of data area provided
PUSH@  OTHER   Structure      ;Input data structure
PUSH@  OTHER   MemoryBuffer   ;Data area to be filled by the function
PUSH@  WORD    DataLength     ;Length of data (returned)
CALL   DosGetCtryInfo
```

Where

Length
is the byte length of the data area (MemoryBuffer) provided by the caller. This length should not be less than 38 bytes.

Structure
is a two word input data structure where:

word 0
Country Code (0 - default country)
word 1
Code Page ID (0 - current process code page).

MemoryBuffer
is the area where the country dependent information is returned. This memory area is provided by the caller. The size of the area is provided by the input parameter Length and should be at least 38 bytes. If it is too small to hold all the available information as much information as possible is provided in the available space. If the amount of data returned is not enough to fill the memory area provided by the caller the memory that is unaltered by the available data is set at 0. The format of the information returned in this buffer is:

1 Word -	Country Code
1 Word -	Code Page ID
1 Word -	Date format: 0 = mm/dd/yy, 1 = dd/mm/yy, 2 = yy/mm/dd
5 Bytes -	Currency indicator, null terminated
2 Bytes -	Thousands separator, null terminated
2 Bytes -	Decimal separator, null terminated
2 Bytes -	Date separator, null terminated
2 Bytes -	Time separator, null terminated
1 Byte -	Bit field for currency format
Bit 0 -	1 = currency indicator follows money value and 0 = currency indicator precedes money value.
Bit 1 -	Number of spaces (0 or 1) between currently indicator and money value.
Bit 2 -	When this bit is set, ignore first two bits; currency indicator replaces decimal indicator.
1 Byte -	Binary number of decimal places used in currency indication.
1 Byte -	Time format for file directory presentation: • Bit 0 = 1 - 24 hour • Bit 0 = 0 - 12 hour with “a” or “p”.
2 Words -	Reserved (set to 0).
2 Bytes -	Data list separator, null terminated.
5 Words -	Reserved (set to 0).

DataLength
is where the length in bytes of the country dependent information is returned.

Returns

IF AX = 0 then NO error

ELSE AX = error code

DosGetCtryInfo – Get Country Information

Family API Considerations

Some options operate differently in the DOS mode than they do in OS/2 mode. Therefore, the following restrictions apply to DosGetCtryInfo when coding in the DOS mode:

Not all country information is available in DOS 3.3.

Remarks

The returned country dependent information may be for the default country and current process code page or for a specific country and code page. For more information on code page, see "DosSetCp – Set Code Page" on page 2-222.

Purpose

DosGetDateTime gets the current date and time maintained by the operating system.

Calling Sequence

```
EXTRN  DosGetDateTime:FAR

PUSH@  OTHER   DateTime      ;Date/time structure (returned)
CALL   DosGetDateTime
```

Where

DateTime
is a structure that receives the following data items:

BYTE 0	-Hour is current hour
BYTE 1	-Minute is current minute
BYTE 2	-Second is current second
BYTE 3	-Hundredth is current hundredths of a second
BYTE 4	-Day is current day
BYTE 5	-Month is current month
WORD 6	-Year is current year
WORD 8	-TimeZone is minutes from UTC (Universal Time Coordinate)
BYTE 10	-DayofWeek is the current day of the week.

Note: The numbers following BYTE and WORD in the description of the above structure, represent decimal offset values from the beginning of the structure.

DosGetDateTime – Get Current Date and Time

Returns

AX = 0

Remarks

The DayofWeek value is based on Sunday equal to 0. TimeZone is the difference in minutes between the current time zone and UTC. This number is positive if it is earlier than UTC and negative if it is later than UTC. For eastern standard time, this value is 300 (5 hours earlier than UTC).

The application need not call this function to obtain the date or time. The address of memory containing a continuously updated date and time is obtained from the DosGetInfoSeg function. Applications written to the family API cannot depend on the availability of DosGetInfoSeg.

DosGetDBCSEv – Get DBCS Environmental Vector

Purpose

DosGetDBCSEv obtains a DBCS (double byte character set) environmental vector that resides in the country information file.

Calling Sequence

```
EXTRN DosGetDBCSEv:FAR

PUSH   WORD    Length        ;Length of data area provided
PUSH@  OTHER   Structure     ;Input data structure
PUSH@  OTHER   MemoryBuffer  ;Data area to contain
                             ; the information
CALL   DosGetDBCSEv
```

Where

Length

is the byte length of the data area (MemoryBuffer) provided by the caller. This value should be at least 10.

Structure

is a two word input data structure where:

word 0
  Country Code (0 - default country)
word 1
  Code Page ID (0 - current process code page).

MemoryBuffer

is where the country dependent information for the DBCS environmental vector is returned. This memory area is provided by the caller. The size of the area is provided by the input parameter Length. If it is too small to hold all the available information then as much information as possible is provided in the available space The format of the information returned in this buffer is:

2 Bytes - First range definition for DBCS lead byte values

- Byte 1 = Binary start value (inclusive) for range one
- Byte 2 = Binary stop value (inclusive) for range one

DosGetDBCSEv – Get DBCS Environmental Vector

2 Bytes - Second range definition

- Byte 1 = Binary start value for range two
- Byte 2 = Binary stop value for range two

2 Bytes - Nth range definition

- Byte 1 = Binary start value for Nth range
- Byte 2 = Binary stop value for Nth range

2 Bytes Two bytes of binary 0 terminate list.

For example:

```
DB   81H,9FH
DB   E0H,FCH
DB   0,0
```

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

The returned DBCS environmental vector may be for the default country and current process code page or for a specific country and code page. For more information on code page see "DosSetCp – Set Code Page" on page 2-222.

DosGetEnv – Get Address of Process Environment String

Purpose
DosGetEnv returns the address of the process environment string for the current process.

Calling Sequence
```
EXTRN  DosGetEnv:FAR

PUSH@  WORD    EnvSegment    ;Selector (returned)
PUSH@  WORD    CmdOffset     ;Command line offset (returned)
CALL   DosGetEnv
```

Where
EnvSegment
is where the selector for the environment segment is returned.

CmdOffset
is where the offset to the command line within the environment segment is returned.

Returns
IF   AX = 0 then NO error

ELSE AX = error code

Remarks
This call can be used by library routines that need to determine the environment for the current process.

DosGetHugeShift – Get Shift Count

Purpose

DosGetHugeShift returns a shift count used to derive the selectors that address memory allocated with DosAllocHuge.

Calling Sequence

```
EXTRN  DosGetHugeShift:FAR

PUSH@  WORD    ShiftCount    ;Shift Count (returned)
CALL   DosGetHugeShift
```

Where

ShiftCount
is where the shift count is returned.

Returns

AX = 0

Remarks

To compute the next sequential selector in a huge memory area, take the value 1, shift it left by the number of bits specified in shift count. Use the resulting value as an increment to add to the previous selector (using the selector returned by DosAllocHuge as the first selector) Refer to "DosAllocHuge – Allocate Huge Memory" on page 2-2 for more information regarding selector use.

DosGetInfoSeg – Get Address of System Variables Segment

Purpose
DosGetInfoSeg returns the address of a global and process local data segment used to determine the value of several items of general information.

Calling Sequence
```
EXTRN  DosGetInfoSeg:FAR

PUSH@  WORD    GlobalSeg      ;Global segment (selector)
PUSH@  WORD    LocalSeg       ;Local segment (selector)
CALL   DosGetInfoSeg
```

Where
GlobalSeg
is where the selector for the global information segment is returned.

LocalSeg
is where the selector for the local information segment is returned.

Returns
AX = 0

Remarks
Items of general interest are kept in the global information segment. Items of information specific to a particular process are kept in the local information segment. This information can be directly read by the application program. Both these segments are defined as read/only segments. The application program cannot modify this data.

DosGetInfoSeg – Get Address of System Variables Segment

Format of the Global Information Segment

Data Item		Size
TIME	DD	Time from 1-1-1970 in seconds
	DD	Milliseconds
	DB	Hours
	DB	Minutes
	DB	Seconds
	DB	Hundredths
	DW	Timezone (minutes from UTC, -1 = timezone is undefined)
	DW	Timer interval (units = 0.0001 seconds)
DATE	DB	Day
	DB	Month
	DW	Year
	DB	Day-of-week (0 = Sunday, 1 = Monday,...etc.)
VERSION	DB	Major version number
	DB	Minor version number
	DB	Revision letter
SYSTEM STATUS	DB	Current foreground session
	DB	Maximum number of sessions
	DB	Shift count for huge segments
	DB	OS/2 mode only indicator (0 = DOS mode and OS/2 mode, 1 = OS/2 mode only)
	DW	PID of last process to do a KbdCharIn or KbdGetFocus in foreground session.

Data Item		**Size**
SCHEDULER PARMS	DB	Dynamic variation flag (=1 if enabled, = 0 if absolute.)
	DB	Maximum wait (seconds)
	DW	Minimum timeslice (milliseconds)
	DW	Maximum timeslice (milliseconds)
BOOT DRIVE	DW	Boot drive number (1 = A, 2 = B,...etc.)
SYSTEM TRACE FACILITY TRACE FLAGS	DB	32 - System Trace Major Code flag bits. Each bit corresponds to a trace major code from 00H to FFH. The most significant bit (left-most) of the first byte, corresponds to major code 00H. Bit=0, trace disabled; Bit=1, trace enabled.

DosGetInfoSeg – Get Address of System Variables Segment

Format of the Local Information Segment

Data Item		Size
MISC	DW	Current process ID
	DW	Process ID of parent
	DW	Priority of current thread • High-Order byte = Priority class • Low-Order byte = Priority level
	DW	Thread ID of current thread
	DW	Session
	DW	Unused
	DW	Current process has keyboard focus. (-1 implies yes, 0 implies no)
	DB	Current process requires DOS mode. (=1 implies requires DOS mode).

The application program must be careful when referencing the date and/or time fields in the Global Information Segment. A timer interrupt can be received by the system in between the instructions that read the individual fields, changing the data in these fields. For example, if the time is currently 11:59:59 on Tuesday, 6/2/87, the program can read the hours field (23). A timer interrupt can now be received, changing the time to 12:00:00 on Wednesday, 6/3/87. The program will now read the rest of the time field (0 minutes) and the date field. The program would then think the current time and date is 11:00:00 on Wednesday, 6/3/87, which is incorrect.

The application program should read all time and date fields it is interested in as quickly as possible. It can then compare the least significant time field it is interested in (milliseconds, hundredths, seconds, or minutes) against the current value in the Global Information Segment. If the value has not changed, the rest of the information is valid. If the value has changed, the program time and/or date information should be read again, since the information was updated while the program was reading it.

DosGetMachineMode – Return Current Mode of Processor

Purpose

DosGetMachineMode returns the current mode of the processor, whether the processor is running in the DOS mode or the OS/2 mode. This allows an application to determine whether a dynamic link call is valid or not.

Calling Sequence

```
EXTRN  DosGetMachineMode:FAR

PUSH@  BYTE    MachineMode   ;Byte to receive mode
CALL   DosGetMachineMode
```

Where

MachineMode

is where the value to indicate the current processor mode is returned. This value may be:

0 = DOS mode
1 = OS/2 mode

Returns

AX = 0

Remarks

All dynamic link calls are available to an application if the MachineMode values indicate the program is in OS/2 mode. This method provides a self-tailoring application that allows the application to adapt to the execution environment by limiting or enhancing the functions it provides.

If the MachineMode values indicate the program is in DOS mode, the application is limited to a subset of dynamic link calls listed in the Family API.

DosGetMessage – System Message with Variable Text

Purpose

DosGetMessage retrieves a message from a message file and inserts variable information into the body of the message.

Calling Sequence

```
EXTRN  DosGetMessage:FAR

PUSH@  OTHER   IvTable      ;Table of variables to insert
PUSH   WORD    IvCount      ;Number of variables
PUSH@  OTHER   DataArea     ;Message buffer
PUSH   WORD    DataLength   ;Length of buffer
PUSH   WORD    MsgNumber    ;Number of the message
PUSH@  ASCIIZ  FileName     ;Message path and file name
PUSH@  WORD    MsgLength    ;Length of message (returned)
CALL   DosGetMessage
```

Where

IvTable
is a list of double-word pointers. Each pointer points to an *ASCIIZ* or null-terminated DBCS string (variable insertion text). 0 to nine strings can be present.

IvCount
is 0-9, the count of variable insertion text strings. If IvCount is 0, IvTable is ignored.

DataArea
is where the requested message is returned. If the message is too long to fit in the caller's buffer, as much of the message text will be returned as possible, with the appropriate error return code.

DataLength
is the length (in bytes) of the user's storage area.

MsgNumber
is the message number requested.

FileName
is the optional drive and path, and file name of the file, where the message can be found. If messages are bound to the .EXE file using MSGBIND utility, then filename is the name of the message file from which the messages will be extracted.

MsgLength
is where the length in bytes, of the message is returned.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Family API Considerations

Some options operate differently in the DOS mode than in OS/2 mode. Therefore, the following restriction applies to DosGetMessage when coding in the DOS mode:

If the message file name is not a fully qualified name, DosGetMessage searches the root directory of the default drive for the message file, instead of the root directory of the startup drive.

Remarks

If IvCount is greater than 9, DosGetMessage returns an error that indicates IvCount is out of range. If the numeric value of x in the %x sequence for %1-%9 is less than or equal to IvCount, text insertion by substitution for %x, is performed for all occurrences of %x in the message. Otherwise text insertion is ignored and the %x sequence is returned in the message unchanged. Text insertion is performed for all text strings defined by IvCount and IvTable.

Variable data insertion is not dependent on blank character delimiters nor are blanks automatically inserted.

For warning and error messages, the message ID (seven alphanumeric characters consisting of a three-character component ID concatenated with a four-digit message number) followed by a colon and a blank character are returned to the caller as part of the message text. (DosGetMessage determines the type of message based on the message classification generated in the output file of the MKMSGF utility).

The following is an example of a sample error message returned with the message ID:

```
SYS0100: File not found
```

DosGetMessage retrieves messages previously prepared by the utility MKMSGF to create a message file, or MSGBIND to bind a message segment to an .EXE file. DosGetMessage tries to retrieve the message from RAM in the message segment bound to the .EXE program. If the message cannot be found, DosGetMessage retrieves the message from the message file on DASD (direct access storage device, such as floppy diskette or fixed-disk). If the message cannot be found, then DosGetMessage retrieves the message from the message file on the fixed disk or diskette.

If DosGetMessage is unable to find the specified directory path, DosGetMessage searches the following directories for the message file:

- Directories listed in the DPATH statement (OS/2 mode only)
- Directories listed in the APPEND statement (DOS mode. only)

If a message cannot be retrieved because of a DASD error or file not found condition, a default message is placed in the user's buffer area. A message is placed in the buffer area based on the following error conditions:

- The message number cannot be found in the message file.
- The message file cannot be found.
- The system detected a disk error trying to access the message file, or the message file format is incorrect.
- IvCount is out of range.
- A system error occurred trying to allocate storage.

When these conditions occur, the default message allows the application program to issue a message and prompt that enables recovery opportunity.

The residency of the message in RAM (EXE bound) or on DASD is transparent to the caller and handled by DosGetMessage. In either case the message is referenced by message number and file name.

In order for DosGetMessage to be properly resolved, an application must be linked with DOSCALLS.LIB.

DosGetModHandle – Get Dynamic Link Module Handle

Purpose

DosGetModHandle returns a handle to a previously loaded dynamic link module.

Calling Sequence

```
EXTRN  DosGetModHandle:FAR

PUSH@  ASCIIZ  ModuleName    ;Module name string
PUSH@  WORD    ModuleHandle  ;Module handle (returned)
CALL   DosGetModHandle
```

Where

ModuleName
contains the dynamic link module name.

ModuleHandle
is where the handle for the dynamic link module is returned.

Returns

IF   AX = 0 then NO error

ELSE AX = error code

Remarks

This interface provides a mechanism for determining whether a dynamic link module is loaded.

The module name must match the name of the previously loaded module. If these names do not match, an error code is returned. The module name string must contain the 1-8 character module name, not the name of the file containing the module.

DosGetModName – Get Dynamic Link Module Name

Purpose
DosGetModName returns the fully qualified drive, path, FileName, and extension associated with a referenced module handle.

Calling Sequence
```
EXTRN  DosGetModName:FAR

PUSH   WORD    ModuleHandle  ;Module handle
PUSH   WORD    BufferLength  ;Buffer length
PUSH@  OTHER   Buffer        ;Buffer (returned)
CALL   DosGetModName
```

Where

ModuleHandle
is handle of the dynamic link module being referenced.

BufferLength
is the maximum length of the buffer, in bytes, where the name is stored.

Buffer
is the buffer where the fully qualified drive, path, filename, and extension of the dynamic link module is returned.

Returns
IF   AX = 0 then NO error

ELSE AX = error code

Remarks
An error is returned if the buffer is not large enough.

DosGetPID – Return Process ID

Purpose

DosGetPID returns the current process' process ID (PID), thread ID, and the PID of the process that created it.

Calling Sequence

```
EXTRN  DosGetPID:FAR

PUSH@  OTHER   ProcessIDsArea  ;Process IDs (returned)
CALL   DosGetPID
```

Where

ProcessIDsArea
is the area where the various IDs are returned.

Returns

AX = 0

Remarks

The format of the returned IDs in ProcessIDsArea is illustrated below:

WORD the current process' process ID
WORD the current thread ID
WORD the process ID of the parent.

The process ID may be used to generate uniquely named temporary files, or for communication via signals.

For an OS/2 mode process only, it is more efficient to obtain these variables from DosGetInfoSeg.

DosGetProcAddr – Get Dynamic Link Procedure Address

Purpose

DosGetProcAddr returns a far address to a desired procedure within a dynamic link module.

Calling Sequence

```
EXTRN  DosGetProcAddr:FAR

PUSH   WORD    ModuleHandle  ;Module handle
PUSH@  ASCIIZ  ProcName      ;Module name string
PUSH@  DWORD   ProcAddress   ;Procedure address (returned)
CALL   DosGetProcAddr
```

Where

ModuleHandle
is the handle of the dynamic link module where the procedure is located.

ProcName
is a name string that contains the referenced procedure name.

DosGetProcAddr for entries within the DOSCALLS module are only supported for ordinal references. References to the DOSCALLS module by name strings are not supported and will return an error. Dynamic link ordinal numbers for DOSCALLS routines are resolved by linking with DOSCALLS.LIB.

ProcAddress
is where the procedure address is returned.

Returns

IF AX = 0 then NO error

ELSE AX = error code

DosGetProcAddr — Get Dynamic Link Procedure Address

Remarks

If the selector portion of the pointer is null, the offset portion of the pointer is an explicit entry number (ordinal) within the dynamic link module. A 32-bit address, consisting of a selector and offset, is returned for a specified procedure.

Purpose

DosGetPrty gets the priority of a process or thread in the current process.

Calling Sequence

```
EXTRN  DosGetPrty:FAR

PUSH   WORD    Scope          ;Indicate scope of query
PUSH@  WORD    Priority       ;Priority (returned)
PUSH   WORD    ID             ;Process or thread ID
CALL   DosGetPrty
```

Where

Scope

is used to define the scope the request will have.

If value = 0
: the priority of the first thread of the indicated process is returned.

If value = 2
: the priority of the indicated thread is returned.

Priority

is a word where the base priority of the indicated process or thread is placed. The high-order byte of this word is set equal to the priority class. The low-order byte is set equal to the priority level.

ID is either a process ID (scope = 0) or a thread ID (scope = 2). If this operand is equal to 0, the current process or thread is assumed.

Returns

IF   AX = 0 then NO error

ELSE AX = error code

DosGetPrty – Get Process′s Priority

Remarks

If scope = 0 (process) the priority of the first thread of a process is returned. If that thread has terminated, the "ERROR_INVALID_THREAD_ID" error code will be returned.

The segment must have been allocated with DosAllocSeg with Flags bit 1 (0010b) set.

Purpose

DosGetSeg gets access to a shared memory segment.

Calling Sequence

```
EXTRN DosGetSeg:FAR

PUSH   WORD    Selector        ;Selector to access
CALL   DosGetSeg
```

Where

Selector
is used to get access to a segment.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

Any process that gains access to a discardable segment through DosGetSeg, may unlock the segment for discard through calls to DosUnlockSeg. For example, a process may allocate a discardable segment that is accessed by another process via DosGetSeg. The second process may call DosUnlockSeg on that selector until it is fully unlocked. The segment may be discarded later in the event memory is nearly full. If the first process accesses the segment, assuming it is still locked, it will fail. Locking is an attribute of the segment, not the processes using the segment.

The segment must have been allocated with DosAllocSeg with Flags bit one (0010b) set.

DosGetShrSeg – Access Shared Segment

Purpose

DosGetShrSeg accesses a shared memory segment previously allocated by another process.

Calling Sequence

```
EXTRN  DosGetShrSeg:FAR

PUSH@  ASCIIZ  Name          ;Name string
PUSH@  WORD    Selector      ;Selector (returned)
CALL   DosGetShrSeg
```

Where

Name
is the name string associated with the shared memory segment to be accessed. The name is an ASCIIZ string in the format of an OS/2 filename in a subdirectory called \SHAREMEM\, for example, \SHAREMEM\PUBLIC.DAT.

Selector
is where the selector for the shared memory segment is returned.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

The reference count for the shared segment is increased. The selector returned to the process that initially allocated it will be the same as that returned by the DosAllocShrSeg.

Purpose

DosGetVersion returns the OS/2 version number.

Calling Sequence

```
EXTRN  DosGetVersion:FAR

PUSH@  WORD    VersionWord   ;Version number(returned)
CALL   DosGetVersion
```

Where

VersionWord
is where the OS/2 version number is returned. The version is stored with the minor version first.

Returns

AX = 0

Remarks

None

DosGiveSeg – Give Access to Segment

Purpose

DosGiveSeg gives another process access to a shared memory segment.

Calling Sequence

```
EXTRN  DosGiveSeg:FAR

PUSH   WORD CallerSegSelector    ;Caller's segment selector
PUSH   WORD ProcessID            ;Process ID of recipient
PUSH@  WORD RecipientSegSelector ;Recipient's segment selector
                                 ; (returned)
CALL   DosGiveSeg
```

Where

CallerSegSelector
is the segment selector of the memory segment to be shared.

ProcessID
is the process ID of the process to receive access to the shared memory segment.

RecipientSegSelector
is where the recipient's segment selector to access the shared memory segment is returned.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

It is the caller's responsibility to pass the recipient's segment selector to the recipient using some form of interprocess communication.

Any process that gains access to a discardable segment through DosGiveSeg may unlock the segment for discard through calls to

DosUnlockSeg. For example, a process may allocate a discardable segment and give it to another process by way of DosGiveSeg. The second process may then call DosUnlockSeg on that selector until it is fully unlocked. The segment may be discarded later in the event memory is nearly full. If the first process accesses the segment, assuming it is still locked, it will fail. Locking is an attribute of the segment and not of the processes using the segment.

If the memory being given is a huge memory area allocated by DosAllocHuge, the CallerSegSelector must be the same selector as that returned by the corresponding DosAllocHuge request: for example, the selector for the first segment of the huge area. The returned RecipientSegSelector, which must be passed to the target process, is for the first segment in the recipient's address space. The recipient process must use DosGetHugeShift and calculate the selector values for the remaining segments of the area.

DosHoldSignal – Disable/Enable Signals

Purpose

DosHoldSignal temporarily disables or enables signal processing for the current process.

Calling Sequence

```
EXTRN  DosHoldSignal:FAR

PUSH   WORD    ActionCode    ;Indicate to Disable/Enable Signals
CALL   DosHoldSignal
```

Where

ActionCode

indicates to disable or enable signals intended for the current process:

If value = 0
: signals are enabled.

If value = 1
: signals are disabled.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Family API Considerations

Some options operate differently in the DOS mode than they do in the OS/2 mode. Therefore, the following restriction applies to DosHoldSignal when coding in the DOS mode:

The only signal recognized in the DOS mode is SIGINTR (Ctrl-C).

Remarks

DosHoldSignal with ActionCode = 1 causes signal processing (except SIGTERM and the numeric processor errors) to be postponed until a DosHoldSignal with ActionCode = 0 is issued. Any signals that occur while processing is disabled are recognized, but not accepted until signal recognition is enabled.

To allow for nesting of requests, a count of the number of outstanding DosHoldSignal requests with ActionCode = 1 are maintained.

DosHoldSignal is used by library routines, subsystems, and similar code that lock critical sections or temporarily reserve resources needed to prevent a signal from terminating a task.

Signals can be held for a short period and should be released and re-held, if necessary. Their guidelines for proper use are similar to hardware interrupt counterparts such as the CLI/STI instructions.

DosInsMessage — Insert Variable Text Strings In Message

Purpose

DosInsMessage inserts variable text string information into the body of a message. This is useful when messages are loaded before insertion text strings are known.

Calling Sequence

```
EXTRN  DosInsMessage:FAR

PUSH@  OTHER   IvTable        ;Table of variables to insert
PUSH   WORD    IvCount        ;Number of variables
PUSH@  OTHER   MsgInput       ;Address of input message
PUSH   WORD    MsgInLength    ;Length of input message
PUSH@  OTHER   DataArea       ;Buffer address to return message
PUSH   WORD    DataLength     ;Length of buffer (returned)
PUSH@  WORD    MsgLength      ;Length of updated message
CALL   DosInsMessage
```

Where

IvTable
is a list of double-word pointers. Each pointer points to an ASCIIZ or null terminated DBCS string (variable insertion text). 0 to nine strings can be present.

IvCount
is 0-9, the count of variable insertion text strings. If IvCount is 0, then IvTable is ignored.

MsgInput
is the input message.

MsgInLength
is the length in bytes of the input message.

DataArea
is the user storage where the system returns the updated message. If the message is too long to fit in the caller's buffer, as much of the message text as possible will be returned with the appropriate error return code.

DataLength
is the length (in bytes) of the user's storage area.

MsgLength
is where the actual length in bytes of the message is returned.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

If IvCount is greater than 9, DosInsMessage returns an error indicating that IvCount is out of range. A default message is also placed in the caller's buffer. Refer to "DosGetMessage – System Message with Variable Text" on page 2-88 for details on the default messages. If the numeric value of x in the %x sequence for %1-%9 is less than or equal to IvCount, then text insertion, by substitution for %x, is performed for all occurrences of %x in the body of the message. Otherwise text insertion is ignored and the %x sequence is returned unchanged in the message. Text insertion is performed for all text strings defined by IvCount and IvTable.

Variable data insertion does not depend on a blank character delimiter nor are blanks automatically inserted.

DosKillProcess – Terminate Process

Purpose

DosKillProcess flags a process to terminate and returns the termination code to its parent (if any).

Calling Sequence

```
EXTRN  DosKillProcess:FAR

PUSH   WORD    ActionCode     ;Indicate to flag descendant processes
PUSH   WORD    ProcessID      ;ID of process or root of process tree
CALL   DosKillProcess
```

Where

ActionCode

indicates whether to flag existing descendant processes in addition to the process indicated by ProcessID.

If value = 0
: the indicated process plus all descendant processes (except detached processes) are flagged for termination. The indicated process must be the current process, or it must have been created by the current process as a non-detached process. If the indicated process is terminated, its descendants will be flagged for termination.

If value = 1
: only the indicated process is flagged for termination. Any process may be specified.

ProcessID

is the process ID of the process or root process, of the process tree to be flagged for termination.

Returns

IF   AX = 0 then NO error

ELSE AX = error code

Remarks

DosKillProcess allows a process to send the termination signal SIGTERM to another process or group of processes. The default action of the system will be to terminate each of the processes. A process may intercept this action by installing a signal handler for SIGTERM (refer to "DosSetSigHandler – Set Signal Handler" on page 2-243). In such a case the program will clean up its files and execute DosExit. If there is no signal handler, the effect on the process will be the same as if one of its threads had done DosExit for the entire process. All file buffers are flushed and the handles opened by the process are closed, but any internal buffers managed by programs external to OS/2 will not be flushed. An example of such a buffer could be a C language libraries' internal character buffer.

The process' parent will get the "un-intercepted DosKillProcess" code returned when it does a DosCwait call.

DosLoadModule – Load Dynamic Link Module

Purpose

DosLoadModule loads a dynamic link module and returns a handle for the module.

Calling Sequence

```
EXTRN  DosLoadModule:FAR

PUSH@  OTHER   ObjNameBuf     ;Object name buffer
PUSH   WORD    ObjNameBufL    ;Length of object name buffer
PUSH@  ASCIIZ  ModuleName     ;Module name string
PUSH@  WORD    ModuleHandle   ;Module handle (returned)
CALL           DosLoadModule
```

Where

ObjNameBuf
is a buffer where the name of the object that contributed to the failure of DosLoadModule is returned.

ObjNameBufL
is the length, in bytes, of the buffer described by ObjNameBuf.

ModuleName
is a name string that contains the dynamic link module name.

ModuleHandle
is where the handle for the loaded dynamic link module is returned.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

If the file is an OS/2 dynamic link module, it is loaded, and a handle is returned. This handle must be passed to DosFreeModule to free the dynamic link module. This handle may be passed to DosGetProcAddr to get the addresses of procedures within the module, or passed to DosGetModName to get the fully qualified module name.

The module name string must contain the 1-8 character module name. The module is assumed to reside in a file whose name is ModuleName.DLL. This file must be within one of the directories specified in the library search path. (See configuration statement LIBPATH in the IBM Operating System/2 User's Reference.)

DosLoadModule may not be called from a dynamic library initialization routine if the module to be loaded or any module referenced by it contains a dynamic link library initialization routine.

DosLockSeg – Lock Segment in Memory

Purpose

DosLockSeg locks a discardable segment in memory.

Calling Sequence

```
EXTRN DosLockSeg:FAR

PUSH   WORD    Selector      ;Selector to lock
CALL  DosLockSeg
```

Where

Selector
is the selector of the segment to be locked.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

Discardable segments are useful for holding objects that are accessed for short periods of time and can be regenerated quickly if discarded. Examples are cache buffers for a data base package, saved bitmap images for obscured windows, or precomputed display images for a word processing application.

Allocating objects as discardable allows the memory manager to reclaim their space when the system is low on memory and they are not currently being accessed.

When a discardable segment is locked, it still can be moved and swapped. It can be swapped to disk if necessary, but will not be discarded until it is unlocked by DosUnlockSeg.

While locked, a segment may not be discarded. If a segment is discarded while unlocked, a subsequent DosLockSeg request for that segment will return an error code indicating the segment no longer exists. DosReallocSeg must allocate a fresh copy of the segment, and its contents must be regenerated by the caller.

DosLockSeg and DosUnlockSeg calls may be nested. If DosLockSeg is called multiple times to lock a segment the same number of calls must be made to DosUnlockSeg to unlock the segment. However, if a segment is locked 255 times it becomes permanently locked. Additional calls to DosLockSeg and DosUnlockSeg will have no effect on the segment's locked state.

Note that a call to DosAllocSeg with AllocFlags bit 2 (0100B) set or a call to DosReallocSeg, for a segment so allocated, will allocate the memory, and perform the same action as a call to DosLockSeg.

This function is valid only on segments allocated through DosAllocSeg with AllocFlags bit two (0100B) set.

DosMakePipe — Create Pipe

Purpose

DosMakePipe creates a pipe.

Calling Sequence

```
EXTRN  DosMakePipe:FAR

PUSH@  WORD    ReadHandle     ;Read handle (returned)
PUSH@  WORD    WriteHandle    ;Write handle (returned)
PUSH   WORD    PipeSize       ;Size to reserve for the pipe
CALL   DosMakePipe
```

Where

ReadHandle
is where the read handle for the pipe is returned.

WriteHandle
is where the write handle for the pipe is returned.

PipeSize
is the storage size, in bytes, to reserve for the pipe.

Returns

IF   AX = 0 then NO error

ELSE AX = error code

Remarks

Pipes are mechanisms used within a closely related group of processes. There are no control, permission mechanisms, or checks performed on operations to pipes.

When there is insufficient space in a pipe for the data being written, a requesting thread blocks until enough data is removed to allow the write request to be satisfied. If the parameter size is 0, then the pipe is created with a default size of 512 bytes.

When all users close the handles, a pipe is deleted. If two processes are communicating by a pipe and the processes reading the pipe ends, the next write gets the "write to a broken pipe" error code.

DosMemAvail – Get Size of Largest Free Memory Block

Purpose

DosMemAvail returns the size of the largest block of free memory.

Calling Sequence

```
EXTRN DosMemAvail:FAR

PUSH@  DWORD   MemAvailSize  ;Size available (returned)
CALL   DosMemAvail
```

Where

MemAvailSize
is where the size of the largest free block of memory is returned.

Returns

AX = 0

Remarks

DosMemAvail allows an application to determine how heavily used system memory is at a particular time. The returned value is a "snapshot" which may be valid only at the moment this function is issued and can be expected to change at any time due to system activity.

DosMkDir – Make Subdirectory

Purpose

DosMkDir creates a specified directory.

Calling Sequence

```
EXTRN  DosMkDir:FAR

PUSH@  ASCIIZ  DirName       ;New directory name
PUSH   DWORD   0             ;Reserved (must be 0)
CALL   DosMkDir
```

Where

DirName
is the ASCIIZ directory path name that specifies a drive.

Returns

IF   AX = 0 then NO error

ELSE AX = error code

Remarks

If any member of the directory path does not exist, the directory path is not created. On return, a new directory is created at the end of the specified path.

DosMonClose – Close Connection to Device Monitor

Purpose

DosMonClose terminates character device monitoring. All monitor buffers associated with this process are flushed and closed.

Calling Sequence

```
EXTRN  DosMonClose:FAR

PUSH   WORD    Handle        ;Handle from DosMonOpen
CALL   DosMonClose
```

Where

Handle
is the device handle returned from a previous DosMonOpen call.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

A single process may register one or more monitors with a character device using the same device handle returned from a previous DosMonOpen call. When DosMonClose is issued for a specific, opened device handle, all monitors for the current process registered with this handle terminate.

When DosMonClose is issued, the monitor loses access to the device data stream. Before issuing DosMonClose, monitor threads issuing DosMonRead and DosMonWrite calls should be terminated. DosMonRead calls issued after DosMonClose and DosMonWrite calls issued after DosMonClose return the errors ERROR_MON_BUFFER_EMPTY and ERROR_NOT_ENOUGH_MEMORY, respectively.

DosMonOpen – Open Connection to Device Monitor

Purpose
DosMonOpen gains access to a character device data stream.

Calling Sequence

```
EXTRN  DosMonOpen:FAR

PUSH@  ASCIIZ  Devname       ;Device name (returned)
PUSH@  WORD    Handle        ;Handle value (returned)
CALL   DosMonOpen
```

Where

Devname
is the device name string.

Handle
is where the handle for the monitor is returned. This value must be passed to DosMonReg to communicate with the device, and is passed to DosMonClose to close the connection to the monitor.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks
A process must issue DosMonOpen to get a handle to the specified device. A single process may register one or more monitors with the same or different data stream for one or more character devices.

Issue DosMonOpen before registering monitors. A process must get the handle to each device whose data stream it wishes to monitor. This handle is used by the process to communicate with the device driver in subsequent DosMonReg and DosMonClose calls.

Issue DosMonOpen once for each device. DosMonOpen must be issued before monitors are registered through DosMonReg, but does not have to be reissued before each subsequent DosMonReg call to register a monitor with the same device.

When DosMonClose is issued, all monitors in the process registered with the same device (that is, using the same handle) are closed.

Purpose

DosMonRead waits for and moves a data record from the input buffer of a registered character device monitor and places it in a private data area where the monitor can freely access it.

Calling Sequence

```
EXTRN  DosMonRead:FAR

PUSH@  OTHER   BufferI       ;Monitor input buffer
PUSH   WORD    WaitFlag      ;Block/Run indicator
PUSH@  OTHER   DataBuffer    ;Buffer into which records are read
PUSH@  WORD    Bytecnt       ;Input/output parm-#bytes
CALL   DosMonRead
```

Where

BufferI
is where the monitor input buffer is located.

WaitFlag
equals 0 if the monitor thread that issues DosMonRead wishes to block until a data record is available in its input buffer. WaitFlag equals 1 if the monitor thread that issues DosMonRead does not wish to block when its input buffer is empty.

DataBuffer
is the private data area where the data record taken from the monitor's input buffer is returned. The length of DataBuffer must be the entry value of Bytecnt.

Bytecnt
is the length of DataBuffer, on entry to DosMonRead. On the return from DosMonRead, Bytecnt specifies the number of bytes of data moved.

Returns

IF AX = 0 then NO error

ELSE AX = error code

DosMonRead – Read Input from Monitor Structure

Remarks

Device monitors are part of a data flow path through a device driver. They must respond rapidly to insure they do not delay I/O. This is especially important in the case of keyboard monitors.

A monitor process should be written so the threads that read and write the monitor data run at a high priority. Threads that read and write the monitor data should not perform operations, such as I/O or semaphore waits, which may cause considerable delay. A monitor process can have threads running at normal priorities to address these things.

Note: Each call to DosMonRead receives a single complete record. Multiple or partial records are not supported.

It is necessary to guarantee all data is cleaned out of the data stream at certain times. This is accomplished by flushing the data stream. A marked record or flush record is placed in the data stream by the device driver and passes through all monitors in the chain to allow them to perform a device-specific, prescribed activity. Placement of new data records in the data stream is suspended until the flush record reaches the device driver's buffer. Flush records must not be consumed by the monitor. That is, a monitor that receives a flush record on a DosMonRead call, must return it to the data stream by calling DosMonWrite.

Note: Refer to information for a specific character device driver in IBM Operating System/2 Technical Reference, Volume 1 regarding the type of flush support required for its monitors.

A data record consists of a flag word and the data itself. A flag word contains information meaningful to the monitors and devices whose data streams they are monitoring. The flag WORD is always the first word in the data record. The length of a data record that passes through a monitor chain must be less than or equal to the length of the device driver's monitor chain buffer minus 2 bytes.

Note: Refer to information for a specific device driver in the IBM Operating System/2 Technical Reference, Volume 1 for descriptions of flags and data in the records passing through its monitor chains.

On return from DosMonRead, Bytecnt is set to the number of bytes in the recently moved data record. Since this length may not equal the length of the private data area, the value of Bytecnt should be refreshed before the monitor reissues the DosMonRead call.

A monitor can only issue DosMonRead from an input buffer previously registered to an opened device by the same process. That is, a monitor registered by one process, or application, may not DosMonRead data from an input buffer of a monitor registered by another process.

If DosMonReg has not completed registration of a monitor's input and output buffers, DosMonRead returns ERROR_MON_INVALID_PARMS. A monitor does not have access to the device's data stream until DosMonReg completes successfully.

DosMonRead issued by a monitor thread after DosMonClose is issued by another thread in the same process returns the error code, ERROR_MON_BUFFER_EMPTY. When DosMonClose is issued, the monitor loses access to the device's data stream. The monitor should stop issuing DosMonRead calls before making the DosMonClose call.

Threads responsible for moving keystroke data through a monitor chain must pay special attention to the thread priority. Keystroke monitor threads must execute within the time critical priority class. More specifically these threads must execute at a priority level greater than or equal to the lowest level in the time critical priority class. The preferred level is level 0. This applies to any threads that read (DosMonRead), process or write (DosMonWrite) keystroke monitor data.

DosMonReg — Register Set of Buffers as Monitor

Purpose

DosMonReg establishes an input and output buffer structure to monitor an I/O stream for a character device.

Calling Sequence

```
EXTRN  DosMonReg:FAR

PUSH   WORD    Handle        ;Handle from DosMonOpen
PUSH@  OTHER   BufferI       ;Input buffer
PUSH@  OTHER   BufferO       ;Output buffer
PUSH   WORD    Posflag       ;Position flag
PUSH   WORD    Index         ;Index
CALL   DosMonReg
```

Where

Handle
is the device handle returned from a previous DosMonOpen call.

BufferI
is the monitor's input buffer. The monitor dispatcher moves data records into this buffer from the previous monitor, if any, in the chain or from the chain's first buffer. The monitor takes data from this buffer for filtering by calling DosMonRead.

BufferO
is the monitor's output buffer. The monitor places filtered data into this buffer by calling DosMonWrite. The monitor dispatcher moves data records from this buffer to the next monitor, if any, in the chain or into the device driver's monitor chain buffer, if the monitor is the last in the chain.

Posflag
is the position preference for locating the monitor in the monitor chain.

0 = no position preference
1 = monitor placed at beginning of chain
2 = monitor placed at end of chain.

Based on this specified position preference and on the position preference of those monitors from the same or different processes that have previously registered with the same chain, the monitor will be positioned within the chain as follows:

The first monitor in a chain that registers as 1 will be placed at the head of the chain. The next monitor that registers as 1 will follow the last monitor registered as 1, and so forth. Similarly, the first monitor that registers as 2 will be placed at the end of the chain. The next monitor that registers as 2 will be placed before the last monitor that registered as 2, and so forth. The first monitor that registers as 0 will be placed before the last monitor if any, that registered as 2. The next monitor that registers as 0 will be placed before the last monitor that registered as 0 and so forth.

Index
is a device specific value, denoting the data stream for the device to be monitored. Currently, the keyboard and mouse devices define this in terms of the session number (from one to maximum session number). "-1", for keystroke and mouse monitors, indicates the session of the calling thread. Refer to OS/2 Technical Reference, Volume 1 for further information on how data streams are defined for each device.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

DosMonOpen must first be issued to establish a connection between the device and monitor. The monitor requires this handle to register the pair of buffers with the device.

The monitor's input and output buffers must be in the same segment. The first word of each buffer must contain the buffer length, length word inclusive, when DosMonReg is issued. The length of each buffer must be greater than or equal to the length of the device driver's monitor chain buffer plus 20 bytes.

DosMonReg — Register Set of Buffers as Monitor

Note: Refer to information for a specific character device driver in the IBM Operating System/2 Technical Reference, Volume 1 for specification of the size of the device driver monitor chain buffer.

DosMonReg formats the monitor's input and output buffers as needed. The format and semantics of this buffer are not visible to the monitor application and are subject to change.

Until there is successful return from the DosMonReg call no character will enter the monitor's input buffer. Therefore, the monitor will not have access to the device's data stream. It is the application's responsibility to synchronize completion of DosMonReg and subsequent data stream monitoring with device input into the data stream.

Suppose an application required that all key strokes are monitored including 'type-ahead' key strokes. First the application issues DosMonOpen and DosMonReg to register a keystroke monitor. From the time the application is invoked through the time the keystroke monitor is registered and gains access to the data stream, the monitor sees no type-ahead key strokes. Since no monitor is registered at this time, type-ahead key strokes pass directly through to the device driver's API buffer. So that these type-ahead key strokes are not lost to the monitor, the monitor empties the keyboard's API buffer by making repeated KbdCharIn calls before issuing its first DosMonRead call. Refer to the OS/2 Programmer's Guide for more information on this technique.

Threads responsible for moving keystroke data through a monitor chain must pay special attention to the thread priority. Keystroke monitor threads must execute within the time critical priority class. More specifically these threads must execute at a priority level greater than or equal to the lowest level in the time critical priority class. The preferred level is level 0. The thread that makes the call to DosMonReg must also be executing in the time critical priority class.

Purpose

DosMonWrite moves a filtered data record from the monitor's private data area into the monitor's output buffer.

Calling Sequence

```
EXTRN  DosMonWrite:FAR

PUSH@  OTHER   BufferO       ;Monitor output buffer
PUSH@  OTHER   DataBuffer    ;Buffer from which records are taken
PUSH   WORD    Bytecnt       ;Number of bytes
CALL   DosMonWrite
```

Where

BufferO
is the monitor output buffer.

DataBuffer
is the monitor's private data area that contains a filtered data record of length Bytecnt. DosMonWrite moves this filtered data record into the monitor's output buffer.

Bytecnt
is the number of bytes in the data record.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

Device monitors are part of the data flow path through a device driver. They must respond rapidly so they do not delay I/O. This is especially important in the case of keyboard monitors.

A monitor process should be written so the threads that read and write the monitor data run at a high priority and so they never perform operations, such as I/O or semaphore waits, that might delay them. The monitor process can have other threads running at normal priorities to handle such things.

Each call to DosMonWrite places a single complete filtered data record into the data stream. The data sent by this call is considered to be a whole record.

It is necessary to ensure all data is cleaned out of the data stream at certain times. This technique is known as flushing. A specially marked record or flush record, is placed into the data stream by the device driver and passes through all monitors in the chain to allow them to perform a device-specific, prescribed activity. Placement of new data records into the data stream is suspended until the flush record reaches the device driver's buffer. Flush records must not be consumed by the monitor. Flush records must be returned to the data stream by issuing DosMonWrite.

Note: Refer to information for a specific device driver in the IBM Operating System/2 Technical Reference, Volume 1 regarding the type of flush support required for its monitors and restrictions on data record consumption.

A data record is defined as a word that contains flags meaningful to monitors and devices whose data streams they are monitoring. The flag word is always the first word in the data record. A monitor can modify the data record received on the last DosMonRead call before it issues DosMonWrite to return it to the device's data stream. However, the monitor should not alter the order within a data record.

Note: Refer to information for a specific device driver in the IBM Operating System/2 Technical Reference, Volume 1 for descriptions of flags and data in the records passing through its monitor chains. A monitor cannot write a data record into its output buffer that has a length greater than the length of the device driver's monitor chain buffer minus 2 bytes.

A monitor can only issue DosMonWrite to an output buffer already registered to an opened device. DosMonWrite returns the ERROR_MON_INVALID_PARMS return code if the DosMonReg call that registered the monitor's input and output buffers is not complete.

A monitor does not have access to the device's data stream until DosMonReg completes successfully.

DosMonWrite issued by a monitor thread after DosMonClose is issued by another thread in the same process returns the error, ERROR_NOT_ENOUGH_MEMORY. The monitor loses access to the device's data stream when DosMonClose is issued. The monitor should stop issuing DosMonWrite calls before making the DosMonClose call.

Threads responsible for moving keystroke data through a monitor chain must pay special attention to the thread priority. Keystroke monitor threads must execute within the time critical priority class. More specifically these threads must execute at a priority level greater than or equal to the lowest level in the time critical priority class. The preferred level is level 0. This applies to any threads that read (DosMonRead), process, or write (DosMonWrite) keystroke monitor data.

DosMove — Move a File

Purpose

DosMove moves a specified file.

Calling Sequence

```
EXTRN  DosMove:FAR

PUSH@  ASCIIZ  OldPathName    ;Old path name
PUSH@  ASCIIZ  NewPathName    ;New path name
PUSH   DWORD   0              ;Reserved (must be 0)
CALL   DosMove
```

Where

OldPathName
is the old path name of the file to be moved.

NewPathName
is the new path name of the file.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Family API Considerations

Some options operate differently in the DOS mode than they do in the OS/2 mode. Therefore, the following restriction applies to DosMove when coding in the DOS mode:

Files passed to OldPathName and NewPathName are truncated by the system in the DOS mode only. The operator must truncate all files passed to OldPathName and NewPathName in the OS/2 mode or an error code is returned.

Remarks

If a drive is used in the NewPathName string, it must be the same as the drive specified or implied in the OldPathName string. The directory paths need not be the same, allowing a file to be moved to another directory and renamed in the process. Global filename characters are not allowed in the filename.

DosMuxSemWait – Wait for One of N Semaphores to Clear

Purpose

DosMuxSemWait blocks a current thread until one of the specified semaphores clear.

Calling Sequence

```
EXTRN  DosMuxSemWait:FAR

PUSH@  WORD    IndexNbr      ;Index number of event (returned)
PUSH@  OTHER   ListAddr      ;Semaphore list
PUSH   DWORD   Timeout       ;Timeout
CALL   DosMuxSemWait
```

Where

IndexNbr

is where the index number of the semaphore that satisfies the wait request is returned.

ListAddr

is a list of event descriptors that define the semaphores to be waited on. The list is composed of a one word count of the number of semaphore descriptors in the list, followed by the semaphore descriptors. An application can wait on up to 16 semaphores at once.

```
Semaphore list format:

      DW    semcount    ; Number of semaphores
                        ; specified in list
N * / DW    0           ; Reserved, must be 0
    \ DD    ?    ; Semaphore handle
```

Timeout

is the count, in milliseconds, until the requesting task is to resume execution if none of the specified semaphores are cleared. The meaning of the specified values are:

If value = -1
: DosMuxSemWait waits indefinitely for a semaphore clear.

If value = 0
: there is an immediate return if no semaphores are clear.

If value > 0
value is the number of milliseconds to wait for a semaphore to clear.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

DosMuxSemWait checks a semaphore list. If any of the semaphores clear, DosMuxSemWait returns. If all are set, DosMuxSemWait blocks (until Timeout) until one of the semaphores clear. DosMuxSemWait returns when one of the semaphores on the list clears. This is known as an "edge-triggered" procedure. It is possible for the semaphore to reset before the thread returns to the caller from the DosMuxSemWait call.

Waiting threads using one of the functions DosSemRequest, DosSemSetWait, or DosSemWait, (known as "level-triggered" functions) may or may not resume. This decision depends on the scheduler's dispatch order and the activity of other threads in the system.

DosNewSize – Change File Size

Purpose

DosNewSize changes the size of a file.

Calling Sequence

```
EXTRN  DosNewSize:FAR

PUSH   WORD    FileHandle    ;File handle
PUSH   DWORD   FileSize      ;File's new size
CALL   DosNewSize
```

Where

FileHandle
is the handle of the file whose size is being changed.

FileSize
is the file's new size in bytes.

Returns

IF   AX = 0 then NO error

ELSE AX = error code

Remarks

DosNewSize can not change the size of Read/Only files. DosSetFileMode (Set File Mode) must be used to change a Read/Only file attribute to 0, and then change the file's size. Refer to "DosSetFileMode – Set File Mode" on page 2-230.

The value of new bytes in the extended file is undefined. The file system attempts to allocate the new size in a contiguous space on the media.

Purpose

DosOpen creates or opens a specified file.

Calling Sequence

```
EXTRN  DosOpen:FAR

PUSH@  ASCIIZ  FileName       ;File path name
PUSH@  WORD    FileHandle     ;File handle
PUSH@  WORD    ActionTaken    ;Action taken
PUSH   DWORD   FileSize       ;File primary allocation
PUSH   WORD    FileAttribute  ;File Attribute
PUSH   WORD    OpenFlag       ;Open function type
PUSH   WORD    OpenMode       ;Open mode of the file
PUSH   DWORD   0              ;Reserved (must be 0)
CALL   DosOpen
```

Where

FileName
is the path name of the file to be opened.

FileHandle
is where the system returns the file handle.

ActionTaken
is where the system returns a description of the action taken as a result of DosOpen.

0001H = file exists
0002H = file created
0003H = file replaced.

FileSize
is the file's new size in bytes.

FileAttribute
File attribute bits are defined as follows:

0001H = read only file
0002H = hidden file
0004H = system file
0010H = subdirectory

0020H	= file archive
0040H	= Reserved
0080H	= Reserved
0100H	= Reserved
0200H	= Reserved
0400H	= Reserved
0800H	= Reserved
1000H	= Reserved
2000H	= Reserved
4000H	= Reserved
8000H	= Reserved

These bits may be set individually or in combination. For example, an attribute of 0021H indicates a read-only file which should be archived.

OpenFlag
specifies the action to take if the file exists.

```
OpenFlag specification:

Low Order Byte

---- xxxx   action taken if file exists
---- 0000          fail
---- 0001          open file
---- 0010          replace file

xxxx ----   action taken if file doesn't exist
0000 ----          fail
0001 ----          create file

High Order Byte

'00000000'B                  reserved and set to 0
```

OpenMode
is an open mode that consists of the following bit fields:

- DASD Open flag
- Inheritance flag
- Write–through flag
- Fail–errors flag

- Sharing mode field
- Access field
- Reserved bit fields

Open Mode Bits - the bit field mapping is shown as follows:

```
Open Mode  5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0
  bits     D W F R R R R R I S S S R A A A
```

D - DASD Open

The file is opened as follows:

If D = 0 FileName represents a file to be opened in the normal way.

If D = 1 FileName is "Drive:" and represents a mounted disk or diskette volume to be opened for direct access.

W - File Write-through

The file is opened as follows:

If W = 0 Writes to the file may be run through the DOS buffer cache.

If W = 1 Writes to the file may go through the DOS buffer cache but the sectors are written (actual file I/O completed) before a synchronous write call returns. This state of the file defines it as a synchronous file.

I - Inheritance Flag

If I = 0 File handle is inherited by a spawned process resulting from a DosExecPgm call.

If I = 1 File handle is private to the current process.

This bit is not inherited by child processes.

F - Fail-Errors

Media I/O errors are handled as follows:

If F = 0 Reported through the system critical error handler.

If F = 1 Reported directly to the caller via return code.

This bit is not inherited by child processes. Media I/O errors generated through an IOCtl category eight function always get reported directly to the caller via return code. The Fail-Errors function applies only to non-IOCtl handle-based type file I/O calls.

R Reserved and must be 0 field.

S - Sharing Mode
The file sharing mode field defines what operations other processes may perform on the file.

If S = 001 Deny Read/Write access

If S = 010 Deny Write access

If S = 011 Deny Read access

If S = 100 Deny neither Read or Write access (Deny None)

Any other value is invalid.

A - Access Mode
The file access is assigned as follows:

If A = 000	Read/Only access
If A = 001	Write/Only access
If A = 010	Read/Write access

Any other combinations are invalid. When opening a file, inform OS/2 what operations other processes can perform on this file (sharing mode). If it is permissible for other processes to continue to read this file while the process is operating, specify Deny Write.

If a read/write file is opened with read/write access, an open request from another process will fail unless both processes use sharing mode Deny None. However, if the file is read-only and the file is opened with read-only access, an open request from another process will fail unless both processes used sharing modes of Deny None or Deny Write.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Family API Considerations

Some options operate differently in the DOS mode than they do in the OS/2 mode. Therefore, the following restrictions apply to DosOpen when coding in the DOS mode:

- Inheritance Flag is not supported.
- WriteThroughFlag must be set to 0.
- FailErrorsFlag must be set to 0.
- Share Mode has meaning only if SHARE is loaded, ignored if SHARE is not loaded.
- Access field has meaning only if SHARE is loaded, ignored if SHARE is not loaded.
- Access mode has meaning only if SHARE is loaded, ignored if SHARE is not loaded.
- FileName files passed to DosOpen must be truncated by the operator or an error code is returned. In the OS/2 mode, the system does the truncation.

When a read/only file is created it is always opened in compatibility mode, and is given read/write access.

Remarks

The read/write pointer is set at the first byte of a file. Issue DosChgFilePtr to change the read/write pointer.

The following example illustrates how DosScanEnv and DosSearchPath could be used to provide DosOpen with path searching:

Let DPATH be an environment variable in the environment segment of the process.

```
"DPATH=c:\sysdir;c:\init" /* in the
           environment */
```

The following two code fragments are equivalent:

```
DosScanEnv("DPATH", &PathRef);

DosSearchPath(0, /* Path Source Bit = 0 */

PathRef, "myprog.ini", &ResultBuffer,
ResultBufLen);

DosOpen(ResultBuffer, ...  );

DosSearchPath(2, /* Path Source Bit = 1 */

"DPATH", "myprog.ini", &ResultBuffer,
ResultBufLen);

DosOpen(ResultBuffer, ...  );
```

Issue DosQFileInfo to obtain the file's date and time. Issue DosSetFileInfo to set date and time, its attribute can be obtained through DosQFileMode.

The FileSize parameter affects the size of the file only when it is created or replaced. If an existing file is opened, FileSize is ignored.

The DASD Open bit parameter is the Direct I/O flag. It provides an access mechanism to a disk or diskette volume independent of the file system. This mode should only be used by systems programs and not by application programs. This mode of opening the currently mounted volume on the drive is used to return a handle to the caller that represents the logical volume as a single file. To block other processes from accessing the logical volume, the caller must issue DosDevIOCtl Category 8, sub-function 0 which requires the file handle for the logical volume returned by DosOpen.

DosOpen and DosSetFHandState can set the file handle state bits. An application can issue DosQFHandState to query the file handle state bits and the Open Mode field. Use the returned file handle for subsequent input and output to the file. The value of new bytes in the extended file is undefined.

When a critical error occurs that the application cannot handle, it must reset critical error handling to be done by the system. Issue

DosSetFHandState and reissue the I/O to accomplish this. The expected critical error reoccurs and is passed to the system critical error handler. The instant in time at which the effect of the DosOpen is visible at the application level it is unpredictable when asynchronous I/O is pending. This is the recommended action to take and is not done automatically.

Notes:

- DosOpen opens any normal or hidden file whose name matches the name specified.
- A multitasking system must be able to use semaphores to create and manage files. DosOpen may be used as a test and set semaphore when used to create a new file.
- When a file is closed, any sharing restrictions placed on it by the open are canceled.
- The file system attempts to allocate the new size in a contiguous space on the media.
- FileAttribute can not be set to Volume Label. Volume labels can not be opened.
- To set the file read/only attribute issue DosSetFileMode or the OS/2 ATTRIB command.
- If the file is inherited by a spawned process, all sharing and access restrictions are also inherited.
- If an open file handle is duplicated by DosDupHandle, all sharing and access restrictions are also duplicated.

Sharing Modes

- Deny Read/Write Mode (Exclusive)

  If a file is successfully opened in Deny Read/Write mode, access to the file is exclusive. A file currently open in this mode cannot be opened again in any sharing mode by any process until the file is closed.
- Deny Write Mode

  A file successfully opened in Deny Write sharing mode, prevents any other write access opens to the file (A = 001 or 010) until the file is closed. An attempt to open a file in Deny Write mode is unsuccessful if the file is currently open with a write access.

- Deny Read Mode

  A file successfully opened in Deny Read sharing mode, prevents any other read sharing access opens to the file (A = 000 or 010) until the file is closed. An attempt to open a file in Deny Read sharing mode is unsuccessful if the file is currently open with a read access.
- Deny None Mode

  A file successfully opened in Deny None mode, places no restrictions on the read/write accessibility of the file.

Purpose

DosOpenQueue opens a queue for the current process.

Calling Sequence

```
EXTRN  DosOpenQueue:FAR

PUSH@  WORD    OwnerPID       ;Queue owners' PID
PUSH@  WORD    QueueHandle    ;Handle of queue
PUSH@  ASCIIZ  QueueName      ;Queue name string
CALL   DosOpenQueue
```

Where

OwnerPID
is where the process ID of the queue owner is returned.

QueueHandle
is where the write handle of the queue is returned.

QueueName
is the name of the queue provided by a previous DosCreateQueue call.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

Before elements can be sent to a queue, the queue must be opened. DosCreateQueue opens the queue for that process.

When specifying the name for the queue, the ASCIIZ name string provided must include the prefix \QUEUES\ The process that issues the DosCreateQueue does not have to do a DosOpenQueue.

DosOpenSem – Open Existing System Semaphore

Purpose

DosOpenSem opens a system semaphore.

Calling Sequence

```
EXTRN  DosOpenSem:FAR

PUSH@  DWORD   SemHandle     ;Semaphore handle (returned)
PUSH@  ASCIIZ  SemName       ;Semaphore name string
CALL   DosOpenSem
```

Where

SemHandle
is where the handle of the system semaphore created by DosCreateSem is returned. This handle must be supplied by any requests directed to the same semaphore.

SemName
is the name of the system semaphore which was created using DosCreateSem.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

DosCreateSem must create the semaphore before DosOpenSem can open it. DosOpenSem only returns the handle of the semaphore, it does not test or change the value of the semaphore.

If a process with open semaphores issues a DosExecPgm, the new process inherits any open semaphore handles. All inherited semaphores are initially not owned by the child process, even if the parent owned them at the time of the Exec. (Only one process can own a semaphore at a time.)

Note: Under OS/2, system semaphores reside in a memory buffer rather than on a disk file. This means that when the last process which has a semaphore open (via DosCreateSem or DosOpenSem) exits, the semaphore disappears and must be recreated by its next user.

DosPeekQueue – Peek Queue

Purpose

DosPeekQueue retrieves an element from a queue without removing it from the queue.

Calling Sequence

```
EXTRN   DosPeekQueue:FAR

PUSH    WORD    QueueHandle     ;Handle of queue to read from
PUSH@   DWORD   Request         ;Request identification data
                                ; (returned)
PUSH@   WORD    DataLength      ;Length of element received
                                ; (returned)
PUSH@   DWORD   DataAddress     ;Address of element received
                                ; (returned)
PUSH@   WORD    ElementCode     ;Indicator of element received
                                ; (returned)
PUSH    WORD    NoWait          ;Indicate no wait if queue empty
PUSH@   WORD    ElemPriority    ;Priority of element (returned)
PUSH    DWORD   SemaphoreHandle ;Semaphore Handle
CALL    DosPeekQueue
```

Where

QueueHandle
is the handle of the queue from which to obtain an element.

Request
is filled in with the following information:

The first word is the PID of the process which added the element to the queue.

The second word is used for event coding by the application. The data in this word is the same as that furnished by the Request parameter on the DosWriteQueue request for the corresponding queue element. The value of this data is understood by the client thread and by the server thread. There is no special meaning to this data and the operating system does not alter the data.

DataLength
is where the length of the received data is returned.

DataAddress
is where the element being retrieved from the queue is returned.

ElementCode
indicates to start at the beginning of the queue or at a particular element. This field is set to:

0 by the application
to indicate start at the beginning of the queue
Non-0 by the DosPeekQueue function
to indicate the element returned, or by the owner to indicate "get next element."

NoWait
specifies the action to be performed when there are no entries on the queue.

If value = 0
the requesting thread waits.
If value = 1
the requesting thread does not wait.

ElemPriority
is the priority specified when the element is added to the queue. This is a numeric value in the range of zero to 15, with 15 being the highest priority.

SemaphoreHandle
is the handle of a semaphore which is to be cleared when the queue has data placed into it and NoWait = 1 is specified. The semaphore may be either a *RAM* or system semaphore.

If this handle is for a *RAM* semaphore, that semaphore must be in a segment shared between the queue owner's process and any process that issues a DosWriteQueue request to an associated queue.

If multiple threads are processing elements from the queue using a NoWait value = 1, the same semaphore must be provided on all DosPeekQueue or DosReadQueue requests.

DosPeekQueue — Peek Queue

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

DosPeekQueue retrieves elements from a specified queue without removing that element from the queue. If the queue is empty, and wait is specified, the thread is placed in a wait state waiting for an element to be added to the queue. If the NoWait option is selected, the request does not place the thread in a wait state, but returns with a code indicating there are no entries on the queue.

ElementCode is an indicator of the element to peek or read next. Following a peek request, ElementCode contains an identifier of the element which has been peeked The next DosPeekQueue which provides this identifier as the ElementCode parameter, returns the next element following the element indicated by ElementCode. A subsequent DosReadQueue request that provides this identifier reads the indicated element.

When the application program sets this field to 0, the next DosPeekQueue or DosReadQueue request accesses the first element in the queue.

Only the queue owner (the process which created the queue via DosCreateQueue) is allowed to issue this call. Any thread within that process may also issue DosPeekQueue calls to any queue owned by that process.

The semaphore provided by SemaphoreHandle would typically be used with a DosMuxSemWait request to wait on a queue or any of several other events. This operand is ignored if NoWait = 0 is specified.

Purpose

DosPFSActivate specifies the code page and font to make active for the specified printer and Process ID.

Calling Sequence

```
EXTRN DosPFSActivate:FAR

PUSH   WORD   SplHandle    ;Temporary Spool File handle
PUSH@  DWORD  BytesWritten ;Number of bytes written (returned)
PUSH@  ASCIIZ PrinterName  ;Printer name string
PUSH   WORD   CodePage     ;Code Page to make active
PUSH   WORD   FontID       ;Font ID to make active
PUSH   WORD   ProcessID    ;ProcessID
PUSH   DWORD  Reserved     ;Reserved, set to 0
CALL   DosPFSActivate
```

Where

SplHandle
is the file handle of the temporary spool file for which code page and font switching is being activated.

BytesWritten
is where the number of bytes written to the temporary spool file are returned.

PrinterName
is the name of the printer for which code page and font switching is being activated.

CodePage
specifies the code page to make active for the specified printer and process id.

FontID
specifies the font within the specified code page to make active for the specified printer and process id.

For download fonts, the FontID is that specified in the printer font file.

For cartridge fonts, the FontID is the number specified on the label of the cartridge and in the DEVINFO statement for the printer.

A value of 0 (0000h) for both the CodePage and FontId indicates that the hardware default code page and font should be made active.

A value of 0 for the font ID but not the code page indicates that any font ID is acceptable for the code pages.

ProcessID
specifies the process ID of the requester.

Reserved
is reserved and set equal to zero.

Returns

IF AX = 0 then NO error

ELSE AX = error code

For a description of the return values for DosPFSActivate, see the Remarks section below:

Remarks

DosPFSActivate is intended for use only by applications that replace the spooler as a print monitor and that do code page switching. Other applications should use printer IOCtls to manipulate printer code page switching.

DosPFSActivate is located in SPOOLCP.DLL (not in DOSCALLS.LIB) and requires an import statement in the module definition file. See the IBM Operating System/2 Programmer's Guide, Module Definition File Statements section for information regarding the import statement.

Return values are:

Value	*Meaning*
2	Code page not available
4	Font ID not available
9	Code page switcher internal error
10	Invalid printer name as input
13	Received code page request when code page switcher not initialized
15	PID table full. Cannot activate another entry
19	I/O error reading font file control sequence section
21	I/O error reading font file font definition block
23	I/O error while writing to temporary spool file
24	Disk full error while writing to temporary spool file
25	Bad spool file handle

DosPFSCloseUser – Close Font User Interface

Purpose

DosPFSCloseUser indicates to the Font Switcher that the specified process has closed its spool file. The Font Switcher may then free any resources being used to track code page and font switching for a process.

Calling Sequence

```
EXTRN DosPFSCloseUser:FAR

PUSH@  ASCIIZ PrinterName  ;Printer name string
PUSH   WORD   ProcessID    ;ProcessID
PUSH   DWORD  Reserved     ;Reserved, set to 0
CALL   DosPFSCloseUser
```

Where

PrinterName
is the name of the printer for which code page and font switching is being closed.

ProcessID
specifies the process ID of the requester.

Reserved
is reserved and set equal to 0.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

DosPFSActivate is intended for use only by applications that replace the spooler as a print monitor and that do code page switching. Other applications should use printer IOCtls to manipulate printer code page switching.

DosPFSActivate is located in SPOOLCP.DLL (not in DOSCALLS.LIB) and requires an import statement in the module definition file. Refer to the IBM Operating System/2 Programmer's Guide, Module Definition File Statements section for information regarding the import statement.

Return values are:

Value	*Meaning*
8	Attempted to close process ID not active
9	Code page switcher internal error
10	Invalid printer name as input
13	Received code page request when code page switcher not initialized

DosPFSInit – Initialize Code Page and Font

Purpose

DosPFSInit allows the Font Switcher to initialize code page and font switching for a specified printer.

Calling Sequence

```
EXTRN DosPFSInit:FAR

PUSH@  OTHER  CPHdw        ;Hdw Font definition list
PUSH@  ASCIIZ FontFileName ;File pathname of the
                           ;font file to be used
PUSH@  ASCIIZ PrinterType  ;Printer type string
PUSH@  ASCIIZ PrinterName  ;Printer name string
PUSH   WORD   Instances    ;Number of spool instances
PUSH   DWORD  Reserved     ;Reserved
CALL   DosPFSInit
```

Where

CPHdw

points to a list in the following format which specifies the Hardware code page and fonts the printer is equipped with:

Word 0

Number of definitions which follow

DWord 1 / n

Code page number (1st Word of DWord) and Font ID (2nd Word of DWord) for each hardware font in order corresponding to the hardware code page and font selection numbers (i.e. the first code page and font ID value corresponds to the default hardware font 0, second value corresponds to hardware font 1, third to hardware font 2, etc.. If the default hardware font is not known, 0 should be specified for the default code page and font).

FontFileName

is the pathname of the font file of the specified printer for which code page and font switching is being initialized.

PrinterType
is the printer type ID.

PrinterName
is the name of the printer for which code page and font switching is being initialized.

Instances
is the maximum number of different instances of use for which code page and font switching should be tracked. This value is advisory for the Font Switcher to be able to allocate enough resources for the specified number of instances to be tracked.

Reserved
is reserved and set equal to 0.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

DosPFSInit is intended for use only by applications that replace the spooler as a print monitor and that do code page switching. Other applications should use printer IOCtls to manipulate printer code page switching.

DosPFSInit is located in SPOOLCP.DLL (not DOSCALLS.LIB) and requires an import statement in the module definition file. Refer to the IBM Operating System/2 Programmer's Guide, Module Definition File Statements section for information regarding the import statement.

Return values are:

Value	***Meaning***
1	Code page switcher already initialized.
3	User entered too many ROMs in DEVINFO. Initialization continued with the rest.
6	Wrong or missing font file ID.
9	Code page switcher internal error.
10	Invalid printer name as input.
11	Printer type input does not match that in font file.
12	Could not get storage for control blocks.
14	Could not open font file during initialization.
17	Switcher reports too many PID entries.
19	I/O error reading font file control sequence section.
20	I/O error reading font file header.
21	I/O error reading font file font definition block.
22	Some fonts bad due to error in font file. Initialization continued

Purpose

DosPFSQueryAct queries the active code page and font for the specified printer and Process ID.

Calling Sequence

```
EXTRN  DosPFSQueryAct:FAR

PUSH@  ASCIIZ PrinterName ;Printer name string
PUSH@  WORD   CodePage    ;Code Page return
PUSH@  WORD   FontId      ;Font ID return
PUSH   WORD   ProcessID   ;ProcessID
PUSH   DWORD  Reserved    ;Reserved, set to 0
CALL   DosPFSQueryAct
```

Where

PrinterName
  is the name of the printer for which the active code page and font is being queried.

CodePage
  is where the currently active code page for the specified printer and process ID are returned.

FontId
  is where the currently active Font ID number for the specified printer and process ID are returned.

ProcessID
  specifies the process ID of the requester.

Reserved
  is reserved and set equal to 0. is reserved for future use.

Returns

IF   AX = 0 then NO error

ELSE AX = error code

DosPFSQueryAct — Query Active Font

Remarks

DosPFSQueryAct is intended for use only by applications that replace the spooler as a print monitor and that do code page switching. Other applications should use printer IOCtls to manipulate printer code page switching.

DosPFSQueryAct is located in SPOOLCP.DLL (not in DOSCALLS.LIB) and requires an import statement in the module definition file. Refer to the IBM Operating System/2 Programmer's Guide, Module Definition File Statements section for information regarding the import statement.

Return values are:

Value	*Meaning*
9	Code page switcher internal error.
10	Invalid printer name as input.
13	Received code page request when code page switcher not initialized.
16	Received request for process ID not in the PID table.

Purpose

DosPFSVerifyFont indicates whether the specified code page and font within that code page are available in the font file for the specified printer.

Calling Sequence

```
EXTRN DosPFSVerifyFont:FAR

PUSH@  ASCIIZ PrinterName  ;Printer name string
PUSH   WORD   CodePage     ;Code Page to validate
PUSH   WORD   FontId       ;Font Id to validate
PUSH   DWORD  Reserved     ;Reserved, set to 0
CALL   DosPFSVerifyFont
```

Where

PrinterName
is the name of the printer for which the code page and font switching setup is being queried.

CodePage
is the code page to validate. Values may be 0 to 65535.

FontId
is the Font ID to validate. Values may be 0 to 65535. A value of 0 indicates that any font within the specified code page is acceptable.

Reserved
is reserved and set equal to 0.

Returns

IF AX = 0 then NO error

ELSE AX = error code

DosPFSVerifyFont — Verify Font

Remarks

DosPFSVerifyFont is intended for use only by applications that replace the spooler as a print monitor and that do code page switching. Other applications should use printer IOCtls to manipulate printer code page switching.

DosPFSVerifyFont is located in SPOOLCP.DLL (not in DOSCALLS.LIB;) and requires an import statement in the module definition file. Refer to the IBM Operating System/2 Programmer's Guide, Module Definition File Statements section for information about the import statement.

Return values are:

Value	***Meaning***
2	Code page not available
4	Font ID not available
10	Invalid printer name as input.
13	Received code page request when code page switcher not initialized.

Purpose

DosPhysicalDisk obtains information on partitionable disks.

Calling Sequence

```
EXTRN   DosPhysicalDisk:FAR

PUSH    WORD     Function        ;Type of information
PUSH@   OTHER    DataPtr         ;Pointer to return buffer
PUSH    WORD     DataLen         ;Return buffer length
PUSH@   OTHER    ParmPtr         ;Pointer to user-supplied
                                 ; information
PUSH    WORD     ParmLen         ;Length of user-supplied
                                 ; information
CALL    DosPhysicalDisk
```

Where

Function

identifies the type of information on the partitionable disk(s) to obtain.

The functions currently supported are:

1 = obtain total number of partitionable disks
2 = obtain a handle to use with Category 9 IOCtls
3 = release a handle for a partitionable disk

DataPtr

indicates the location of a buffer in which the returned information is placed.

DataLen

specifies the length of the data buffer.

The output data for each function is described below. Note that all lengths are in bytes.

DosPhysicalDisk – Partitionable Disk Support

Function	DataLen	Returned Information
1	2	Total number of partitionable disks in the system, 1-based
2	2	Handle for the specified partitionable disk for the Category 9 IOCtls
3	0	None - Pointer must be 0.

ParmPtr
indicates the location of a buffer used for input parameters.

ParmLen
specifies the length of the parameter buffer.

The input parameters required for each function are described below. Note that all lengths are in bytes.

Function	ParmLen	Input Parameters
1	0	None / must be set to 0
2	string length	ASCIIZ string that specifies the partitionable disk
3	2	Handle obtained from Function 2

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

The ASCIIZ string used to specify the partitionable disk must be of the following format:

```
          number : <null byte>

 where

    number        specifies the partitionable
                  disk (1-based) number in
                  ASCII

    :             must be present

    >null byte>   the byte of 0 for the
                  ASCIIZ string
```

The handle returned for the specified partitionable disk can only be used with the DosDevIOCtl call for the Category 9 Generic IOCtl. Use of the handle for a physical partitionable disk is not permitted for handle-based file system function calls, such as DosRead or DosClose.

DosPortAccess – Request Port Access

Purpose

DosPortAccess requests or releases access to ports for I/O privilege.

Calling Sequence

```
EXTRN  DosPortAccess:FAR

PUSH   WORD    Reserved      ;0
PUSH   WORD    TypeOfAccess  ;Request or release
PUSH   WORD    FirstPort     ;First port number
PUSH   WORD    LastPort      ;Last port number
CALL   DosPortAccess
```

Where

Reserved
must be set to 0.

TypeOfAccess
indicates a request for or release of access to a port.

0 = request access
1 = release access

FirstPort
specifies either the starting (low-end) number in a contiguous range or a single port.

LastPort
specifies either the ending (high-end) number in a contiguous range or a single port. If only one port is being used FirstPort needs to be set to this port number and LastPort needs to be set to this port.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

Note that CLI/STI privilege is also granted automatically, there is no need to make an additional call to DosCLIAccess.

Applications that perform I/O to port(s) in IOPL segments must request port access from the operating system.

An application with no IOPL segments that accesses a device through a device driver or by an interface package such as VIO, will not need to gain port access: the device driver or interface package will be responsible for obtaining the necessary I/O access.

DosPtrace – Interface for Program Debugging

Purpose

DosPtrace provides an interface into the OS/2 kernel to facilitate program debugging.

Calling Sequence

```
EXTRN  DosPtrace:FAR

PUSH@  OTHER   PtraceB   ;Ptrace buffer (returned)
CALL   DosPtrace
```

Where

PtraceB

is the Ptrace command/data buffer. This buffer is used to communicate between the debug program and the DosPtrace routines.

Returns

AX = 0

Remarks

DosPtrace allows a parent process to control the execution of another process. Its intended use is directed toward the implementation of breakpoint debugging using a debugger. The program under test and, the program being debugged, must be executing in OS/2 mode.

To debug a process with multiple threads, DosPtrace allows the debugger to selectively suspend and resume the threads of a program being debugged. It also reads the registers of its individual threads.

A debug program must be able to read and write the instructions, data, and registers of the program being debugged to insert breakpoint instructions. When a process runs in OS/2 mode, one process cannot directly manipulate the address space of another process. OS/2 controls this address space through the use of the trace flag facility in DosExecPgm and the Ptrace buffer in DosPtrace.

Interface for Program Debugging

The steps to program debugging in a OS/2 mode follow:

1. The debug program issues DosExecPgm for the program to be debugged, and specifies the trace option.
2. The debug program calls DosPtrace with the TRC_C_Stop command to initialize the Ptrace Buffer.
3. The debug program sets up a Ptrace buffer with commands for inserting the breakpoints, and issues repeated DosPtrace requests as necessary.
4. The debug program sets up a Ptrace buffer with a command to begin execution and issues DosPtrace. This may be a TRC_C_SStep, or TRC_C_Go.
5. When the kernel DosPtrace program receives control from the program being debugged, it returns to the debug program with the Ptrace buffer set to the current register contents and with indicators of the reason for return.
6. The kernel DosPtrace program receives control at a breakpoint interrupt, at processor exceptions, or when the program ends.

To debug a process with multiple threads, set a field in the Ptrace buffer (Ptrace_B.TID) to the thread ID of the thread of interest. This causes the read/write register commands to receive only the register set of the specified thread.

Note: For a process with multiple threads, the address space is the same for all the threads in the process. When commands are issued to read/write memory locations or set breakpoints it affects all the threads in the process even though the command was issued with a specific thread ID.

The debugger may suspend and resume specific threads through use of the TRC_C_Freeze and TRC_C_Resume commands. Having only selected threads be affected by the breakpoints is useful for manipulating them while other threads are suspended.

When a process debugger terminates, the program being debugged also terminates. To accomplish this, an internal link between the debugger and the program being debugged is maintained. This link is established as a result of the first successful TRC_C_Stop command. Once established, this link can not be reset.

The program being debugged does not need to be a direct child. As a result, a small window of time between the DosExecPgm call and the first DosPtrace call, where if the debugger terminates, the program being debugged cannot be cleaned up. The system terminates the program being debugged in a few minutes.

Contents of the Ptrace Buffer:

Ptrace B			***Structure***
PID	DW	0	; Process ID of the process being debugged
TID	DW	0	; Thread ID of the process being debugged
Cmd	DW	0	; Request to DosPtrace, or DosPtrace result code
Value	DW	?	; Data to DosPtrace, or DosPtrace error code
OffV	DW	?	; Offset value
SegV	DW	?	; Segment value
MTE	DW	?	; Library Module handle
rAX	DW	?	; Registers AX thru SS
rBX	DW	?	
rCX	DW	?	
rDX	DW	?	
rSI	DW	?	
rDI	DW	?	
rBP	DW	?	

Ptrace B			***Structure***
rDS	DW	?	
rES	DW	?	
rIP	DW	?	
rCS	DW	?	
rF	DW	?	
rSP	DW	?	
rSS	DW	?	
Ptrace_B ENDS			

DosPtrace Commands:

PTrace_B.Cmd must contain one of the following commands upon entrance to DosPTrace:

TRC_C_Null	EQU 0		; Invalid
TRC_C_ReadMem_I	EQU 1		
TRC_C_ReadMem_D	EQU 2		
TRC_C_ReadMem	EQU	TRC_C_ReadMem_I	
TRC_C_ReadReg	EQU 3		
TRC_C_WriteMem_I	EQU 4		

TRC_C_WriteMem_D	EQU 5		
TRC_C_WriteMem	EQU	TRC_C_WriteMem_I	
TRC_C_WriteReg	EQU 6		
TRC_C_Go	EQU 7		
TRC_C_Term	EQU 8		
TRC_C_SStep	EQU 9		
TRC_C_Stop	EQU 10		; Ini-tialize
TRC_C_Freeze	EQU 11		
TRC_C_Resume	EQU 12		
TRC_C_NumToSel	EQU 13		
TRC_C_GetFPRegs	EQU 14		
TRC_C_SetFPRegs	EQU 15		
TRC_C_GetLibName	EQU 16		

Commands and Required Input:

A command is issued by placing the command number in Ptrace_B.Cmd, and other required information into a Ptrace buffer, and calling DosPtrace with that buffer.

All of the commands require that Ptrace_B.PID be the PID of the process to debug.

TRC_C_Null	: Not a valid command

Memory Operations:

For the following commands, SegV:OffV is the affected location, and Ptrace_B.Value contains the value to write to or that was read from the debugger's memory.

TRC_C_ReadMem_I	: Read instruction
TRC_C_ReadMem_D	: Read data
TRC_C_ReadMem	: Read any memory
TRC_C_WriteMem_I	: Write instruction
TRC_C_WriteMem_D	: Write data
TRC_C_WriteMem	: Write to any memory

Register / Thread Operations:

For the following commands, Ptrace_B.TID must contain the thread ID of the thread in question.

TRC_C_ReadReg	: Examine thread's registers
TRC_C_WriteReg	: Write thread's registers
TRC_C_Freeze	: Suspend a thread
TRC_C_Resume	: Resume a suspended thread

Command Operations:

For the following commands, the Ptrace_B.PID must be valid. The Ptrace_B registers are ignored for these commands. For TRC_C_Go and TRC_C_SStep, any thread may gain control first. The TRC_C_Term command terminates the program being debugged.

TRC_C_Go	: Run debuggee
TRC_C_Term	: Terminate debuggee
TRC_C_SStep	: Run one instruction
TRC_C_Stop	: Initialize PTrace buffer

Library Support:

For TRC_C_NumToSel, Ptrace_B.Value should be set to the segment number on entrance, and a valid selector on exit. Also, Ptrace_B.MTE should be set to the module's handle. The MTE identifies the different library files in the program being debugged.

For TRC_C_GetLibName, SegV:OffV should point to a buffer where the name of the library will be returned. PTrace_B.Value should hold the library's module handle (MTE).

TRC_C_NumToSel	: Convert Segment number to selector
TRC_C_GetLibName	: Return name of module

Floating Point Support:

For the following two commands, SegV:OffV must contain a pointer to a 94 byte buffer to be used to read/write the floating point registers from/to.

The layout of this area is described in the NPX287 manual under the heading FSAVE/FRSTOR memory layout.

TRC_C_GetFPRegs	: Read floating point regs.
TRC_C_SetFPRegs	: Write floating point regs.

DosPtrace Return Codes:

When DosPtrace returns to the debug program, the result is placed in Ptrace_B.Cmd, and reflects the reason for the return.

The values returned are:

TRC_C_SUC_ret	EQU 0	; Success
TRC_C_ERR_ret	EQU -1	; Error
TRC_C_SIG_ret	EQU -2	; Signal
TRC_C_TBT_ret	EQU -3	; Single Step
TRC_CB_PT_ret	EQU -4	; Breakpoint
TRC_C_NMI_ret	EQU -5	; Parity Error
TRC_C_KIL_ret	EQU -6	; Process dying
TRC_C_GPF_ret	EQU -7	; GP fault
TRC_C_LIB_ret	EQU -8	; Library load

TRC_C_FPE_ret	EQU -9	; FP error

If Ptrace_B.Cmd is returned as TRC_C_ERR_ret, Ptrace_B.Value is set to one of the following:

TRACE_BAD_COMMAND	EQU 1
TRACE_CHILD_NOT_FOUND	EQU 2
TRACE_CHILD_UNTRACEABLE	EQU 5

If Ptrace_B.Cmd is returned as TRC_C_SIG_ret, the process is about to receive a signal.

If Ptrace_B.Cmd is returned as TRC_C_KIL_ret, the process is about terminate.

If Ptrace_B.Cmd returns as TRC_C_GPF_ret, the process creates a General Protection fault. The fault type is returned in PTrace_B.Value, and SegV:OffV contains the reference that generated the fault.

If Ptrace_B.Cmd is returned as TRC_C_LIB_ret, a library module has been loaded. The new module table entry (MTE) is returned in Ptrace_B.Value. This can be used with the library support commands to identify the library module. The program module's MTE is returned in PTrace_B.MTE. In this case, the initial TRC_C_Stop command should be re-issued until TRC_C_SUC_ret is returned.

If Ptrace_B.Cmd is returned as TRC_C_FPE_ret, the process has generated a floating point error.

DosPurgeQueue – Purge Queue

Purpose

DosPurgeQueue purges a queue of all elements.

Calling Sequence

```
EXTRN  DosPurgeQueue:FAR

PUSH   WORD    QueueHandle   ;Handle of queue to purge
CALL   DosPurgeQueue
```

Where

QueueHandle
   is the handle of the queue to purge.

Returns

IF   AX = 0 then NO error

ELSE AX = error code

Remarks

Only the queue owner (the process which created the queue via DosCreateQueue) is allowed to issue this call. Any thread within that process can issue DosPurgeQueue calls to any queue owned by that process.

DosPutMessage – Output Message Text to Indicated Handle

Purpose

DosPutMessage outputs the message in a buffer passed by a caller to the specified handle. The function formats the buffer to prevent words from wrapping if displayed to a screen.

Calling Sequence

```
EXTRN  DosPutMessage:FAR

PUSH   WORD    FileHandle     ;Handle of output file/device
PUSH   WORD    MessageLength  ;Length of message buffer
PUSH@  OTHER   MessageBuffer  ;Message buffer
CALL   DosPutMessage
```

Where

FileHandle
is the handle of the output file or device.

MessageLength
is the length of the message to be output.

MessageBuffer
is the buffer that contains the message to be output.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

Screen width is assumed to be 80 characters. If a word is about to span column 80, the word will start on a new line at column 1. DosPutMessage assumes the starting cursor position is column one when handling a word wrap.

If the last character to be positioned on a line is a double-byte character that would be bisected, the rule above insures that the character is not bisected.

DosQCurDir – Query Current Directory

Purpose

DosQCurDir gets the full path name of the current directory for the requesting process for the specified drive.

Calling Sequence

```
EXTRN  DosQCurDir:FAR

PUSH   WORD    DriveNumber    ;Drive number
PUSH@  OTHER   DirPath        ;Directory path buffer (returned)
PUSH@  WORD    DirPathLen     ;Directory path buffer
                              ; length (returned)
CALL   DosQCurDir
```

Where

DriveNumber
is the drive number, for example:

0 = default
1 = A

DirPath
is where the system returns the full directory path name.

DirPathLen
is the length of the DirPath buffer. When DosQCurDir is called, this field must contain the length of the directory path buffer. If an error is returned by DosQCurDir because the buffer is too small, the DirPathLen field is updated with the required length.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

The drive letter is not part of the returned string. The string does not begin with a backslash and is terminated by a byte containing *00H*.

Purpose

DosQCurDisk determines the current default drive for the requesting process.

Calling Sequence

```
EXTRN  DosQCurDisk:FAR

PUSH@  WORD    DriveNumber      ;Default drive number (returned)
PUSH@  DWORD   LogicalDriveMap  ;Drive/map area (returned)
CALL   DosQCurDisk
```

Where

DriveNumber
is where the system returns the number of the default drive, for example:, 1=A, 2=B...

LogicalDriveMap
is a bit map (stored in the low-order portion of the 32-bit, double word area) in which the system returns the mapping of the logical drives. Logical Drives A to Z have a one-to-one mapping with the bit positions 0 to 25 of the map.

If bit value = 0
the logical drive does not exist.

If bit value = 1
the logical drive exists.

Returns

AX = 0

Remarks

None

DosQFHandState – Query File Handle State

Purpose

DosQFHandState queries the state of the specified file.

Calling Sequence

```
EXTRN  DosQFHandState:FAR

PUSH   WORD    FileHandle       ;File handle
PUSH@  WORD    FileHandleState  ;File handle state (returned)
CALL   DosQFHandState
```

Where

FileHandle
is the handle of the file to be queried.

FileHandleState
is the file handle state and consists of the following bit fields:

- Inheritance flag
- Write/through flag
- Fail/errors flag
- Sharing mode field
- Access field
- Reserved bit fields.

The bit field mapping is:

```
Open Mode bits  5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0
                D W F R R R R R I S S S R A A A
```

D DASD Open

The file is opened as follows:

If D = 0
FileHandle represents a file opened in the normal way.

If D = 1
FileHandle represents a mounted disk or diskette volume opened for direct access.

I* *Inheritance Flag

If I = 0
File handle is inherited by a spawned process resulting from a DosExecPgm call.
If I = 1
File handle is private to the current process.

W* *File Write/through

The file is opened as follows:

If W = 0
Writes to the file may be run through the DOS buffer cache.
If W = 1
Writes to the file may go through the DOS buffer cache but sectors are written (actual file I/O completed) before a synchronous write call returns. This state of the file defines it as a synchronous file.

This bit is not inherited by child processes.

F* *Fail/Errors

Media I/O errors are handled as follows:

If F = 0
Reported through the system critical error handler.
If F = 1
Reported directly to the caller via return code.

This bit is not inherited by child processes. Media I/O errors generated through an IOCtl Category 8 function always get reported directly to the caller via return code. The Fail-Errors function applies only to non-IOCtl handle-based type file I/O calls.

R These bits are reserved and should be set to the values returned by DosQFHandState in these positions.

S* *Sharing Mode

The file sharing mode field defines what operations other processes may perform on the file.

If S = 001
Deny Read/Write access

If S = 010
  Deny Write access
If S = 011
  Deny Read access
If S = 100
  Deny Neither Read or Write access (Deny None)

Any other value is invalid.

A Access Mode

The file access is assigned as follows:

If A = 000
  Read/only access
If A = 001
  Write/only access
If A = 010
  Read/Write access

Any other value is invalid.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

When a critical error occurs that the application cannot handle, it may reset critical error handling to be done by the system. This is done by issuing DosQFHandState, turning off the fail/errors bit, issuing DosSetFHandState and subsequently reissuing the I/O. The expected critical error reoccurs and passes to the system critical error handler. The instant in time at which the effect of this function is visible at the application level is unpredictable when asynchronous I/O is pending.

The DASD Open bit parameter is the "Direct I/O flag." It provides an access mechanism to a disk or diskette volume independent of the file system. This mode should only be used by systems programs and not by application programs.

Purpose

DosQFileInfo returns information for a specific file.

Calling Sequence

```
EXTRN  DosQFileInfo:FAR

PUSH   WORD    FileHandle        ;File handle
PUSH   WORD    FileInfoLevel     ;File data required
PUSH@  OTHER   FileInfoBuf       ;File data buffer
PUSH   WORD    FileInfoBufSize   ;File data buffer size
CALL   DosQFileInfo
```

Where

FileHandle
is the file handle.

FileInfoLevel
is the level of file information required. Level 1 file information is returned in the following standard format and, where applicable, is based on the most recent DosClose or DosSetFileInfo:

2 bytes -	File date of creation
2 bytes -	File time of creation
2 bytes -	File date of last access
2 bytes -	File time of last access
2 bytes -	File date of last write
2 bytes -	File time of last write
2 bytes -	File end of data (low word)
2 bytes -	File end of data (high word)
2 bytes -	File allocation (low word)
2 bytes -	File allocation (high word)
2 bytes -	File attribute

FileInfoBuf
is the storage area where the system returns the requested level of file information.

FileInfoBufSize
is the length of FileInfoBuf.

DosQFileInfo – Query File Information

Returns

IF   AX = 0 then NO error

ELSE AX = error code

Remarks

Level 1 is the only defined level of information.

The date and time formats are the same as those for the directory entry. For more information refer to "DosFindFirst – Find First Matching File" on page 2-59.

File date/time of creation and file date/time of last access are not supported in this release and are returned as zeros.

Purpose

DosQFileMode queries the mode (attribute) of the specified file.

Calling Sequence

```
EXTRN  DosQFileMode:FAR

PUSH@  ASCIIZ  FilePathName      ;File path name
PUSH@  WORD    CurrentAttribute  ;Data area (returned)
PUSH   DWORD   0                 ;Reserved (must be 0)
CALL   DosQFileMode
```

Where

FilePathName
is the file path name.

CurrentAttribute
is where the file's current attribute is returned.

Returns

IF   AX = 0 then NO error

ELSE AX = error code

Remarks

The volume label type attribute is not returned by DosQFileMode, DosQFsInfo may be used for this purpose.

DosQFileMode – Query File Mode

File attribute bits are defined as follows:

0001H =	read only file
0002H =	hidden file
0004H =	system file
0010H =	subdirectory
0020H =	file archive
0040H =	reserved
0080H =	reserved
0100H =	reserved
0200H =	reserved
0400H =	reserved
0800H =	reserved
1000H =	reserved
2000H =	reserved
4000H =	reserved
8000H =	reserved

These bits may be set individually or in combination. For example, an attribute of 0021H indicates a read-only file which should be archived.

Purpose

DosQFsInfo queries information from a file system device.

Calling Sequence

```
EXTRN  DosQFsInfo:FAR

PUSH   WORD    DriveNumber    ;Drive number
PUSH   WORD    FSInfoLevel    ;File system data required
PUSH@  OTHER   FSInfoBuf      ;File system info buffer
PUSH   WORD    FSInfoBufSize  ;File system info buffer size
CALL   DosQFsInfo
```

Where

DriveNumber
is the logical drive number (0 = default, 1 = A, etc.).

FSInfoLevel
is the level of file information required.

Level 1 information is returned in the following standard format:

4 bytes -	File System ID
4 bytes -	Number of sectors per allocation unit
4 bytes -	Number of allocation units
4 bytes -	Available allocation units
2 bytes -	Bytes per sector

Level 2 information is returned in the following standard format:

4 bytes -	Reserved
1 byte -	Length of Volume label (null not included)
n bytes -	Volume label ASCIIZ string

FSInfoBuf
is the storage area where the system returns the requested level of file information.

FSInfoBufSize
is the length of FSInfoBuf.

DosQFsInfo – Query File System Information

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

Trailing blanks supplied at volume label definition time are not considered to be part of the label and are therefore not returned as label data.

Purpose

DosQHandType determines whether a handle references a file or a device.

Calling Sequence

```
EXTRN  DosQHandType:FAR

PUSH   WORD    FileHandle     ;File handle
PUSH@  WORD    HandType       ;Handle type (returned)
PUSH@  WORD    FlagWord       ;Device driver attribute (returned)
CALL   DosQHandType
```

Where

FileHandle
is the file handle

HandType

is where the system returns the value indicating the handle type. HandType is composed of two bytes:

HandleClass
describes the handle class. It may take on the following values in the low byte of HandleType:

- 0 = handle is for a disk file
- 1 = handle is for a character device
- 2 = handle is for a pipe.

Values greater than 2 are reserved.

HandleBits
provides further information about the handle in the high byte of HandleType. This byte is broken into eight bits, whose meaning depends upon the value of HandleClass:

DosQHandType – Query Handle Type

```
                HandleBits    HandleClass
              5 4 3 2 1 0 9 8  7 ----- 0
Disk file     N u u u u u u u      0
Char device   N u u u u u u u      1
Pipe          N u u u u u u u      2
```

The network bit = N. If set, it means that the handle refers to a remote file, device, or pipe. Otherwise, the handle refers to a local file, device, or pipe.

The undefined and reserved bit = u. A program should not depend upon the values of these bits as they are subject to change.

FlagWord
is where the system returns the device driver's attribute word if HandleType indicates a local character device.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

DosQHandType allows some programs which may be interactive or file-oriented to determine the source of their input. For example, COMMAND.COM suppresses writing prompts if the input is from a disk file.

Purpose

DosQueryQueue finds the size of a queue.

Calling Sequence

```
EXTRN  DosQueryQueue:FAR

PUSH   WORD    QueueHandle     ;Handle of queue to find size
PUSH@  WORD    NumberElements  ;Size of the queue (returned)
CALL   DosQueryQueue
```

Where

QueueHandle
is the handle of the queue to find size.

NumberElements
is where the number of entries currently in the queue waiting to be processed are returned.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

Any process which has a queue open may issue this request.

If the owning process closes the queue prior to this request being issued, the "Queue does not exist (invalid queue handle)" return code is returned.

DosQVerify – Query Verify Setting

Purpose

DosQVerify returns the value of the verify flag.

Calling Sequence

```
EXTRN  DosQVerify:FAR

PUSH@  WORD    VerifySetting  ;Verify setting (returned)
CALL   DosQVerify
```

Where

VerifySetting
is where the current verify mode for the process is returned.

If value = 00H
verify mode is not active.
If value = 01H
verify mode is active.

Returns

AX = 0

Remarks

None

Purpose

DosRead reads the specified number of bytes from a file or device to a buffer location.

Calling Sequence

```
EXTRN  DosRead:FAR

PUSH   WORD    FileHandle    ;File Handle
PUSH@  OTHER   BufferArea    ;User buffer
PUSH   WORD    BufferLength  ;Buffer length
PUSH@  WORD    BytesRead     ;Bytes read (returned)
CALL   DosRead
```

Where

FileHandle
is the file handle obtained from DosOpen.

BufferArea
is the input buffer.

BufferLength
is the number of bytes to be read.

BytesRead
is where the the number of bytes read is returned.

Returns

IF AX = 0 then NO error

ELSE AX = error code

DosRead — Read from File

Family API Considerations

Some options operate differently in the DOS mode than they do in the OS/2 mode. Therefore, the following restrictions apply to DosRead when coding in the DOS mode:

Use only single-byte DosReads to COMx in PC/DOS. The COM device driver supplied with PC/DOS does not support multiple-byte I/O.

Remarks

The requested number of bytes may not be read. If the value in BytesRead = 0, then the program has tried to read from the end of file.

The file pointer is moved to the desired position by reading, writing, and performing function DosChgFilePtr (Move File Read/Write Pointer).

Purpose

DosReadAsync transfers the specified number of bytes from a file to a buffer, asynchronously with the requesting process execution.

Calling Sequence

```
EXTRN DosReadAsync:FAR

PUSH   WORD    FileHandle    ;File handle
PUSH@  DWORD   RamSemaphore  ;Ram semaphore
PUSH@  WORD    ReturnCode    ;I/O error RC (returned)
PUSH@  OTHER   BufferArea    ;User buffer
PUSH   WORD    BufferLength  ;Buffer length
PUSH@  WORD    BytesRead     ;Bytes read (returned)
CALL  DosReadAsync
```

Where

FileHandle
is the file handle obtained from DosOpen.

RamSemaphore
is used by the system to signal the caller that the read operation is complete.

ReturnCode
is where the return code is returned.

BufferArea
is the input buffer.

BufferLength
is the number of bytes to be read.

BytesRead
is where the number of bytes read is returned.

Returns

AX = 0

Note: When RamSemaphore is cleared and the read operation is complete, ReturnCode can be checked.

DosReadAsync – Asynchronous Read from File

Remarks

The requested number of bytes may not be read. When RamSemaphore is cleared, if the value in BytesRead is 0, the program attempted to read from the end of the file.

The value of the file read/write pointer is updated before the I/O request is queued to the device driver.

RamSemaphore must be set by the application before the DosReadAsync call is made. The application issues the following sequence:

- DosSemSet
- DosReadAsync
- DosSemWait.

The program must not look at the values returned in ReturnCode, BufferArea, or BytesRead until after RamSemaphore is cleared.

Purpose

DosReadQueue reads an element from a queue and removes it.

Calling Sequence

```
EXTRN DosReadQueue:FAR

PUSH   WORD   QueueHandle       ;Handle of queue to read from
PUSH@  DWORD  Request           ;Request identification data (returned)
PUSH@  WORD   DataLength        ;Length of element received (returned)
PUSH@  DWORD  DataAddress       ;Element received
PUSH   WORD   ElementCode       ;Indicate want a particular element
PUSH   WORD   NoWait            ;Indicate to not wait if queue is empty
PUSH@  WORD   ElemPriority      ;Priority of element
PUSH   DWORD  SemaphoreHandle   ;Semaphore handle
CALL   DosReadQueue
```

Where

QueueHandle
is the handle of the queue to read from.

Request
is an area in which the following information is returned:

The first word is the PID of the process which added the element to the queue.

The second word is used for event encoding by the application. The data in this word is the same as that furnished by the Request parameter on the DosWriteQueue request for the corresponding queue element. The value of this data is understood by the client thread and by the server thread. There is no special meaning to this data and the operating system does not alter the data.

DataLength
is where the length of the data being received is returned.

DataAddress
is where the address of the received element is returned.

ElementCode
indicates to override the normal priority, FIFO, or LIFO read ordering. This operand is used to identify a specific element which is to be read. This field should be set to 0 (by the application) to read the first element in the queue, or set to non-zero (to the value returned by a previous DosPeekQueue operation) to indicate read a peeked element.

NoWait
specifies the action to be performed when there are no entries in the queue.

If value = 0
the requesting thread waits.

If value = 1
the requesting thread does not wait.

ElemPriority
is where the priority specified when the element was added to the queue is received. This is a numeric value in the range of 0 to 15 with 15 being the highest priority.

SemaphoreHandle
is the handle of the semaphore cleared when the queue has data placed into it and NoWait=1 is specified. The semaphore may be either a *RAM* or system semaphore.

If this handle is for a RAM semaphore, that semaphore must be in a shared segment between the queue owner's process and any process that issues a DosWriteQueue request to an associated queue.

If multiple threads are processing elements from the queue using a NoWait value = 1, the same semaphore must be provided on all DosPeekQueue or DosReadQueue requests.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

DosReadQueue retrieves and removes an element from a specified queue.

If the queue is empty, the requesting thread is placed in a wait state until an element is added to the queue. If the NoWait option is selected, the thread is not placed in a wait state, but is returned with a code indicating there are no entries on the queue.

If ElementCode is provided, the element indicated is returned. If ElementCode equals 0, the first element in the queue is returned. This allows a thread to read an element from a queue and use DosPeekQueue to compare other elements in the queue to the one read.

Only the queue owner (the process which created the queue via DosCreateQueue) is allowed to issue this call. Any thread within that process may also issue DosReadQueue calls to any queue owned by that process.

The semaphore provided by SemaphoreHandle would typically be used with a DosMuxSemWait request to wait on a queue or other events. This operand is ignored if NoWait = 0 is specified.

DosReallocHuge – Change Huge Memory Size

Purpose

DosReallocHuge changes the size of memory originally allocated by DosAllocHuge.

Calling Sequence

```
EXTRN  DosReallocHuge:FAR

PUSH   WORD     NumSeg     ;Number of 65536-byte segments requested.
PUSH   WORD     Size       ;Number of bytes in last segment
PUSH   WORD     Selector   ;Selector
CALL   DosReallocHuge
```

Where

NumSeg
is the number of 65536 byte segments requested.

Size
is the number of bytes requested in the last non-65536 byte segment. A value of 0 indicates none.

Selector
is the selector returned on a previous DosAllocHuge.

Returns

IF   AX = 0 then NO error

ELSE AX = error code

Family API Considerations

Some options operate differently in the DOS mode than they do in the OS/2 mode. Therefore, the following restriction applies to DosReallocHuge when coding in the DOS mode:

The requested Size value is rounded up to the next paragraph.

Since the MaxNumSeg parameter in DosAllocHuge is ignored in the DOS mode, any subsequent call to DosReallocHuge will also ignore any previous setting of MaxNumSeg.

Remarks

The maximum new size is the value specified for MaxNumSeg on the original DosAllocHuge request.

DosReallocSeg – Change Segment Size

Purpose

DosReallocSeg changes the size of a segment already allocated.

Calling Sequence

```
EXTRN DosReallocSeg:FAR

PUSH   WORD   Size     ;New size requested in bytes
PUSH   WORD   Selector ;Selector
CALL   DosReallocSeg
```

Where

Size
is the new segment size requested in bytes. A value of 0 indicates 65536 bytes.

Selector
is the selector of the segment to be resized.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Family API Considerations

Some options operate differently in the DOS mode than in the OS/2 mode. Therefore, the following restriction applies to DosReallocSeg when coding in the DOS mode:

- the requested Size value is rounded up to the next paragraph.

Remarks

DosReallocSeg is supported for shared and unshared segments. Shared segments can be increased but not decreased in size.

Note that a call to DosReallocSeg referencing a discardable segment (a segment which was allocated via DosAllocSeg with AllocFlags bit 2 (0100B) set) will, in addition to reallocating the memory, perform the same action as a call to DosLockSeg. Refer to "DosLockSeg — Lock Segment in Memory" on page 2-112 for more information about this option.

Note: Data in segments discarded in low-memory situations is not retained. The only way to reference the discarded segments again is to reallocate them.

DosReallocHuge is used to change the size of memory allocated with DosAllocHuge.

DosResumeThread – Restart Thread

Purpose

DosResumeThread restarts a thread previously stopped by way of the DosSuspendThread system call.

Calling Sequence

```
EXTRN  DosResumeThread:FAR

PUSH   WORD    ThreadID     ;Thread ID of thread to resume
CALL   DosResumeThread
```

Where

ThreadID
is the Thread ID of the thread to be resumed.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

None

Purpose

DosRmDir removes a subdirectory from the specified disk.

Calling Sequence

```
EXTRN  DosRmDir:FAR

PUSH@  ASCIIZ  DirName       ;Directory name
PUSH   DWORD   0             ;Reserved (must be 0)
CALL   DosRmDir
```

Where

DirName
Is the directory path name.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

The directory must be empty before it can be removed with the exception of the "." and ".." . You cannot remove subdirectories that contain hidden files. The last directory name in the path is the directory to be removed. The root directory and the current directory cannot be removed.

DosScanEnv – Scan an Environment Segment

Purpose

DosScanEnv scans (searches) an environment segment for an environment variable.

Calling Sequence

```
EXTRN DosScanEnv:FAR

PUSH@  ASCIIZ  EnvVarName     ;Environment variable name
PUSH@  DWORD   ResultPointer  ;Search result pointer (returned)
CALL  DosScanEnv
```

Where

EnvVarName
is the name of the environment variable to be located. Do not include a trailing "=", since this is not part of the name.

ResultPointer
is where the address of the value for the specified environment variable is returned.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

Assume that the processes' environment contains:

```
"DPATH=c:\sysdir;c:\libdir"
       ^
       :
       +...ResultPointer points here after call
       to DosScanEnv below.

        DosScanEnv("DPATH", &ResultPointer);

        As noted above, ResultPointer will point to the first
        character of the value of the environment variable.
```

DosSearchPath – Search Path for File Name

Purpose

DosSearchPath provides a general path search mechanism which allows applications to find files residing along paths. The path string may come from the process environment, or be supplied directly by the caller.

Calling Sequence

```
EXTRN DosSearchPath:FAR

PUSH   WORD    Control          ;Function control vector
PUSH@  ASCIIZ  PathRef          ;Search path reference
PUSH@  ASCIIZ  FileName         ;File name
PUSH@  OTHER   ResultBuffer     ;Search result buffer
PUSH   WORD    ResultBufferLen  ;Search result buffer length
CALL   DosSearchPath
```

Where

Control

is a word bit vector which controls the behavior of DosSearchPath:

- Bit 0 = implied current bit
- Bit 1 = path source bit
- Bits 2-15 = reserved bits, must be 0.

The implied current bit controls whether the current directory is implicitly on the front of the search path. If the implied current bit = 0, DosSearchPath will only search the current directory if it appears in the search path. If the implied current bit = 1, DosSearchPath will search the current working directory before it searches the directories in the search path.

For example, implied current bit = 0 and path = ".\;a;b" is equivalent to implied current bit = 1 and path = "a;b".

The path source bit determines how DosSearchPath interprets the PathRef argument. If the path source bit = 0, then PathRef points to the actual search path. The search path string may be anywhere in the calling processes' address space, therefore, it may be in the environment, but does not have to be.

DosSearchPath – Search Path for File Name

If the path source bit = 1, then PathRef points to the name of an environment variable in the process environment, and that environment variable contains the search path.

PathRef
If the path source bit of control = 0, then PathRef is the search path, which may be anywhere in the caller's address space.

If the path source bit of control = 1, then PathRef is the name of an environment variable which contains the search path.

A search path consists of a sequence of paths separated by ";". It is a single ASCIIZ string. The directories will be searched in the order they appear in the path.

Environment variable names are simply strings which match name strings in the environment. The "=" sign is not part of the name.

FileName
is the ASCIIZ file name to search for. It may contain global characters. If FileName does contain global characters, they will remain in the result path returned in ResultBuffer. This allows applications like CMD.EXE to feed the output directly to DosFindFirst. If there are no wild cards in FileName, the result path returned in ResultBuffer will be a full qualified name, and may be passed directly to DosOpen, or any other system call.

ResultBuffer
is where the result pathname of the file is returned, if found.

ResultBufLen
is the length in bytes of the ResultBuffer.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

PathRef always points to an ASCIIZ string. Let DPATH be an environment variable in the environment segment of the process.

```
"DPATH=c:\sysdir;c:\init"/* in the environment */
```

The following two code fragments are equivalent:

```
DosScanEnv("DPATH", &PathRef);
DosSearchPath(0, /* Path Source Bit = 0 */
    PathRef, "myprog.ini", &ResultBuffer, ResultBufLen);

DosSearchPath(2, /* Path Source Bit = 1 */
    "DPATH", "myprog.ini", &ResultBuffer, ResultBufLen);
```

Both of them use the search path stored as DPATH in the environment segment. In the first case, the application uses DosScanEnv to find the variable, in the second case DosSearchPath calls DosScanEnv for the application.

DosSearchPath does not check for consistency or formatting on the names, it does a DosFindFirst on a series of names it constructs from PathRef and FileName.

To determine the size of the returned pathname, the ResultBuffer must be scanned for the ASCIIZ terminator.

DosSelectDisk – Select Default Drive

Purpose
DosSelectDisk selects the drive specified as the default drive for the calling process.

Calling Sequence

```
EXTRN  DosSelectDisk:FAR

PUSH   WORD    DriveNumber   ;Default drive number
CALL   DosSelectDisk
```

Where

DriveNumber
contains the new default drive number, where 1 = A and 2 = B., and so on.

Returns

IF   AX = 0 then NO error

ELSE AX = error code

Remarks
None

Purpose

DosSelectSession allows a parent session to switch one of its child sessions to the foreground.

Calling Sequence

```
EXTRN DosSelectSession:FAR

PUSH   WORD    SessID        ;Session ID
PUSH   DWORD   Reserved      ;Reserved (must be zero)
CALL DosSelectSession
```

Where

SessID
is the ID of the session to be switched to the foreground. The value specified for SessID must have been returned on a prior call to DosStartSession except that a value of 0 indicates to switch the caller's session, (that is, the parent session), to the foreground.

Reserved
is a DWORD of 0.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

DosSelectSession can only be issued by a parent session to select itself or a child session. DosSelectSession can not be used to select a grandchild session. DosSelectSession may only be used to select child sessions which were originally started by the caller with DosStartSession specifying Related equal 1. That is, sessions started as independent sessions can not be selected through this call.

DosSelectSession – Select Foreground Session

When DosSelectSession is issued, the session specified will not be brought to the foreground unless the parent session or one of its descendant sessions is currently executing in the foreground. Otherwise, a unique error code is returned in AX.

DosSelectSession can only be issued by the process that originally started (using DosStartSession) the SessID specified.

Purpose

DosSemClear unconditionally clears a semaphore. If any threads were blocked on the semaphore, they are restarted.

Calling Sequence

```
EXTRN  DosSemClear:FAR

PUSH   DWORD   SemHandle     ;Semaphore handle
CALL   DosSemClear
```

Where

SemHandle
is the handle for the semaphore. For a system semaphore, this handle is the result of the DosCreateSem or DosOpenSem request which granted this process access to the semaphore. For a RAM semaphore, this handle is the address of the storage allocated for the semaphore.

Returns

IF   AX = 0 then NO error

ELSE AX = error code

Remarks

DosSemClear is typically used to release a semaphore obtained through DosSemRequest. DosSemClear is also used with the semaphore signalling functions DosSemSetWait, DosSemWait, and DosMuxSemWait, to clear a semaphore. A semaphore is checked only when a process receives its time slice and if a semaphore clears and resets during the time it is not processing, then the process does not become blocked.

DosSemClear — Clear (Release) Semaphore

DosSemClear cannot be issued against a system semaphore owned by another process unless the NoExclusive option was selected on the DosCreateSem request that created the semaphore. However, at interrupt time any thread may clear an exclusively owned semaphore.

DosSemRequest – Request Semaphore

Purpose

DosSemRequest obtains a semaphore. If the semaphore is already owned, the requesting thread is placed in a wait state until the semaphore is released or until a time out occurs.

Calling Sequence

```
EXTRN
DosSemRequest:FAR

PUSH   DWORD   SemHandle      ;Semaphore handle
PUSH   DWORD   Timeout        ;Timeout
CALL   DosSemRequest
```

Where

SemHandle
is the handle for the semaphore. For a system semaphore, this handle is the result of the DosCreateSem or DosOpenSem request which granted this thread access to the semaphore. For a RAM semaphore, this handle is the address of the storage allocated for the semaphore.

Timeout
is the time, in milliseconds, until the requesting thread resumes execution if the requested semaphore did not become available. The meaning of the values specified are:

If value = -1
there is no time out, if the semaphore is owned. The requestor waits indefinitely.

If value = 0
there is an immediate return if the semaphore is owned.

If value >0
the value is the number of milliseconds to wait if the semaphore is owned.

DosSemRequest – Request Semaphore

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

DosSemRequest checks the status of the semaphore. If a semaphore is unowned, DosSemRequest sets it owned and returns immediately to the caller. If the semaphore is owned, DosSemRequest can optionally block the thread until it is unowned, then try again. The Timeout parameter places an upper bound on the amount of time to block before returning even though the semaphore is owned.

When a thread owns a semaphore, it is invalid for another thread to issue a semaphore request that changes the state of that semaphore, unless the NoExclusive option is specified. However, at interrupt time, any thread may clear any exclusive, owned, semaphore. For exclusive system semaphores, recursive requests for system semaphores are supported by means of a use count of the number of times the owner has issued a DosSemRequest without a corresponding DosSemClear. When a thread owns a semaphore it is invalid for another thread to issue any semaphore request such as, (DosSemClear), that will change the state of the semaphore unless the NoExclusive option was specified in the original DosSemRequest.

The unblocking of a DosSemRequest does not return unless the indicated semaphore remains clear until the affected thread is redispatched and is able to claim it. This procedure is known as "level triggered" unblocking.

RAM semaphores are generated by a doubleword in RAM. The doubleword initialized to 0 indicates the semaphore is unowned.

Note: An application can issue DosSemSet if it requires a semaphore to be initially set to owned.

System semaphores are generated by a semaphore data structure allocated by DosCreateSem, and controlled by OS/2. System semaphores are initialized to unowned. However, if an application requires the semaphore to be initially set to owned, issue DosSemSet after DosCreateSem.

If a thread terminates while it owns a system semaphore, the ERROR_SEM_OWNER_DIED return code is returned to the thread that gets the next semaphore via DosSemRequest. That thread takes steps to ensure the integrity of the resource. The thread can release the resource by issuing a DosSemClear or it can reset the ERROR_SEM_OWNER_DIED error condition flagged in the semaphore data structure.

When a thread no longer requires the protected resource, it issues DosSemClear and sets the semaphore unowned. Any threads that were blocked waiting for that semaphore are started at this time.

Before owned system semaphores are freed, issue DosExitList. This allows a process to clean up the current resource before it terminates and avoids receiving any error code.

DosSemSet – Set Semaphore Owned

Purpose

DosSemSet unconditionally sets a semaphore.

Calling Sequence

```
 EXTRN  DosSemSet:FAR

PUSH   DWORD   SemHandle     ;Semaphore handle
CALL   DosSemSet
```

Where

SemHandle
is the handle for the semaphore. For a system semaphore, this is the result of the DosCreateSem or DosOpenSem request which granted this thread access to the semaphore. For a RAM semaphore, this handle is the address of the storage allocated for the semaphore.

Returns

IF   AX = 0 then NO error

ELSE AX = error code

Remarks

DosSemSet is not required in a resource control environment using DosSemRequest and DosSemClear. However, it is typically used in a signaling environment implemented via DosSemClear, DosSemWait, and DosMuxSemWait. These function calls can be used in combination with DosSemClear and DosSemSet to awaken a blocked thread whenever a semaphore is cleared rather than when it is no longer owned.

Note: DosSemSet cannot be issued against a system semaphore which is owned by another process unless the NoExclusive option was selected on the original DosCreateSem request.

DosSemSetWait – Set Semaphore and Wait for Next Clear

Purpose

DosSemSetWait blocks the current thread until the next DosSemClear is issued. However, DosSemSetWait does not establish ownership of this semaphore.

Calling Sequence

```
EXTRN DosSemSetWait:FAR

PUSH   DWORD   SemHandle      ;Semaphore handle
PUSH   DWORD   Timeout        ;Timeout
CALL   DosSemSetWait
```

Where

SemHandle
is the handle for the semaphore. For a system semaphore, this handle is the result of the DosCreateSem or DosOpenSem request that granted this thread access to the semaphore. For a RAM semaphore, this handle is the address of the storage allocated for the semaphore.

Timeout
is the time, in milliseconds, until the requesting process is to resume execution if the requested semaphore does not become available. The meaning of the values specified are

If value = -1
there is no timeout, if a DosSemClear is not issued. The requestor waits indefinitely.

If value = 0
there is an immediate return.

If value > 0
value is the number of milliseconds to wait if a DosSemClear is not issued.

DosSemSetWait – Set Semaphore and Wait for Next Clear

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

DosSemSetWait is set before the thread is blocked, if the semaphore is not initially set. The semaphore resets on return.

The unblocking of DosSemSetWait does not return unless the indicated semaphore remains clear until the affected thread is redispatched and determines the semaphore is clear. This is known as a "level triggered" procedure.

DosSemSetWait cannot be issued against a system semaphore owned by another thread unless the NoExclusive option was selected on the DosCreateSem request that created the semaphore. If a system semaphore is created with the exclusive option, DosSemSetWait should not be used to coordinate execution between threads. The owner must clear this semaphore.

DosSemWait – Wait for Semaphore To Clear

Purpose

DosSemWait blocks the current thread until an indicated semaphore clears, but does not establish ownership of the semaphore.

Calling Sequence

```
EXTRN  DosSemWait:FAR

PUSH   DWORD   SemHandle     ;Semaphore handle
PUSH   DWORD   Timeout       ;Timeout
CALL   DosSemWait
```

Where

SemHandle
is the handle for the semaphore. For a system semaphore, this handle is the result of the DosCreateSem or DosOpenSem request that granted this thread access to the semaphore. For a RAM semaphore, this handle is the address of the storage allocated for the semaphore.

Timeout
is the time, in milliseconds, until the requesting process is to resume execution if the requested semaphore does not become available. The meaning of the values specified are:

If value = -1
there is no time out if the semaphore is set. The requestor waits indefinitely.

If value = 0
there is an immediate return if the semaphore is set.

If value > 0
the value is the number of milliseconds to wait if the semaphore is set.

DosSemWait – Wait for Semaphore To Clear

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

The unblocking of DosSemWait does not return unless the indicated semaphore remains clear until the affected thread has been redispatched and determines that the indicated semaphore is clear. This is known as a "level-triggered" procedure.

Purpose

DosSendSignal sends a CTL-C or CTL-Break signal to the last process in the command subtree (leaf-most) that has a corresponding signal handler installed. A command subtree is all of the processes created as a result of a single command.

Calling Sequence

```
EXTRN DosSendSignal:FAR

PUSH   WORD    PID           ;PID of root of subtree
PUSH   WORD    SigNumber     ;Signal Number to send
CALL   DosSendSignal
```

Where

PID

is the process ID of the root process of the subtree. It is not necessary that this process still be alive, but it is necessary that this process be a direct child of the process which issues this call.

SigNumber

is the signal to send. It may be:

- 1 - Ctrl-C (SIGINTR)
- 4 - Ctrl-Break (SIGBREAK)

Returns

IF   AX = 0 then NO error

ELSE AX = error code

Remarks

The signal is sent by descending the process tree to the leaf-most process. Then starting with that process, look for one that has a handler installed for the corresponding signal. If a handler is found, give the signal to that process. Otherwise look at the parent process. Continue until either the signal is sent or the original process is looked at. The latter case is indicated by a unique error code.

DosSetCp – Set Code Page

Purpose

DosSetCp allows a process to set its code page and the session's display code page and keyboard code page.

Calling Sequence

```
EXTRN   DosSetCp:FAR

PUSH   WORD    CodePage      ;Code page identifier
PUSH   WORD    Reserved      ;Reserved, set to 0
Call   DosSetCp
```

Where

CodePage
is a code page identifier word that has one of the following values:

Identifier	Description
437	IBM PC US 437 code page
850	Multilingual code page
860	Portuguese code page
863	Canadian-French code page
865	Nordic code page

Reserved
is a reserved word that must be set to 0.

Returns

IF AX = 0 then no error

ELSE AX = error code.

Remarks

DosSetCp allows a program to set its code page. See CONFIG.SYS and the CODEPAGE command for preparing code pages for the system. The first code page specified in the CODEPAGE command is the default system code page. The session code page of a new session is set to the default system code page. A session's code page can be changed by the user with the CHCP command at the command prompt. The process code page of a new program started from a session command prompt is set to that session's code page.

DosSetCp sets the process code page of the calling process. The code page of a process is used in the following ways. First, the printer code page is set to the process code page through the file system and printer spooler (the system spooler must be installed) when the process makes an open printer request. Calling DosSetCp does not affect the code page of a printer opened prior to the call and does not affect the code page of a printer opened by another process. Second, country dependent information will, by default, be retrieved encoded in the code page of the calling process. And third, a newly created process inherits its process code page from its parent process.

DosSetCp also sets, in the session to which the calling process belongs, the code page for the session's default logical keyboard and automatically flushes the keyboard buffer. It also sets the display code page for the session's logical display. This setting of the code page for the session's default logical keyboard and display overrides any previous setting by DosSetCp, KbdSetCp, and VioSetCp by any process in the same session.

Also see DosSetProcCp.

DosSetDateTime – Set Current Date and Time

Purpose

DosSetDateTime is used to set the date and time that are maintained by the operating system.

Calling Sequence

```
EXTRN  DosSetDateTime:FAR

PUSH@  OTHER   DateTime      ;Date/time structure
CALL   DosSetDateTime
```

Where

DateTime

is a structure that contains the following data items:

BYTE 0	- Hours is the new hour.
BYTE 1	- Minutes is the new minute.
BYTE 2	- Seconds is the new second.
BYTE 3	- Hundredths is the new hundredths of a second.
BYTE 4	- Day is the day to be set.
BYTE 5	- Month is the month to be set.
WORD 6	- Year is the year to be set.
WORD 8	- TimeZone minutes from UTC.

Note: The numbers values listed in the above structure represent decimal offsets.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

The DayofWeek value is based on Sunday = to 0. TimeZone is the difference in minutes between the current time zone and UTC. This is a positive number if earlier than UTC and a negative number if later. For Eastern Standard Time this value would be 300 (5 hours earlier than UTC).

Purpose

DosSetFHandState sets the state of the specified file.

Calling Sequence

```
EXTRN  DosSetFHandState:FAR

PUSH   WORD    FileHandle       ;File handle
PUSH   WORD    FileHandleState  ;File handle state
CALL   DosSetFHandState
```

Where

FileHandle

is the handle of the file to be set.

FileHandleState

is the file handle state and consists of the following bit fields:

- Inheritance flag
- Write-through flag
- Fail-errors flag
- Zero bit field.

```
File Handle State Bits

5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0
0 W F R R R R R I 0 0 0 R 0 0 0
```

I Inheritance Flag

If I = 0
File handle is inherited by a spawned process resulting from a DosExecPgm call.

If I = 1
File handle is private to the current process.

W File Write-through

The file is opened as follows:

If W = 0
Writes to the file may be run through the DOS buffer cache.

If W = 1
: Writes to the file may go through the DOS buffer cache but the data is written (actual file I/O completed) before a synchronous write call returns. This state of the file defines it as a synchronous file.

This bit is not inherited by child processes.

F Fail-Errors

Media I/O errors are handled as follows:

If F = 0
: Reported through the system critical error handler.

If F = 1
: Reported directly to the caller by way of the return code.

This bit is not inherited by child processes. Media I/O errors generated through an IOCtl category eight function always get reported directly to the caller via return code. The Fail-Errors function applies only to non-IOCtl handle-based type file I/O calls.

0 Zero bits

These bits must be set to zero. Any other values for FileHandleState are invalid

R Reserved bits

These bits are reserved and should be set to the values returned by DosQFHandState in these positions.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Family API Considerations

Some options operate differently in the DOS mode than they do in OS/2 mode. Therefore, the following restrictions apply to DosSetFHandState when coding in the DOS mode:

- for FileHandle, the validity of the handle is not checked.
- Inheritance Flag must be set equal to zero.
- Write-through Flag must be set equal to zero.
- Fail-Errors Flag must be set equal to zero.

Remarks

OS/2 does not guarantee the order in which sectors are written, for multiple sector writes. If an application requires several sectors written in a specific order, the operator should issue them as separate synchronous write operations. Setting the synchronous (nonbuffered) I/O flag does not affect any previous writes. That data may remain in the buffers.

When a critical error occurs and the application cannot solve it, critical error handling is reset to be done by the system. This is done by issuing DosSetFHandState and subsequently re-issuing the I/O. The expected critical error will re-occur and be passed to the system critical error handler. The instant in time at which the effect of this function is visible at the application level is unpredictable when asynchronous I/O is pending.

The file handle state bits set by this function can be queried by DosQFHandState.

DosSetFileInfo – Set File Information

Purpose

DosSetFileInfo specifies information for a file.

Calling Sequence

```
EXTRN  DosSetFileInfo:FAR

PUSH   WORD    FileHandle       ;File handle
PUSH   WORD    FileInfoLevel    ;File info data required
PUSH@  OTHER   FileInfoBuf      ;File info buffer
PUSH   WORD    FileInfoBufSize  ;File info buffer size
CALL   DosSetFileInfo
```

Where

FileHandle
is the file handle.

FileInfoLevel
is the level of file information being set. Level 1 information is specified in the following standard format:

2 bytes -	File date of creation
2 bytes -	File time of creation
2 bytes -	File date of last access
2 bytes -	File time of last access
2 bytes -	File date of last write
2 bytes -	File time of last write

FileInfoBuf
is the storage area where the system gets the file information.

FileInfoBufSize
is the length of FileInfoBuf.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

Level 1 is currently the only defined level of information. The date and time formats are the same as those for the directory entry.

The DosSetFileInfo level 1 structure is a prefix of the DosQFileInfo level 1 structure.

DosSetFileInfo will work only for files opened in a mode that allows write-access.

A zero value in the date and time components of a field does not change the field. For example, if both "last write date" and "last write time" are specified as zero in the level one information structure, then both attributes of the file are left unchanged. If either "last write date" or "last write time" are specified as non-zero, then both attributes of the file will be set to the new values.

DosSetFileMode – Set File Mode

Purpose

DosSetFileMode changes the mode (attribute) of the specified file.

Calling Sequence

```
EXTRN  DosSetFileMode:FAR

PUSH@  ASCIIZ  FileName      ;File path name
PUSH   WORD    NewAttribute  ;New attribute of file
PUSH   DWORD   0             ;Reserved (must be zero)
CALL   DosSetFileMode
```

Where

FileName
is the file path name.

NewAttribute
is the file's new attribute.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

Attributes for Volume Label (0008H) and Subdirectory (0010H) cannot be changed using DosSetFileMode. If the above referenced attributes are used to change a file's mode, an error code is returned.

File attributes are defined as follows:

0001H = read only file
0002H = hidden file
0004H = system file (excluded from
normal directory searches)
0008H = volume label
0010H = subdirectory
0020H = file archive
0040H = reserved
0080H = reserved
0100H = reserved
0200H = reserved
0400H = reserved
0800H = reserved
1000H = reserved
2000H = reserved
4000H = reserved
8000H = reserved

DosSetFsInfo – Set File System Information

Purpose

DosSetFsInfo sets information for a file system device.

Calling Sequence

```
EXTRN  DosSetFsInfo:FAR

PUSH   WORD    DriveNumber   ;Drive number
PUSH   WORD    FSInfoLevel   ;File system data type
PUSH@  OTHER   FSInfoBuf     ;File system info buffer
PUSH   WORD    FSInfoBufSize ;File system info buffer size
CALL   DosSetFsInfo
```

Where

DriveNumber
is the logical drive number, for example, 0 = default and 1 = A.

FSInfoLevel
is the level of file information to be set.

Level 2 information is specified in the following standard format:

1 byte = length of Volume Label (null not included)
N bytes = Volume Label ASCIIZ string

FSInfoBuf
is the storage area where the system gets the new file system information.

FSInfoBufSize
is the length of FSInfoBuf.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

Trailing blanks supplied at volume label definition time are not returned by DosQFsInfo.

File system information can only be set if the volume is opened in a mode that allows write-access.

DosSetMaxFH – Set Maximum File Handles

Purpose

DosSetMaxFH defines the maximum number of file handles for the current process.

Calling Sequence

```
EXTRN  DosSetMaxFH:FAR

PUSH   WORD    NumberHandles  ;Number of file handles
CALL   DosSetMaxFH
```

Where

NumberHandles
is the total number of file handles to be provided.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

All currently open file handles are preserved.

Purpose

DosSetProcCp allows a process to set its code page.

```
EXTRN  DosSetProcCp:FAR

PUSH   WORD    CodePage        ;Code page identifier
PUSH   WORD    Reserved        ;Reserved
Call   DosSetProcCp
```

Where

CodePage
is a code page identifier word that has one of the following values:

Identifier	**Description**
437	IBM PC US 437 code page
850	Multilingual code page
860	Portuguese code page
863	Canadian-French code page
865	Nordic code page

Reserved
is a reserved word that must be set to zero.

Returns

IF AX = 0 then no error

ELSE AX = error code.

Remarks

DosSetProcCp sets the process code page of the calling process. The code page of a process is used in the following ways. First, the printer code page is set to the process code page through the file system and printer spooler (the system spooler must be installed) when the process makes an open printer request. Calling DosSetProcCp does not affect the code page of a printer opened prior to the call and does not affect the code page of a printer opened by another process. Second, country dependent information will, by default, be retrieved encoded in the code page of the calling process.

DosSetProcCp — Set Process Code Page

And third, a newly created process inherits its process code page from its parent process. DosSetProcCp does not affect the display or keyboard code page.

Also see DosSetCp.

Purpose

DosSetPrty allows the caller to change the base priority of a child process or thread in the current process.

Calling Sequence

```
EXTRN  DosSetPrty:FAR

PUSH   WORD    Scope          ;Indicate scope of change
PUSH   WORD    PriorityClass  ;Priority class to set
PUSH   WORD    PriorityDelta  ;Priority delta to apply
PUSH   WORD    ID             ;Process or thread ID
CALL   DosSetPrty
```

Where

Scope
is used to define the scope of the request.

If value = 0
the priority of the indicated process and all its threads will be changed. Any process may be specified.

If value = 1
the priority of the indicated process and all its threads will be changed with the priorities of all descendant processes, (except detached processes) and their threads will be changed. The indicated process must be the current process, or must have been created by the current process as a non-detached process. When the indicated process is terminated, its descendants can still be accessed.

If value = 2
the priority of a single thread within the current process will be changed.

PriorityClass
is used to set the priority class of a process. The values and their meanings are:

0 = no change, leave as is
1 = idle-time
2 = regular
3 = time-critical

PriorityDelta
is the delta priority to apply to the process's current base priority level. This value must range from -31 to +31.

ID is either a process ID (scope = 0 or 1) or a thread ID (scope = 2). If this operand is equal to 0, the current process or thread is assumed.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

The OS/2 scheduler has a concept of priority classes and priority levels. Through this system call, threads may move between classes in response to changes in their execution environments. Within each class, a thread's priority level may vary either through system action or through this system call. System initiated priority variation is performed as a combination of a specific thread's actions and the overall system activity.

Priority Classes A Time-Critical thread is one of the highest priority. Any runable Time-Critical threads, will execute before any Regular or Idle-Time threads. Time-Critical threads have a static priority which is not varied by OS/2. They are scheduled among themselves in priority order with round-robin scheduling of threads of equal priority.

The majority of threads fall into the Regular thread class. The priority level of a Regular thread is varied by OS/2 around a base value according to the activity of the thread and the system at any time. The base value is set by the thread itself.

An Idle-Time thread is low priority and executes only when there are no Regular or Time-Critical threads to execute. Idle-Time threads have a static priority that is not varied by OS/2. They are scheduled among themselves in priority order with round-robin scheduling of threads of equal priority.

Priority Levels: For each of the above priority classes, there are 32 distinct priority levels, 0 to +31. Whenever this request is issued specifying a priority class, the base defaults to 0 for the new class if not otherwise specified.

A process priority consists of a computed priority value that is based upon the process's display status (foreground or background), its recent I/O and processor time-usage history, and other factors. A user-settable value is added to the computed priority to produce the actual priority used by the scheduler. Thus, specifying a larger priority allows a process to obtain better Processor scheduling than it normally would. A smaller priority gives the process less Processor resource than it would normally receive.

The argument to this call specifies a signed delta value. That value is added to the current priority, the result is restricted to the legal range based on the process current Priority Class.

When used with PriorityClass to change to a different class, the delta value applies to the base priority. If PriorityClass is specified, then a new base level of 0 will be assigned the target thread and any PriorityDelta specified will be relative to 0.

The process ID argument specifies which process is to be affected by the call. The Scope determines the extent of the priority change. A process may change the its own priority, of any process which is a descendent, or of one of its threads. When changing the priority of another process or its descendents, only the default priority is changed. Any thread which has specifically changed its priority is unaffected.

When a thread is created, it is initially dispatched in the same class and priority as its parent.

DosSetSession – Set Session Status

Purpose

DosSetSession sets the status of a child session.

Calling Sequence

```
EXTRN DosSetSession:FAR

PUSH   WORD    SessID         ;Session ID
PUSH@  OTHER   StatusData     ;Session status data
CALL DosSetSession
```

Where

SessID

is the ID of the target session. The value specified for SessID must have been returned on a prior call to DosStartSession.

StatusData

is a structure that contains the session status data.

Size	Description
WORD	Length
WORD	SelectInd
WORD	BondInd

Length

is the length of the data structure in bytes including Length itself. This value should be 6.

SelectInd

specifies whether the target session should be flagged selectable or non-selectable.

If value = 0
leave current setting unchanged.
If value = 1
selectable
If value = 2
non-selectable

BondInd

specifies which session to bring to the foreground the next time the parent session is selected.

If value = 0
leave current setting unchanged

If value = 1
establishes a bond between the parent session and the child session. The child session is brought to the foreground the next time the parent session is selected. If the child session is selected, the child session is brought to the foreground.

If value = 2
specifies to bring the parent session to the foreground the next time the parent session is selected and to bring the child session to the foreground if the child is selected. Any bond previously established with a child session is broken.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

DosSetSession sets/resets one or both of the following parameters related to a child session. The parameters can be set individually. Either parameter can be changed without affecting the current setting of the other.

- Selectable/non-selectable: This parameter allows a parent session to set one of its child sessions selectable or non-selectable.
- Bond/no bond: This parameter allows a parent session to bond one of its child sessions to itself. This means if the operator selects the parent session from the Program Selector the child session is brought to the foreground.

These parameters affect selections made by the operator from the Program Selector Switch list, however, they do not affect selections made by the parent session. When a parent session selects its own session, the parent is brought to the foreground even if a bond is in effect. When a parent session selects a child session, the child is brought to the foreground even if the parent had set the child non-selectable.

DosSetSession may only be issued by a parent session for a child session. Neither the parent session nor any grandchild, may be the target of this call. DosSetSession may only be used to change the status of child sessions which were originally started by the caller with DosStartSession specifying Related equal 1.

A bond established between a parent session and a child session can be broken by reissuing DosSetSession and specifying either

BondInd = 2 to break the bond, or

BondInd = 1 to establish a bond with a different child session. In this case the bond with the previous child is broken.

Assume a bond is established between session A and its immediate child session B. Assume another bond is established between session B and its immediate child session C. Now if the operator selects session A session C is brought to the foreground. However, if session A selects its own session, session A is brought to the foreground. If session A selects session B, session C is brought to the foreground. Note that, in the latter case, the bond between B and C is honored.

Assume a bond is established between session A and its immediate child session B, and assume B is non-selectable. The operator cannot select session B directly. However, if the operator selects session A, session B is brought to the foreground.

A parent session can run in either the foreground or background when DosSetSession is issued. DosSetSession can only be issued by the process that originally started (using DosStartSession) the SessID specified.

DosSetSigHandler – Set Signal Handler

Purpose

DosSetSigHandler notifies OS/2 of a handler for a signal. It may also be used to ignore a signal or install a default action for a signal.

Calling Sequence

```
EXTRN  DosSetSigHandler:FAR

PUSH@  OTHER   Routine       ;Signal handler
PUSH@  DWORD   PrevAddress   ;Previous handler (returned)
PUSH@  WORD    PrevAction    ;Previous action (returned)
PUSH   WORD    Action        ;Indicate request type
PUSH   WORD    SigNumber     ;Signal number of interest
CALL   DosSetSigHandler
```

Where

Routine

entry point of routine which is to receive control when a signal equal to SigNumber is issued.

PrevAddress

is where the address of the previous signal handler is returned. This operand may be coded as null (= 0) in which case it will be ignored.

PrevAction

is where the Action of the previous signal handler is returned. Only values 0 to 3 are returned. This operand may be coded as null (= 0) in which case it will be ignored.

Action

is an indicator of the type of request:

If value = 0
: the system default action is installed for the signal.

If value = 1
: the signal is to be ignored.

If value = 2
: the routine receives control when the SigNumber occurs.

If value = 3
: it is an error for any program to signal this SigNumber to this process.

If value = 4
: the current signal is reset without affecting the disposition of the signal.

SigNumber
is the signal number to be intercepted by this signal handler. The signal numbers defined are:

Number	Term	Definition
1	(SIGINTR)	Ctrl-C
3	(SIGTERM)	program terminated
4	(SIGBREAK)	Ctrl-Break
5		Process flag A
6		Process flag B
7		Process flag C

The following chart indicates what signal to specify to cause the signal handler to get control for the CTRL-C and CTRL-Break key sequences in each of the keyboard modes (ASCII and Binary):

	ASCII Mode	BINARY Mode
CTRL-C	SIGINTR	
CTRL-Break	SIGINTR	SIGBREAK

Returns

IF AX = 0 then NO error

ELSE AX = error code

Family API Considerations

Some options operate differently in the DOS mode than they do in OS/2 mode. Therefore, the following restriction applies to DosSetSigHandler when coding in real mode:

The only signal recognized in DOS is SIGINTR (Ctrl-C).

SIGINTR is fully supported, and SIGBREAK is related to SIGINTR. Therefore, if SIGINTR is specified, both SIGINTR and SIGBREAK are transferred to the SIGINTR handler. SIGBREAK is permitted as a coded value, but the request to set SIGBREAK is ignored. To be compatible in all environments, SIGBREAK and SIGINTR should be considered together in all cases.

Remarks

When the signal indicated by SigNumber occurs, the signal handling routine receives control with:

(SS:SP)	= far return address
(SS:SP+4)	= SigNumber being processed
(SS:SP+6)	= SigArg furnished on the DosFlagProcess request, if appropriate.

Other than SS, SP, CS, IP and flags, all other registers contain the same values they contained at the time the signal was received. The handler may exit by executing an intersegment return instruction, or by manually setting the stack frame to some known state and jumping to some known location. If the former option is selected, execution will resume where it was interrupted, and all registers will be restored to their values at the time of the interruption.

The signal handler is given control under the first thread of a process, not a thread created by the DosCreateThread system request.

To return from the signal, the handler must remove the signal number and signal argument passed as parameters. For handlers written in most high-level languages, this is done automatically. A handler written in assembly language must execute a far RET 4 instruction or its equivalent, to return to the caller.

The signal handler may also reset the stack pointer to some previous valid stack frame and jump to some other part of the program.

The values returned in PrevAddress and PrevAction are to be used for restoring the previous signal handler when the current process no longer wishes to intercept this signal. For Action = 4, no values will be returned for PrevAddress or PrevAction.

When a signal is issued from the base keyboard device driver in response to a Ctrl-C or Ctrl-Break key press, the default action will terminate the process if the application did not install a signal handler for any signal numbers 1-4.

It will be invalid to issue this system call when thread 1 has terminated. If thread 1 terminates with other threads still active, all signals will be reset to the default action.

For signals of type SIGINTR or SIGBREAK, a call to DosSetSigHandler also determines which process within the current session will be signalled as a result of a device driver call to Device Helper Services for the SendEvent function and CTRL-C (or CTRL-BREAK) event type (see the OS/2 Technical Reference, Volume 1, chapter nine for Device Helper Services discussion). This process is known as the "signal focus" for SIGINTR (or SIGBREAK) within its session. The signal focus for SIGINTR need not be the same process as the signal focus for SIGBREAK. The signal focus for a session is determined as follows:

Initially, a session has no signal focus for SIGINTR (or SIGBREAK). A process becomes the signal focus for SIGINTR (or SIGBREAK) within its session if it calls DosSetSigHandler with ActionCode equal to 1, 2, or 3. A process remains the signal focus until;

- the process terminates.
- the process calls DosSetSigHandler with ActionCode equal to 0.
- another process calls DosSetSigHandler with ActionCode equal to 1, 2, or 3.

In the first two cases, the parent or most closely related ancestor process that has a handler installed for the appropriate signal becomes the focus. If no eligible process exists, the session ceases to have a signal focus for that signal.

If a device driver makes a SendEvent call for CTRL-C or CTRL-BREAK and the current session has no focus for the corresponding signal, all processes in the session are signaled with SIGTERM to terminate.

DosSetVec – Establish Handler for Exception Vector

Purpose

DosSetVec allows a process to register an address to be used when a machine exception occurs.

Calling Sequence

```
EXTRN DosSetVec:FAR

PUSH   WORD    VecNum          ;Function request code
PUSH@  OTHER   Routine         ;Handler routine
PUSH@  DWORD   PrevAddress     ;Previous handler address (returned)
CALL DosSetVec
```

Where

VecNum
is the number of the vector to be serviced by this routine. Legal numbers are:

00 = divide overflow
04 = overflow
05 = bound
06 = invalid opcode
07 = processor extension not available
16 = processor extension error

Routine
is a routine to be entered when the exception occurs. If this parameter is 0, any previous address is de-registered.

PrevAddress
is where the address of the previous handler routine is returned. This is provided so a handler may be set then later restored to the previous handler.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Family API Considerations

Some options operate differently in the DOS mode than they do in OS/2 mode. Therefore, the following restriction applies to DosSetVec when coding in the DOS mode:

- VecNum = 7 not supported.

Remarks

DosSetVec allows a process to register an address to be used when a machine exception occurs. The process is analogous to setting an address in the interrupt vector table when running in 8086 mode.

Should an exception occur, and the process has registered a handler, that handler is entered as if its address had been stored in the CPUs interrupt vector, except that interrupt is still enabled. If no address has been registered for that vector, the process is terminated.

When a process registers an exception handler for VecNum 7 (processor extension not available) the machine status word (MSW) for that process will be set to indicate a numeric processor extension (NPX) 287 is not present in the machine. The Emulate bit will be set and the Monitor Processor bit will be reset. This is done without regard for the true state of the hardware.

When a process de-registers a handler for VecNum 7, the MSW will be set to reflect the true state of the hardware.

When an NPX287 exception is being processed, the NPX287 status word will be passed to the exception handler by being pushed on the stack prior to the exception handler being invoked. When the exception handler has completed execution, this word must be popped from the stack before an IRET is issued to return to the exception handler interface routine.

DosSetVerify – Set/Reset Verify Switch

Purpose

DosSetVerify sets the verify switch.

Calling Sequence

```
EXTRN DosSetVerify:FAR

PUSH   WORD    VerifySetting  ;New value of verify switch
CALL   DosSetVerify
```

Where

VerifySetting
is the new state of Verify Mode for the requesting process.

If value = 0
verify mode is deactivated.
If value = 1
verify mode is activated.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

When verify is on, OS/2 performs a verify operation each time it does a file write to assure proper data recording on the disk. Although disk recording errors are rare, this function has been provided for applications which may wish to verify the proper recording of critical data.

Purpose

DosSleep suspends the current thread for a specified time, or if the requested interval is 0, gives up the remainder of the current time slice.

Calling Sequence

```
EXTRN  DosSleep:FAR

PUSH   DWORD  TimeInterval  ;Interval size
CALL   DosSleep
```

Where

TimeInterval
is the interval in milliseconds until the thread is awakened.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Family API Considerations

Some options operate differently in the DOS mode than they do in OS/2 mode. Therefore, the following restrictions apply to DosSleep when coding in the DOS mode :

- DosSleep accuracy can be in error by 0.5%.
- DosSleep can degrade system performance of non-foreground program operations when DOS mode is in foreground.

Remarks

DosSleep suspends the current thread for the specified time period. The actual time it is asleep may be off by a clock tick or two, depending on the execution status of the other threads running in the system.

If the time is 0, then the thread foregoes the remainder of the current time slice and allows any other ready threads of equal priority to run with the current thread for its next slice. Since the amount of sleep time specified is 0, an immediate return with 0 delay is made if no other ready thread is found.

If the time is non-0, the time will be rounded up to the resolution of the scheduler clock.

If DosSleep is used to regularly poll an external source to determine the occurrence of some event, a time equal the longest response interval should be used.

For short time intervals the rounding-up process combined with the thread priority interactions may cause a sleeping interval to be longer than the requested time. Also, when a process completes sleeping, it is scheduled for execution. But that execution could be delayed by hardware interrupts or by another thread running at a higher priority. A program should not use the DosSleep call as a substitute for a real time clock because rounding of the sleep interval will cause cumulative errors.

Note: For a time of 0, DosSleep will not yield to a thread of lower priority.

Purpose

DosStartSession provides an application program interface to start another session and specify the name of the program to start in the session.

Calling Sequence

```
EXTRN DosStartSession:FAR

PUSH@  OTHER   StartData  ;Start session data
PUSH@  WORD    SessID     ;Session ID (returned)
PUSH@  WORD    PID        ;Process ID (returned)
CALL   DosStartSession
```

Where

StartData

is a structure containing the data describing the session to be started.

Size	Description
WORD	Length
WORD	Related
WORD	FgBg
WORD	TraceOpt
DWORD	PgmTitle
DWORD	PgmName
DWORD	PgmInputs
DWORD	TermQ

Length

is the length of the data structure in bytes including Length itself. For OS/2, Length is 24 bytes.

Related

specifies whether the session created is related to the calling session.

Value = 0
new session is an independent session (not related)

Value = 1
new session is a child session (related)

An independent session is not a child session and cannot be controlled by the calling program. It cannot be specified as the target of DosSelectSession, DosSetSession, or DosStopSession. The TermQ parameter is ignored for independent sessions, and the SessID and PID are not returned.

The calling program (parent session) may specify a child session as the target of DosSelectSession, DosSetSession, and DosStopSession, for related sessions. The TermQ, SessID, and PID parameters are applicable when Related = 1 is specified. Note also that for related sessions, although a parent session/child session relationship is established, a parent process/child process relationship is not established.

FgBg
specifies whether the new session should be started in the foreground or background.

If value = 0
start session in foreground.
If value = 1
start session in background.

TraceOpt
specifies whether the program started in the new session should be executed under conditions for tracing.

- If value = 0, no trace.
- If value = 1, trace.

PgmTitle
is the far address of an ASCIIZ string containing the program title. The string can be up to 31 bytes long including the terminating byte of 0. If the far address specified is 0, or if the ASCIIZ string is null, the initial title is PgmName minus any leading drive and path information.

PgmName
is the far address of an ASCIIZ string containing the fully-qualified drive, path, and filename of the program to be loaded.

PgmInputs
is the far address of an ASCIIZ string containing the input arguments to be passed to the program.

TermQ
is an optional parameter. It is either 0 or the far address of an ASCIIZ string containing the fully-qualified path and filename of an OS/2 queue. The OS/2 session manager writes a data element into the queue specified when the child session created as a result of this DosStartSession terminates. A parent session issues DosReadQueue to receive notification when a child session terminates. The request word returned by DosReadQueue is 0. The data element structure has the following format:

Size	**Description**
WORD	Session ID of child
WORD	Result code

DosReadQueue is issued by the process that originally issued the DosStartSession request. This process is the only process that has addressability to the notification data element. The NoWait parameter on DosReadQueue must be set equal to zero. After reading and processing the data element, the caller frees the segment containing the data element using DosFreeSeg.

SessID
is the session ID associated with the created child session. SessID is returned only when the specified value for Related equals 1. The returned SessID is specified on subsequent calls to DosSelectSession, DosSetSession, and DosStopSession.

PID
is the process ID associated with the created child process. PID is returned only when the value Related equal 1 is specified. The PID returned cannot be used on any OS/2 calls, (for example, DosSetPrty, which require a parent process/child process relationship).

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

New sessions can be started in the foreground only when the caller's session, or one of the caller's descendant sessions, is currently executing in the foreground. The new session appears in the Program Selector Switch list.

Foreground/Background Considerations: if FgBg = 0 is specified, and if neither the calling program nor any of its descendant sessions is executing in the foreground, the new session is started in the background. A unique error code is also returned in AX in this case.

Parent/Child Relationships: when Related = 1 is specified, DosStartSession establishes a parent session/child session relationship. A parent process/child process relationship is not established. The parent process is the shell process just as if the operator had started the program from the shell menu. A child program started through DosStartSession is therefore not a descendant of the calling program and does not inherit the calling process environment, open file handles, and other items passed to child processes through DosExecPgm. Furthermore, the PID returned by DosStartSession may not be used on any OS/2 calls, (for example, DosSetPrty) which require a parent process/child process relationship. Within any one session, DosStartSession, specifying Related = 1, may be issued by one and only one process.

PgmName/PgmInputs Considerations: the program identified by PgmName is executed directly with no intermediate secondary command (CMD.EXE) process. Alternatively, the program can be executed indirectly through a secondary command (CMD.EXE) process by specifying CMD.EXE for PgmName and by specifying either /C or /K followed by the drive, path, and filename of the application to be loaded for PgmInputs. If the /C parameter is inserted at the beginning of the PgmInputs string, when the application program terminates, the session will terminate. If the /K parameter is inserted at the beginning of the PgmInputs string, when the application terminates, the operator will see the OS/2 command line prompt displayed. The operator can then either enter the name of another program or command to execute or enter the OS/2 EXIT command to terminate the session.

When the PgmName far address is 0 or the ASCIIZ string is null, the program specified as a parameter to the shell on the ProtShell statement in the CONFIG.SYS file will be executed and passed the specified PgmInputs. This is the OS/2 mode command processor (CMD.EXE) by default.

The PgmName and PgmInputs ASCIIZ name strings combined length may not exceed 384 characters.

Parent/Child Termination Considerations: a parent session has only one termination queue. The parent creates the termination queue before it issues its first DosStartSession call that specifies the name of the queue. An application can use the termination queue for another purpose by passing a unique request parameter through the DosWriteQueue/DosReadQueue interface. Request parameter values 0 through 99 are reserved for OS/2. Request parameter values greater than or equal to 100 are available for application use.

If a parent session specifies the TermQ parameter on more than one DosStartSession call, the parameter is ignored on subsequent calls. Once a parent establishes a termination queue, the queue is posted when any child session terminates. The queue is posted regardless of who terminates the child session (for example, child, parent, or operator) and whether the termination is normal or abnormal.

When a child session terminates, the result code returned in the TermQ data element will be the result code of the program specified by PgmName assuming either:

1. the program is executed directly with no intermediate secondary command (CMD.EXE) process, or
2. the program is executed indirectly through a secondary command (CMD.EXE) process and the /C parameter is specified.

If the program is executed indirectly through a secondary command (CMD.EXE) process and the /K parameter is specified, the result code of the command process is returned.

When a child session is running in the foreground at the time it terminates, the parent session becomes the foreground session. When a parent session terminates, any child sessions it created with

DosStartSession, specifying Related = 1, are terminated. When an independent session, created specifying Related = 0, is running in the foreground at the time it terminates, the shell chooses the next foreground session.

Purpose

DosStopSession terminates a session previously started with DosStartSession.

Calling Sequence

```
EXTRN DosStopSession:FAR

PUSH   WORD    TargetOption  ;Target option
PUSH   WORD    SessID        ;Session ID
PUSH   DWORD   Reserved      ;Reserved (must be zero)
CALL   DosStopSession
```

Where

TargetOption

specifies whether the session specified by SessID or all sessions should be terminated.

If value = 0
 terminate session specified.
If value = 1
 terminate all child sessions and descendant sessions.

SessID

is the ID of the session to be terminated. The value specified for SessID must have been returned on a prior call to DosStartSession. SessID is ignored if TargetOption = 1.

Reserved

is a DWORD of 0's.

Returns

IF AX = 0 then NO error

ELSE AX = error code

DosStopSession — Stop Session

Remarks

DosStopSession may only be issued by a parent session for a child session. Neither the parent session itself nor any grandchild, and so forth, may be the target of this call. However, if the child session specified on DosStopSession does have descendants, these sessions will also be terminated.

DosStopSession may only be used to terminate child sessions originally started by the caller with DosStartSession specifying Related = 1. That is, sessions started as independent sessions may not be stopped.

If a child session is running in the foreground at the time it is stopped, the parent session becomes the foreground session. DosStopSession breaks any bond that existed between the parent session and the child session specified.

A parent session may be running in either the foreground or background when DosStopSession is issued.

DosStopSession can only be issued by the process that originally started (using DosStartSession) the SessID specified.

The process running in the session specified on the DosStopSession call may refuse to terminate. DosStopSession will return a normal return code in AX in this case. The only way to ensure that the target session has terminated is to wait upon notification through the termination queue specified on DosStartSession.

Purpose

DosSubAlloc allocates memory from a segment previously allocated by DosAllocSeg or DosAllocShrSeg and initialized by DosSubSet

Calling Sequence

```
EXTRN  DosSubAlloc:FAR

PUSH   WORD    SegSelector    ;Segment selector
PUSH@  WORD    BlockOffset    ;Block Offset (returned)
PUSH   WORD    Size           ;Size of requested block
CALL   DosSubAlloc
```

Where

SegSelector
is the selector of the data segment from which the memory should be allocated.

BlockOffset
is where the offset to the block allocated is returned.

Size
is the size of the memory block requested in bytes.

Returns

IF   AX = 0 then NO error

ELSE AX = error code

Remarks

DosSubAlloc allocates a block of memory from a segment previously allocated by DosAllocSeg or DosAllocShrSeg and initialized by DosSubSet.

Allocation size must be a multiple of four bytes. It will be rounded if not. The maximum value for the size parameter is the size of the segment allocated by DosAllocSeg or DosAllocShrSeg minus 8. Note that no paragraph alignment is required; all requests are serviced on a byte alignment basis.

The requestor must first call DosSubSet, after allocating the segment, and before attempting to call DosSubAlloc to allocate memory within the segment. Refer to "DosSubSet – Initialize or Set Allocated Memory" on page 2-264 for more information.

DosSubFree – Free Memory Suballocated Within Segment

Purpose

DosSubFree frees memory previously allocated by DosSubAlloc.

Calling Sequence

```
EXTRN  DosSubFree:FAR

PUSH   WORD    SegSelector    ;Segment selector
PUSH   WORD    BlockOffset    ;Offset of memory block to free
PUSH   WORD    Size           ;Size of block in bytes
CALL   DosSubFree
```

Where

SegSelector
is the selector of the data segment from which the memory should be freed.

BlockOffset
is the offset of the memory block to be freed. The value specified must equal the BlockOffset returned on a previous DosSubAlloc call.

Size
is the size of the block to be freed in bytes.

Returns

IF   AX = 0 then NO error

ELSE AX = error code

Remarks

DosSubFree is used to free a block of memory that was allocated with DosSubAlloc. If the block specified overlaps unallocated memory, an error is generated. Like DosSubAlloc, the size parameter must be a multiple of four bytes, it will be rounded it not.

DosSubSet – Initialize or Set Allocated Memory

Purpose

DosSubSet is used to initialize a segment for suballocation or to increase the size of a previously initialized, suballocated segment.

Calling Sequence

```
EXTRN  DosSubSet:FAR

PUSH   WORD    SegSelector   ;Segment selector
PUSH   WORD    Flags         ;Parameter flags
PUSH   WORD    Size          ;Size of a block
CALL   DosSubSet
```

Where

SegSelector
is the selector of the target data segment.

Flags
is set to 1 indicating initializing a segment, or to 0, indicating increasing the size of a segment already initialized.

Size
is the size of the segment in bytes.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

To initialize a segment, issue DosSubSet before issuing DosSubAlloc and set flags = 1. The segment must have been previously allocated with DosAllocSeg or DosAllocShrSeg. To increase the size of a segment, issue DosReallocSeg before issuing DosSubSet. Failure to issue DosSubSet after changing the size of a segment may yield unpredictable results.

The size parameter should be a multiple of four bytes, or it will be rounded. Note in DosSubSet, a size parameter of 0 indicates the segment is 64K, while in DosSubAlloc and DosSubFree, a size parameter of 0 is an error. Other than this special case of 0 meaning 64KB, the minimum size which can be set is 12 bytes.

DosSuspendThread – Suspend Thread Execution

Purpose

DosSuspendThread temporarily suspends thread execution until a DosResumeThread call is made for that thread ID.

Calling Sequence

```
EXTRN  DosSuspendThread:FAR

PUSH   WORD    ThreadID      ;Thread ID of thread to suspend
CALL   DosSuspendThread
```

Where

ThreadID
is the Thread ID of the thread to be suspended.

Returns

IF   AX = 0 then NO error

ELSE AX = error code

Remarks

The specified thread may not be suspended immediately because it may have some system resources locked that should be freed first. However, the thread is not allowed to execute further application program instructions until a corresponding DosResumeThread is issued.

A thread can only suspend threads within its process.

DosTimerAsync – Start Asynchronous Time Delay

Purpose

DosTimerAsync starts a timer that runs asynchronously to the thread issuing the request and clears a system semaphore when the specified interval expires.

Calling Sequence

```
EXTRN  DosTimerAsync:FAR

PUSH   DWORD   TimeInterval  ;Interval size
PUSH   DWORD   SemHandle     ;System semaphore handle
PUSH@  WORD    Handle        ;Timer handle (returned)
CALL   DosTimerAsync
```

Where

TimeInterval
is the number of milliseconds (rounded up to the next clock tick) that passes before the semaphore is cleared.

SemHandle
is the handle of the system semaphore used to communicate the time out to the calling thread. This semaphore should be set by DosSemSet before DosTimerAsync is called.

Handle
is where the timer handle is returned. This handle may be passed to DosTimerStop to stop this timer before its time interval expires.

Returns

IF AX = 0 then NO error

ELSE AX = error code

DosTimerAsync – Start Asynchronous Time Delay

Remarks

DosTimerAsync is used to wait for a single asynchronous time. The thread waits for the timeout by issuing a DosSemWait.

This function is the asynchronous analog of DosSleep. It allows a thread to start a timer while it is performing another task. This timer can be cancelled by calling the DosTimerStop function with the timer handle returned by DosTimerAsync.

If another time out is needed, the semaphore is set and DosTimerAsync is reissued. To ensure reliable detection of the timer expiration, the system semaphore should be set prior to calling DosTimerAsync.

DosTimerStart – Start Periodic Interval Timer

Purpose

DosTimerStart starts a periodic interval timer that runs asynchronously to the thread issuing the request.

The semaphore is continually cleared at the specified time interval until the timer is turned off by DosTimerStop.

Calling Sequence

```
EXTRN  DosTimerStart:FAR

PUSH   DWORD   TimeInterval  ;Interval size
PUSH   DWORD   SemHandle     ;System semaphore handle
PUSH@  WORD    Handle        ;Timer handle (returned)
CALL   DosTimerStart
```

Where

TimeInterval
is the number of milliseconds (rounded up to the next clock tick) that passes before the semaphore is cleared.

SemHandle
is the handle of the system semaphore used to communicate the time out to the calling thread. This semaphore should be set by DosSemSet before the next clear by the timer.

Handle
is where the timer handle is returned. This handle may be passed to DosTimerStop to stop the periodic timer.

Returns

IF   AX = 0 then NO error

ELSE AX = error code

DosTimerStart — Start Periodic Interval Timer

Remarks

DosTimerStart allows a timer to start and run asynchronously to a thread. A timer interval is cancelled by using the timer handle in the DosTimerStop call. This prevents the semaphore indicated in the DosTimerStart call from being sent notifications.

The application detects the expirations of the timer when the semaphore is set prior to the next expiration of the timer. When an application waits for this semaphore to clear, more than one clearing of the timer may occur before the application resumes execution. If it is necessary to determine the actual elapsed time, the Global Information Segment milliseconds field can be saved prior to calling DosTimerStart. This saved value is compared to the current value when the process resumes. See "DosGetInfoSeg — Get Address of System Variables Segment" on page 2-83 for more information regarding the Global Information Segment.

DosTimerStop — Stop Interval Timer

Purpose

DosTimerStop stops a periodic interval timer started by DosTimerStart, or an asynchronous timer started by DosTimerAsync.

Calling Sequence

```
EXTRN  DosTimerStop:FAR

PUSH   WORD    Handle       ;Handle of the timer
CALL   DosTimerStop
```

Where

Handle
is the handle of the timer to be stopped.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

DosTimerStop is used to stop a periodic interval timer or asynchronous timer from running. No assumptions can be made about the state of the semaphore specified with DosTimerStart or DosTimerAsync. The application should put the semaphores into a known state.

DosUnlockSeg – Unlock Segment

Purpose
DosUnlockSeg unlocks a discardable segment.

Calling Sequence

```
EXTRN DosUnlockSeg:FAR

PUSH  WORD Selector ;Selector to unlock
CALL  DosUnlockSeg
```

Where

Selector
is the selector of the segment to be unlocked.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks
This function is valid only on segments which have been allocated through DosAllocSeg with AllocFlags bit 2 (0100B) set. Note that it is an error to unlock a segment that is already fully unlocked.

DosWrite – Synchronous Write to File

Purpose

DosWrite transfers the specified number of bytes from a buffer to the specified file, synchronously with respect to the requesting process's execution.

Calling Sequence

```
EXTRN  DosWrite:FAR

PUSH   WORD    FileHandle    ;File handle
PUSH@  OTHER   BufferArea    ;User buffer
PUSH   WORD    BufferLength  ;Buffer length
PUSH@  WORD    BytesWritten  ;Bytes written (returned)
CALL   DosWrite
```

Where

FileHandle
is the file handle obtained from DosOpen.

BufferAddress
is the output buffer.

BufferLength
is the number of bytes to write.

BytesWritten
is where the number of bytes written is returned.

Returns

IF AX = 0 then NO error

ELSE AX = error code

DosWrite – Synchronous Write to File

Family API Considerations

Some options operate differently in the DOS mode than they do in OS/2 mode. Therefore, the following restriction applies to DosWrite when coding in the DOS mode :

Use only single-byte DosWrites to COMx in PC/DOS. The COM device driver supplied with PC/DOS does not support multiple-byte I/O.

Remarks

Upon return from this function, BytesWritten is the number of bytes actually written. If BytesWritten is different from BufferLength this usually indicates insufficient disk space.

A BufferLength value of 0 is not considered an error. No data transfer will occur. There is no effect on the file or the file pointer.

Buffers that are multiples in size of the hardware's base physical unit for data (the base physical unit is defined as the smallest block that can be physically written to the device) which are written to the file on these base boundaries, are written directly to the device. Other buffer sizes force some of the I/O to go through an internal system buffer and greatly reduce the efficiency of the I/O operation.

The file pointer is moved to the desired position by reading, writing, and performing function DosChgFilePtr (Move File Read/Write Pointer).

If the file is read-only, the write to the file is not performed.

Purpose

DosWriteAsync transfers the specified number of bytes to a file from a buffer, asynchronously with respect to the requesting process's execution.

Calling Sequence

```
EXTRN  DosWriteAsync:FAR

PUSH   WORD    FileHandle    ;File handle
PUSH@  DWORD   RamSemaphore  ;RAM semaphore
PUSH@  WORD    ReturnCode    ;I/O error RC (returned)
PUSH@  OTHER   BufferArea    ;User buffer
PUSH   WORD    BufferLength  ;Buffer length
PUSH@  WORD    BytesWritten  ;Bytes written (returned)
CALL   DosWriteAsync
```

Where

FileHandle
is the file handle obtained from DosOpen.

RamSemaphore
is used by the system to signal the caller that the write operation is complete.

ReturnCode
is where the return code is returned.

BufferArea
is the output buffer.

BufferLength
is the number of bytes to be written.

BytesWritten
is where the number of bytes written is returned.

DosWriteAsync — Asynchronous Write to File

Returns

AX = 0

Note: When RamSemaphore is cleared and the read operation completes, ReturnCode can be checked.

Remarks

When RamSemaphore is cleared, BytesWritten identifies the number of bytes written. If BytesWritten is different from BufferLength it usually indicates insufficient disk space.

A BufferLength value of 0 is not considered an error. No data transfer will occur. There is no effect on the file or the file pointer.

RamSemaphore must be set by the application before the DosWriteAsync call is made. The application issues the following sequence:

1. DosSemSet
2. DosWriteAsync
3. DosSemWait.

Note: The program must not modify the contents of BufferArea or look at the values returned in ReturnCode or BytesWritten until after RamSemaphore is cleared.

Buffers that are multiples in size of the hardware's base physical unit for data (the base physical unit is defined as the smallest block that can be physically written to the device) which are written to the file on these base boundaries will be written directly to the device. Other buffer sizes will force at least some of the I/O to go through an internal system buffer (if the File State indicates that internal buffers may be used) and reduce the efficiency of the I/O operation.

The file read/write pointer can be moved to the desired position by reading, writing, or performing function DosChgFilePtr (Change File Read/Write Pointer).

The value of the file read/write pointer is updated by the File Level Request Router before the I/O request is queued to the device driver.

If the file is read-only, the write to the file is not performed.

DosWriteQueue – Write to Queue

Purpose

DosWriteQueue adds an element to a queue.

Calling Sequence

```
EXTRN  DosWriteQueue:FAR

PUSH   WORD     QueueHandle   ;Handle of queue to send to
PUSH   WORD     Request       ;Request identification data
PUSH   WORD     DataLength    ;Length of element being added
PUSH@  OTHER    DataBuffer    ;Element being added
PUSH   WORD     ElemPriority  ;Priority of element being added
CALL   DosWriteQueue
```

Where

QueueHandle
is the handle of the queue to write to.

Request
is a value to be passed with the queue element. This word is used for event encoding by the specific application. The data in this word is understood by the thread adding the element to the queue and by the thread which receives the queue element. There is no special meaning to this data and the operating system does not alter the data.

DataLength
is the length of the data being sent to the queue.

DataBuffer
is the buffer where the data, which is to be placed in the queue, is located.

ElemPriority
is the priority of the element being added to the queue. If the priority is specified as 15, the element is added to the top of the queue, If the priority is specified as 0, the element is added as the last element in the queue. Elements with the same priority are in *FIFO* order. This parameter is valid for priority type queues only.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

DosWriteQueue adds entries to a specified queue. If the owning process closes a queue prior to this request being issued, the QUEUE_DOES_NOT_EXIST, Invalid Queue Handle return code is returned. If the owning process invokes a system semaphore when DosReadQueue or DosPeekQueue are issued, other processes that issue DosWriteQueue for the used system semaphore must first issue DosOpenSem.

Chapter 3. OS/2 Keyboard Function Calls

This chapter reflects the Keyboard API interface only. For information regarding the keyboard IOCtl interface and keyboard monitor refer to the OS/2 Technical Reference Volume 1.

KbdCharIn – Read Character, Scan Code

Purpose
KbdCharIn returns a character data record from the keyboard.

Calling Sequence

```
EXTRN KbdCharIn:FAR

PUSH@  OTHER    CharData       ;Buffer for data
PUSH   WORD     IOWait         ;Indicate if wait
PUSH   WORD     KbdHandle      ;Keyboard handle
CALL   KbdCharIn
```

Where

CharData

is a structure that contains the character data in the following format:

Size	**Description**
BYTE	ASCII Character Code
BYTE	Scan Code
BYTE	Status
BYTE	NLS Shift Status
WORD	Shift State
DWORD	Time Stamp

ASCII Character Code
: is derived from translation of the scan code received from the keyboard.

Scan Code
: is code received from the keyboard identifying the key pressed. This scan code may have been modified during the translation process.

Status
: indicates the state of the character:

Bit Values	Function	Description
Bit 7 = 0	Bit 6 = 0	undefined
Bit 7 = 0	Bit 6 = 1	final character, interim character flag off
Bit 7 = 1	Bit 6 = 0	interim character
Bit 7 = 1	Bit 6 = 1	final character, interim character flag on
Bit 5 = 1		On the spot conversion requested
Bit 4 to 1		reserved = 0
Bit 0 = 1		shift status returned without character

NLS Shift Status
Reserved = 0.

Shift State
is the state of the shift keys. The states are defined as follows:

High Byte

Bit#	**Meaning**
15	SysReq Key Down
14	CapsLock Key Down
13	NumLock Key Down
12	ScrollLock Key Down
11	Right Alt Key Down
10	Right Ctrl Key Down
9	Left Alt Key Down
8	Left Ctrl Key Down

Low Byte

Bit#	**Meaning**
7	Insert ON
6	CapsLock ON
5	NumLock ON
4	ScrollLock ON
3	Either Alt Key Down

2	Either Ctrl Key Down
1	Left Shift Key Down
0	Right Shift Key Down

Time Stamp
is the time stamp of the key stroke that occupies a double word and is specified in milliseconds since IPL.

IOWait
indicates whether to wait if a character is not available.

If value = 0
the requestor will wait for a character if one is not available.
If value = 1
the requestor will get an immediate return if no character is available.

KbdHandle
identifies either the default keyboard or the logical keyboard.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Family API Considerations

Some options operate differently in the DOS mode than they do in the OS/2 mode. Therefore, the following restrictions apply to KbdCharIn when coding in the DOS mode :

- Interim Character is not supported
- Status can be 0 or 40H
- KbdHandle is ignored.

Remarks

Extended ASCII codes return the first code as 0DH or E0H in the character field and the second code (extended code) in the scan code field. Usually the extended ASCII code is the scan code of the primary key that was pressed.

Note: On an enhanced keyboard, the secondary enter key returns the normal character of 0Dh and a scan code of E0h.

Double-byte character codes (DBCS) require two function calls to obtain the entire code.

If shift report is set with KbdSetStatus, then the CharData Record returned will reflect changed shift information only.

KbdCharIn completes when the handle has access to the physical keyboard (focus), or is equal to 0, and no other handle has the focus.

The ASCII Character and Scan Code field of the *CharData Structure* are reserved and set to 0 when a shift state is received.

A returned character is indicated by the final character flag, (bit 6 of the status byte), being set to 1. Bit 6 set to 0 indicates that no character was returned.

In general, if a thread can not continue until a character is received, it should use the KbdCharIn function with the IOWait parameter set to wait for a character if one is not available. A thread in the foreground session that repeatedly polls the keyboard with KbdCharIn (with no wait), can prevent all regular priority class threads from executing. If polling must be used and a minimal amount of other processing is being performed, the thread should periodically yield the CPU by issuing a DosSleep for an interval of at least 5 milliseconds

KbdClose – Close a Logical Keyboard

Purpose

KbdClose ends the existing logical keyboard identified by the keyboard handle.

Calling Sequence

```
EXTRN KbdClose:FAR

PUSH  WORD   KbdHandle ;Keyboard handle
CALL  KbdClose
```

Where

KbdHandle
identifies either the default keyboard or the logical keyboard.

Returns

IF   AX = 0 then NO error

ELSE AX = error code

Remarks

The close process results in a KbdFreeFocus and KbdFlushBuffer on the handle. A KbdClose of a 0 handle has no effect on the default keyboard.

KbdDeRegister – Deregister Keyboard Subsystem

Purpose
KbdDeRegister deregisters a keyboard subsystem previously registered within a session. Only the process that issued the KbdRegister may issue KbdDeRegister.

Calling Sequence
```
EXTRN KbdDeRegister:FAR

CALL  KbdDeRegister
```

Where
None

Returns
IF   AX = 0 then NO error

ELSE AX = error code

Remarks
None

KbdFlushBuffer — Flush key stroke Buffer

Purpose
KbdFlushBuffer clears the key stroke buffer.

Calling Sequence

```
EXTRN KbdFlushBuffer:FAR

PUSH  WORD  KbdHandle   ;Keyboard handle
CALL  KbdFlushBuffer
```

Where

KbdHandle
identifies either the default keyboard or the logical keyboard.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Family API Considerations
Some options operate differently in the DOS mode than they do in the OS/2 mode. Therefore, the following restriction applies to KbdFlushBuffer when coding in the DOS mode :

- KbdHandle is ignored.

Remarks
KbdFlushBuffer completes when the handle has access to the physical keyboard (focus), or is equal to 0 and no other handle has the focus.

KbdFreeFocus – Free Keyboard Focus

Purpose
KbdFreeFocus frees the logical to physical keyboard bond created by KbdGetFocus.

Calling Sequence

```
EXTRN KbdFreeFocus:FAR

PUSH   WORD    KbdHandle      ;Keyboard handle
CALL   KbdFreeFocus
```

Where

KbdHandle
identifies either the default keyboard or the logical keyboard.

Returns

IF   AX = 0 then NO error

ELSE AX = error code

Remarks
If other threads are waiting for this bond to end, then one of the waiting threads will be allowed to complete it KbdGetFocus upon completion of the KbdFreeFocus. If no threads are waiting for the a bond, then the physical keyboard reverts to the default keyboard.

KbdFreeFocus may be replaced by issuing KbdRegister. Unlike other keyboard sub-system functions, the replaced KbdFreeFocus will be called only if there is an outstanding focus.

KbdGetCp – Get Loaded Code Page IDs

Purpose

KbdGetCp allows a process to query the code page currently in use to translate scan codes to ASCII characters.

Calling Sequence

```
EXTRN KbdGetCp:FAR

PUSH   DWORD   Reserved      ;Reserved
PUSH@  WORD    CodePageID    ;Code Page ID
PUSH   WORD    KbdHandle     ;Keyboard handle
CALL   KbdGetCp
```

Where

Reserved
is reserved and set to 0.

CodePageID
is located in the application's data area. The keyboard support copies the current code page ID for a specified keyboard handle into this word. The code page ID is equivalent to one of the code page IDs specified in the CONFIG.SYS CODEPAGE = statement or 0000.

KbdHandle
identifies either the default keyboard or the logical keyboard.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

KbdGetCp completes when the handle has access to the physical keyboard (focus), or is equal to 0 and no other handle has the focus.

KbdGetFocus – Get Keyboard Focus

Purpose

KbdGetFocus binds the logical keyboard to the physical keyboard.

Calling Sequence

```
EXTRN KbdGetFocus:FAR

PUSH   WORD    IOWait        ;Indicate if wait
PUSH   WORD    KbdHandle     ;Keyboard handle
CALL   KbdGetFocus
```

Where

IOWait

indicates whether to wait if a character is not available.

Bit#	**Meaning**
15-1	Reserved = 0
0	indicates whether caller wants to wait for the bond. If value = 0, wait If value = 1, do not wait.

KbdHandle

identifies either the default keyboard or the logical keyboard.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

The keyboard handle identifies which logical keyboard to bind to. If the physical keyboard is not bound to a logical or default keyboard, then the bind proceeds immediately. The logical keyboard, identified by the handle, receives all further key strokes from the physical keyboard. If the physical keyboard is already in use by a logical keyboard, then the thread issuing KbdGetFocus waits until the bond can be made. Waiting threads do not execute in any definable order.

KbdGetStatus – Get Keyboard Status

Purpose

KbdGetStatus gets the current state of the keyboard.

Calling Sequence

```
EXTRN   KbdGetStatus:FAR

PUSH@  OTHER   Structure      ;Data structure
PUSH   WORD    KbdHandle      ;Keyboard handle
Call   KbdGetStatus
```

Where

Structure

is a data structure that contains information used to set the Input Mode, Echo, Shift, Interim Character flags and the TurnAround character states to be assigned to the keyboard. The data structure is defined as follows:

Word 0

is the length in bytes of the data structure including this word. High Byte is reserved and equal to 0. Low Byte is equal to 10.

Word 1

is a bit mask that represents functions whose state is to be altered by the current KbdSetStatus call. The bit mask is defined as follows:

Bit #	***Meaning***
15-09	= Reserved, set to 0
08	= Shift Report ON
07	= length of turnaround character 0 = one byte, high byte only (ASCII) 1 = two bytes (extended ASCII) (meaningful only if bit 6 is on)
06	= modify TurnAround character
05	= modify Interim Character Flags
04	= modify Shift State
03	= ASCII mode ON
02	= Binary mode ON

01	= Echo OFF
00	= Echo ON

If both bits 0 and 1 are OFF, the Echo state of the system is not altered. If both bits 2 and 3 are OFF, the BINARY/ASCII state of the system is not altered. If both bits 0 and 1 are ON or if both bits 2 and 3 are ON, the function will return an error. Echo is ignored if BINARY mode is set.

Word 2

Define TurnAround Character

Character (ASCII, extended ASCII) defined as the Turnaround (Carriage Return) character. If the turnaround character is just ASCII then it is defined in the low order byte.

Word 3

Interim Character Flags

High Byte	= NLS Shift State
Low Byte	= Status
07	= Interim Character Flag ON
06	= Reserved, set to 0
05	= Program requested on-the-spot conversion
04-00	= Reserved, set to 0

Word 4

The shift states are defined as follows:

High Byte

15	= SysReq Key Down
14	= CapsLock Key Down
13	= NumLock Key Down
12	= ScrollLock Key Down
11	= Right Alt Key Down
10	= Right Ctrl Key Down
09	= Left Alt Key Down
08	= Left Ctrl Key Down

Low Byte

07	= Insert ON
06	= CapsLock ON
05	= NumLock ON
04	= ScrollLock ON
03	= Either Alt Key Down
02	= Either Ctrl Key Down
01	= Left Shift Key Down

00	= Right Shift Key Down

KbdHandle
is the handle of the KBD resource. OS/2 requires this value always be a reserved word of 0s.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Family API Considerations

Some options operate differently in the DOS mode than they do in the OS/2 mode. Therefore, the following restrictions apply to KbdGetStatus when coding in the DOS mode :

- Interim Character is not supported
- Turnaround character is not supported
- NLS Shift State will always be null
- KbdHandle is ignored.

Remarks

The initial state of the keyboard is established by the system at application load time. Some default states may be modified by the application through KbdSetStatus. KbdGetStatus will return only those keyboard parameters initially set by KbdSetStatus. The returned parameters are:

- Input Mode
- Interim Character Flags
- Shift State
- Echo State
- Turnaround Character

KbdGetStatus completes only when the handle has access to the physical keyboard (focus) or the handle is 0 and no other handle has the focus.

KbdOpen – Open a Logical Keyboard

Purpose

KbdOpen creates a new logical keyboard.

Calling Sequence

```
EXTRN KbdOpen:FAR

PUSH@  WORD    KbdHandle       ;Keyboard handle
CALL   KbdOpen
```

Where

KbdHandle
identifies either the default keyboard or the logical keyboard.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

A keyboard handle is returned by KbdOpen identifying a new logical keyboard. KbdOpen initializes the logical keyboard to the system default CodePage.

KbdPeek — Peek at Character, Scan Code

Purpose

KbdPeek returns the character data record, if available, from the keyboard without removing it from the buffer.

Calling Sequence

```
EXTRN KbdPeek:FAR

PUSH@  OTHER   CharData      ;Buffer for data
PUSH   WORD    KbdHandle     ;Keyboard handle
CALL   KbdPeek
```

Where

CharData

is a structure that contains the character data in the following format:

Size	**Description**
BYTE	ASCII Character Code
BYTE	Scan Code
BYTE	Status
BYTE	NLS Shift Status
WORD	Shift State
DWORD	Time Stamp

ASCII Character Code

is an ASCII character derived from translation of the scan code received from the keyboard.

Scan Code

is code received from the keyboard identifying the key pressed. This scan may have been modified during the translation process.

Status

indicates the state of the character:

Bit Values	Function	Description
Bit 7 = 0	Bit 6 = 0	undefined
Bit 7 = 0	Bit 6 = 1	final character, interim character flag off
Bit 7 = 1	Bit 6 = 0	interim character
Bit 7 = 1	Bit 6 = 1	final character, interim character flag on
Bit 5 = 1		On the spot conversion requested
Bit 4 to 1		reserved = 0
Bit 0 = 1		shift status returned without character

NLS Shift Status
Reserved = 0.

Shift State
is the state of the shift keys. The states are defined as follows:

Bit#	**Meaning**
15	SysReq Key Down
14	CapsLock Key Down
13	NumLock Key Down
12	ScrollLock Key Down
11	Right Alt Key Down
10	Right Ctrl Key Down
09	Left Alt Key Down
08	Left Ctrl Key Down
07	Insert ON
06	CapsLock ON
05	NumLock ON
04	ScrollLock ON
03	Either Alt Key Down
02	Either Ctrl Key Down
01	Left Shift Key Down
00	Right Shift Key Down

Time Stamp
is the time stamp of the key stroke occupying a double word and specified in milliseconds.

KbdHandle
identifies either the default keyboard or the logical keyboard.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Family API Considerations

Some options operate differently in the DOS mode than they do in the OS/2 mode. Therefore, the following restrictions apply to KbdPeek when coding for the DOS mode:

- Interim character is not supported
- Status = 0 or 40H
- KbdHandle is ignored.

Remarks

Extended ASCII codes return the first code as 0Dh or E0H in the character field and the second code (extended code) in the scan code field. Usually the extended ASCII code is the scan code of the primary key that was pressed.

Note: On an enhanced keyboard, the secondary enter key returns the normal character of 0Dh and a scan code of E0h.

Double-byte character codes (DBCS) require two function calls to obtain the entire code.

If shift report is set by KbdSetStatus then the CharData Record returned reflects only the changed shift information.

The ASCII Character and Scan Code. field of the CharData Structure are reserved and set to 0 when a shift state is received.

A returned character is indicated by the final character flag, (bit 6 of the status byte), being set to 1. Bit 6 set to 0 indicates that no character was returned.

KbdPeek completes when the handle has access to the physical keyboard (focus) or is equal to 0 and no other handle has the focus.

In general, if a thread can not continue until a character is received, it should use the KbdCharIn function with the IOWait parameter set to wait for a character if one is not available. A thread in the foreground session that repeatedly polls the keyboard with KbdPeek (with no wait), can prevent all regular priority class threads from executing. If polling must be used and a minimal amount of other processing is being performed, the thread should periodically yield the CPU by issuing a DosSleep for an interval of at least 5 milliseconds.

KbdRegister – Register Keyboard Subsystem

Purpose

KbdRegister registers a keyboard subsystem within a session.

Calling Sequence

```
EXTRN KbdRegister:FAR

PUSH@  ASCIIZ  ModuleName    ;Module name
PUSH@  ASCIIZ  EntryPoint    ;Entry point name
PUSH   DWORD   FunctionMask  ;Function mask
CALL   KbdRegister
```

Where

ModuleName
contains the dynamic link module name. Maximum length is 129 bytes (including ASCIIZ terminator).

EntryPoint
contains the dynamic link entry point name of a routine which will receive control when any of the registered functions are called. Maximum length is 33 bytes (including ASCIIZ terminator).

FunctionMask
is a bit mask where each bit identifies a keyboard function being registered. The bits are defined as follows:

Bit #	*Function*
31-14	Reserved = 0
13	KbdSetCustXt
12	KbdXlate
11	KbdSetCp
10	KbdGetCp
9	KbdFreeFocus
8	KbdGetFocus
7	KbdClose
6	KbdOpen
5	KbdStringIn
4	KbdSetStatus
3	KbdGetStatus
2	KbdFlushBuffer
1	KbdPeek
0	KbdCharIn

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

The Base Keyboard Subsystem is the keyboard subsystem default. There can be only one KbdRegister call outstanding at a time without an intervening KbdDeRegister. KbdDeRegister must be issued by the same process that issued the KbdRegister.

When any registered function is called, control is routed to EntryPoint. When this routine is entered, four additional values are pushed onto the stack. The first is the index number (WORD) of the routine being called. The second is a near pointer (WORD). The third is the caller's DS register (WORD). The fourth is the return address (DWORD) to the keyboard router. For example, if KbdFlushBuffer were a registered function and if KbdFlushBuffer were called and control routed to EntryPoint, the stack would appear as if the following instruction sequence were executed:

KbdRegister – Register Keyboard Subsystem

```
PUSH   WORD  KbdHandle
CALL   FAR   KbdFlushBuffer
PUSH   WORD  Index
CALL   NEAR  Entry point in Kbd router
PUSH   DS
CALL   FAR   Dynamic link entry point
```

The index numbers that correspond to the registered functions are listed below:

00 KbdCharIn	07 KbdClose
01 KbdPeek	08 KbdGetFocus
02 KbdFlushBuffer	09 KbdFreeFocus
03 KbdGetStatus	10 KbdGetCp
04 KbdSetStatus	11 KbdSetCp
05 KbdStringIn	12 KbdXlate
06 KbdOpen	13 KbdSetCustXt

When a registered function returns to the keyboard router, the contents of AX are interpreted as follows:

AX = 0
: no error. Do not invoke the corresponding Base Keyboard Subsystem. Return AX = 0 to the caller.

AX = -1
: invoke the corresponding Base Keyboard Subsystem. Return AX = return code from the Base Keyboard Subsystem.

AX = error, (not 0 or -1)
: do not invoke the corresponding Base Keyboard Subsystem. Return AX = error to the caller.

Within a session, a KbdRegister call remains in effect until a KbdDeRegister is issued or the session ends. KbdDeRegister must be called by the same process that issued KbdRegister.

KbdSetCp – Set the Code Page

Purpose

KbdSetCp allows the process to set the code page used to translate key strokes received from the keyboard.

Calling Sequence

```
EXTRN KbdSetCp:FAR

PUSH   WORD    Reserved        ;Reserved
PUSH   WORD    CodePageID      ;code page ID
PUSH   WORD    KbdHandle       ;Keyboard handle
CALL   KbdSetCp
```

Where

Reserved
is a reserved word equal to 0.

CodePageID
is a word in the application's data area. It represents a code page id. The code page ID word must be equivalent to one of the code page IDs specified on the CONFIG.SYS CODEPAGE = statement or 0000. If the code page ID does not match one of the ids on the CODEPAGE = statement, an error will result. The code page word currently must have one of the following code page identifiers:

Identifier	**Description**
0000	Resident code page
0437	IBM PC US 0437
0850	Multilingual
0860	Portuguese
0863	Canadian-French
0865	Nordic

KbdHandle
identifies either the default keyboard or the logical keyboard.

KbdSetCp – Set the Code Page

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

The code page specified must be 0000 or have been specified on the CONFIG.SYS CODEPAGE = statement. Value = 0000 indicates to use the device driver's built in code page.

To clear key strokes translated with the previous code page, the buffer is flushed when the code page switch completes.

KbdSetCp completes when the handle has access to the physical keyboard (focus), or is equal to 0 and no other handle has the focus.

Purpose

KbdSetCustXt installs on the specified handle the code page pointed to in this call. This code page will affect only this handle.

Calling Sequence

```
EXTRN KbdSetCustXt:FAR

PUSH@  OTHER   CodePageID    ;Translation Table
PUSH   WORD    KbdHandle     ;Keyboard handle
CALL   KbdSetCustXt
```

Where

CodePageID
is a translation table used to translate scan code to ASCII code for a specified handle. The format of the code page is documented in the Set Code Page IOCtl 50H. (Refer to Chapter 3 in this book for a complete discussion of Set Code Page IOCtl 50H).

KbdHandle
identifies either the default keyboard or the logical keyboard.

Returns

IF   AX = 0 then NO error

ELSE AX = error code

Remarks

The code page must be maintained in the callers memory. No copy of the code page is made by KbdSetCustXt.

Note: KbdSetCp reverses the action of KbdSetCustXt and sets the handle equal to one of the system code pages. If memory is dynamically allocated by the caller for the code page and is freed before the KbdSetCp is performed, KbdSetCp and future translations may fail.

KbdSetCustXt completes when the handle has access to the physical keyboard (focus), or is equal to 0 and no other handle has the focus.

KbdSetFgnd — Set Foreground Keyboard Priority

Purpose
KbdSetFgnd raises the priority of the foreground keyboard's thread.

Calling Sequence

```
EXTRN  KbdSetFgnd:FAR

CALL   KbdSetFgnd
```

Where
None

Returns

IF   AX = 0 then NO error

ELSE AX = error code

Remarks

Note: KbdSetFgnd is used by a subsystem, not an application.

KbdSetStatus – Set Keyboard Status

Purpose

KbdSetStatus sets the characteristics of the keyboard.

Calling Sequence

```
EXTRN   KbdSetStatus:FAR

PUSH@  OTHER   Structure     ;Data structure
PUSH   WORD    KbdHandle     ;Keyboard Handle
Call   KbdSetStatus
```

Where

Structure

is a data structure that contains information to set the Input Mode, Echo, Shift, Interim Character Flags and the TurnAround Character States to be assigned to the keyboard. The data structure is defined as follows:

Word 0

is the length in bytes of the data structure including this word High Byte is reserved and equal to 0. Low Byte is equal to 10.

Word 1

is a bit mask that represents functions whose state is to be altered by the current KbdSetStatus call. The bit mask is defined as follows:

Bit #	***Meaning***
15-09	= Reserved, set to 0
08	= Shift Report ON
07 =	length of turnaround character 0 = one byte, high byte only (ASCII) 1 = two bytes (extended ASCII) (meaningful only if bit six is on)
06	= TurnAround character is to be modified
05	= Interim Character Flags are to be modified
04	= Shift state is to be modified
03	= ASCII mode ON
02	= Binary mode ON

01	= Echo OFF
00	= Echo ON

If both bits 0 and 1 are OFF, the Echo state of the system is not altered. If both bits 2 and 3 are OFF, the BINARY/ASCII state of the system is not altered. If both bits 0 and 1 are ON or if both bits 2 and 3 are ON, the function will return an error. Echo is ignored if BINARY mode is set.

Word 2

Define TurnAround Character

Character (ASCII, extended ASCII) defined as the Turnaround (Carriage Return) character. If the turnaround character is just ASCII then it is defined in the low order byte.

Word 3

Interim Character Flags

High Byte	= NLS Shift State
Low Byte	= Status
07	= Interim Character Flag ON
06	= Reserved, set to 0
05	= Program requested on-the-spot conversion
04-00	= Reserved, set to 0

Word 4

Shift State

The shift states are defined as follows:

15	= SysReq Key Down
14	= CapsLock Key Down
13	= NumLock Key Down
12	= ScrollLock Key Down
11	= Right Alt Key Down
10	= Right Ctrl Key Down
09	= Left Alt Key Down
08	= Left Ctrl Key Down
07	= Insert ON
06	= CapsLock ON
05	= NumLock ON
04	= ScrollLock ON
03	= Either Alt Key Down

02	= Either Ctrl Key Down
01	= Left Shift Key Down
00	= Right Shift Key Down

KbdHandle
identifies either the default keyboard or the logical keyboard.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Family API Considerations

Some options operate differently in the DOS mode than they do in the OS/2 mode. Therefore, the following restrictions apply to KbdSetStatus when coding in the DOS mode :

- RAW Mode, and ECHO ON is not supported and will return an error if requested
- KbdHandle is ignored
- Interim character is not supported
- Turnaround character is not supported.

Remarks

KbdSetStatus completes when the handle has access to the physical keyboard (focus), or is equal to 0 and no other handle has the focus.

Note: Shift report is not valid in ASCII mode.

When turning off shift report the application must realize that there may be remaining shift report CharDataRecords to be read. If the application does not want to process these it should do a KbdFlushBuffer after turning off the shift report function.

KbdStringIn – Read Character String

Purpose

KbdStringIn reads a character string (character codes only) from the keyboard.

Calling Sequence

```
EXTRN  KbdStringIn:FAR

PUSH@  OTHER   CharBuffer    ;Char string buffer
PUSH@  OTHER   Length        ;Length table
PUSH   WORD    IOWait        ;Indicate if wait for char
PUSH   WORD    KbdHandle     ;Keyboard handle
CALL   KbdStringIn
```

Where

CharBuffer
is the buffer for the character string.

Length
is the length of the character string buffer. On entry Length is the maximum number of bytes in the buffer. The maximum Length can be specified is 255 bytes.

Length is defined by the following structure:

Size	Description
WORD	Input Buffer Length
WORD	Received Input Length

Note: Template processing has meaning only in the ASCII mode of operation.

IOWait
indicates whether to wait if a character is not available.

0 = Wait
In BINARY input mode, the requestor waits until CharBuffer is full. In ASCII input mode, the requestor waits until a carriage return is struck.

1 = No Wait
The requestor gets an immediate return if no characters are available. If characters are available, KbdStringIn returns immediately with as many characters as are available (up to the maximum). No Wait is not supported for ASCII input mode.

KbdHandle
identifies either the default keyboard or the logical keyboard.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Family API Considerations

Some options operate differently in the DOS mode than they do in the OS/2 mode. Therefore, the following restrictions apply to KbdStringIn when coding in the DOS mode :

- KbdHandle is ignored.
- ECHO state must always be set ON.

See also "DosRead – Read from File" on page 2-191 for the differences between coding in OS/2 mode and DOS mode when reading from a handle opened to the device "CON."

Remarks

The character strings may be optionally echoed on the display if echo mode is set. When echo is on each character is echoed as it is read from the keyboard. Echo mode and BINARY mode are mutually exclusive. Reference KbdSetStatus and KbdGetStatus for more information.

In ASCII mode, 2-byte character codes only return in complete form. An extended ASCII code is returned in a 2-byte string. The first byte is 0Dh or E0H and the next byte is an extended code.

In input mode (BINARY, ASCII), The following returns can be set and retrieved via KbdSetStatus and KbdGetStatus:

Turnaround Character
Echo Mode
Interim Character Flag
Shift State

The default input mode is ASCII.

The received input length is also used by the KbdStringIn line edit functions for re-displaying and entering a caller specified string. On the next KbdStringIn call the received input length indicates the length of the input buffer that may be recalled by the user using the line editing keys. A value of 0 inhibits the line editing function for the current KbdStringIn request.

KbdStringIn completes when the handle has access to the physical keyboard (focus), or is equal to 0 and no other handle has the focus.

KbdSynch — Synchronize Keyboard Access

Purpose

KbdSynch synchronizes access for a keyboard subsystem to the keyboard device driver.

Calling Sequence

```
EXTRN KbdSynch:FAR

PUSH   WORD    IOWait        ;Indicate if wait
CALL   KbdSynch
```

Where

IOWait

indicates whether to wait if a character is not available.

Bit #	**Meaning**
15-01	Reserved = 0
0	Indicates whether requestor waits for access to the device driver. If bit = 1, wait, if bit = 0, do not wait.

Returns

IF   AX = 0 then NO error

ELSE AX = error code

Remarks

KbdSynch requests an exclusive system semaphore. See "DosCloseSem — Close System Semaphore" on page 2-19 for more information regarding system semaphores. This semaphore request clears when the subsystem returns to the keyboard router. KbdSynch blocks all other threads within a session until the semaphore clears (returns from the subsystem to the router). To ensure proper synchronization, KbdSynch should be issued by a keyboard subsystem if it intends to issue DosDevIOCtl or access dynamically modifiable per-session shared data. KbdSynch will not protect globally shared data from threads in other sessions.

KbdXlate – Translate Scan Code

Purpose

KbdXlate translates scan code and shift states into ASCII code.

Calling Sequence

```
EXTRN KbdXlate:FAR

PUSH@  OTHER   XlateRecord    ;Translation Record
PUSH   WORD    KbdHandle      ;Keyboard handle
CALL   KbdXlate
```

Where

XlateRecord

is a structure that contains the translation record in the following format:

Function	Value	
CharData Record	as defined in KbdCharIn	10 BYTES
KbdDDFlags	as defined for Monitor packets	WORD
Xlate Flags	defined below	WORD
Xlate State 1	defined below	WORD
Xlate State 2	defined below	WORD

Xlate Flags

High Byte
    Bits 8-15, Reserved = 0
Low Byte
    Bits 1-7, Reserved = 0
    Bit 0 = Translation complete

Xlate State 1 and Xlate State 2
identifies the state of translation across successive calls. Initially these words should be 0. They should be reset to 0 to start a new translation.

Note: It may take several calls to this IOCtl to complete a character. These field should not be revised unless fresh start to translation is desired.

KbdHandle
identifies either the default keyboard or the logical keyboard.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

The desired shift state to use for translation is located in the CharDataRecord.

KbdXlate completes when the handle has access to the physical keyboard (focus), or is equal to 0 and no other handle has the focus.

Note: It may take several calls to complete a translation due to accent key combinations, etc.

Chapter 4. OS/2 Mouse Function Calls

For information regarding mouse device drivers, mouse pointer draw device, mouse installation and mouse IOCtls, refer to the OS/2 Technical Reference, Volume 1

MouClose – Close Mouse Device

Purpose

MouClose closes the mouse device for the current session.

Calling Sequence

```
EXTRN  MouClose:FAR

PUSH   WORD    DeviceHandle  ;Mouse device handle
CALL   MouClose
```

Where

DeviceHandle
is the handle of the mouse device obtained from a previous MouOpen.

Returns

IF   AX = 0 then NO error

ELSE AX = error code

Remarks

MouClose closes the mouse device for the current session and removes the mouse device driver handle from the list of valid open mouse device handles.

MouDeRegister – Deregister a Subsystem

Purpose

MouDeRegister deregisters a mouse subsystem previously registered within a session.

Calling Sequence

```
EXTRN  MouDeRegister:FAR

CALL   MouDeRegister
```

Where

No parameters are passed.

Returns

IF   AX = 0 then NO error

ELSE AX = error code

Remarks

MouDeRegister causes mouse calls for the session to revert to the Base Mouse Subsystem

Processes issuing MouDeRegister calls must conform to the following rules:

- The process which issued the MouRegister must release the session (by a MouDeRegister), from the registered subsystem before another PID may issue MouRegister.
- The process which issued the MouRegister is the only process which may issue MouDeRegister against the currently registered subsystem.
- Once the owning process has released the subsystem with a MouDeRegister, any other process in the session may issue a MouRegister and therefore modify the mouse support for the entire session.

MouDrawPtr – Mouse Draw Pointer

Purpose

MouDrawPtr allows a process to notify the mouse device driver that an area previously restricted to the pointer image is now available to the mouse device driver.

Calling Sequence

```
EXTRN  MouDrawPtr:FAR

PUSH   WORD    DeviceHandle  ;Mouse device handle
CALL   MouDrawPtr
```

Where

DeviceHandle
is the handle of the mouse device obtained from a previous MouOpen.

Returns

IF   AX = 0 then NO error

ELSE AX = error code

Remarks

The collision area (the pointer image restricted area) is established by MouOpen and by MouRemovePtr. MouDrawPtr nullifies the collision area. If there was no collision area at the time the MouDrawPtr was called, the MouDrawPtr is a null operation.

Immediately after MouOpen is issued, the collision area is defined as the size of the display. A MouDrawPtr can be issued to begin pointer drawing after the MouOpen.

MouFlushQue — Flush Mouse Queue

Purpose
MouFlushQue directs the mouse driver to flush (empty) the mouse event queue for the session.

Calling Sequence
```
EXTRN  MouFlushQue:FAR

PUSH   WORD    DeviceHandle  ;Mouse device handle
CALL   MouFlushQue
```

Where
DeviceHandle
is the handle of the mouse device obtained from a previous MouOpen.

Returns
IF   AX = 0 then NO error

ELSE AX = error code

Remarks
MouFlushQue is implemented via the Generic IOCtl, Category 0BH, option 01H "flush input buffer" call.

MouGetDevStatus – Get Mouse Device Status

Purpose

Returns status flags for the mouse device driver currently installed.

Calling Sequence

```
EXTRN  MouGetDevStatus:FAR

PUSH@  WORD    DeviceStatus  ;Current status flags
PUSH   WORD    DeviceHandle  ;Mouse device handle
CALL   MouGetDevStatus
```

Where

DeviceStatus

is where the current status flag settings are returned to the application for the currently installed mouse device driver.

The return value is a 2-byte set of bit flags.

High Byte:

Bit #	**Meaning**
07-02	-Reserved = 0
01	-set if mouse data returned in mickeys, not display units
00	-set if the interrupt level pointer draw routine is not called

Low Byte:

Bit #	**Meaning**
07-04	-Reserved = 0
03	-set if pointer draw routine disabled by unsupported mode
02	-set if flush in progress
01	-set if block read in progress
00	-set if event queue busy with I/O

DeviceHandle

is the handle of the mouse device obtained from a previous MouOpen.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

None

MouGetEventMask – Get Mouse Event Mask

Purpose

MouGetEventMask returns the current value of the mouse event queue mask.

Calling Sequence

```
EXTRN  MouGetEventMask:FAR

PUSH@  WORD    EventMask     ;Event Mask word
PUSH   WORD    DeviceHandle  ;Mouse device handle
CALL   MouGetEventMask
```

Where

EventMask

is where the current mouse device drivers event mask is returned to the caller by the mouse device driver.

The EventMask is set by MouSetEventMask, and has the following definition:

Bit No.	**Meaning**
07-15	- Reserved = 0
06	- set to receive button 3 press/release events
05	- set to receive button 3 press/release events, with mouse motion
04	- set to receive button 2 press/release events
03	- set to receive button 2 press/release events, with mouse motion
02	- set to receive button 1 press/release events
01	- set to receive button 1 press/release events, with mouse motion
00	- set to receive mouse motion events with no button press/release events.

DeviceHandle

contains the handle of the mouse device obtained from a previous MouOpen.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

Note: Buttons are logically numbered from left to right.

MouGetNumButtons — Get Number of Mouse Buttons

Purpose

MouGetNumButtons returns the number of buttons supported on the mouse driver currently installed.

Calling Sequence

```
EXTRN  MouGetNumButtons:FAR

PUSH@  WORD  NumberOfButtons ;Number of mouse buttons
PUSH   WORD  DeviceHandle    ;Mouse device handle
CALL   MouGetNumButtons
```

Where

NumberOfButtons
is where the number of physical buttons on the mouse is returned.

DeviceHandle
contains the handle of the mouse device obtained from a previous MouOpen.

Returns

IF   AX = 0 then NO error

ELSE AX = error code

Remarks

The return values for the number of buttons supported are:

1 = One mouse button
2 = Two mouse buttons
3 = Three mouse buttons

MouGetNumMickeys – Get Number of Mouse Mickeys

Purpose

MouGetNumMickeys returns the number of mickeys per centimeter for the mouse driver currently installed.

Calling Sequence

```
EXTRN  MouGetNumMickeys:FAR

PUSH@  WORD  NumberOfMickeys ;Number mickeys/centimeter
PUSH   WORD  DeviceHandle    ;Mouse device handle
CALL   MouGetNumMickeys
```

Where

NumberOfMickeys
is where the number of mickeys (mouse movement units) per centimeter of screen height or width are returned by the mouse support.

DeviceHandle
contains the handle of the mouse device obtained from a previous MouOpen.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

The return value is dependent on the mouse and its current setting.

MouGetNumQueEl – Get Event Queue Status

Purpose

MouGetNumQueEl returns the current status for the mouse device driver event queue.

Calling Sequence

```
EXTRN  MouGetNumQueEl:FAR

PUSH@  OTHER  QueDataRecord  ;Ptr to 2-word structure
PUSH   WORD   DeviceHandle   ;Mouse device handle
CALL   MouGetNumQueEl
```

Where

QueDataRecord

is a structure in application storage where queue status is returned. This structure receives 2 full word parameters as follows:

Word 0 =
the current number of event queue elements. The number of queue elements value is in the range of 0 <= value <= MaxNumQueElements.

Word 1 =
the mouse configured MaxNumQueElements value.

DeviceHandle

contains the handle of the mouse device obtained from a previous MouOpen.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

The MaxNumQueElements returned by this function is established during mouse device driver configuration. See DEVICE=MOUSExxx in the IBM Operating System/2 User Reference.

Purpose
MouGetPtrPos queries the mouse driver to determine the current row and column coordinate position of the mouse pointer shape.

Calling Sequence
```
EXTRN  MouGetPtrPos:FAR

PUSH@  OTHER   PtrPos        ;Double word structure
PUSH   WORD    DeviceHandle  ;Mouse device handle
CALL   MouGetPtrPos
```

Where
PtrPos

is a structure in application storage where position information is returned. The structure format of the returned position information is defined as follows:

- Word 0 = current pointer row coordinate screen position.
- Word 1 = current pointer column coordinate screen position.

Both parameters are in either pixel or character units, depending on the mode of the display for that session.

Coordinate positions are relative to physical displacement from the top left corner of the display screen.

DeviceHandle

contains the handle of the mouse device obtained from a previous MouOpen.

Returns
IF AX = 0 then NO error

ELSE AX = error code

Remarks
None

MouGetPtrShape – Get Pointer Shape

Purpose

MouGetPtrShape allows a process to get (copy) the pointer shape for the session.

Calling Sequence

```
EXTRN  MouGetPtrShape:FAR

PUSH@  OTHER   PtrBuffer    ;Pointer shape buffer
PUSH@  OTHER   PtrDefRec    ;Pointer definition structure
PUSH   WORD    DeviceHandle ;Mouse device handle
CALL   MouGetPtrShape
```

Where

PtrBuffer

is an area in application storage where the pointer draw device driver returns the pointer bit image.

PtrDefRec

is a structure in application storage where the application stores the necessary data for the pointer draw device driver to build a Row by Col image for each bit plane for the currently running display mode.

The pointer definition record structure is described below:

Word 0 = TotLength
Word 1 = Col
Word 2 = Row
Word 3 = ColOffset
Word 4 = RowOffset

TotLength

contains the total length in bytes necessary for the pointer draw device driver to build a Row by Column image for each bit plane for the currently running display mode. The length of the pointer buffer is supplied by the calling application. The actual length of the pointer image is always returned in this word. If the total length of the pointer buffer supplied by the application is not large enough to hold the pointer image, an error is returned. In this case, the caller can use the value returned in TotLength to determine the buffer size needed.

Note: See "MouSetPtrShape – Set Mouse Pointer Shape" on page 4-38 for the calculations necessary to determine the size.

Col
  is a full word returned by pointer draw device driver:
  For graphics modes:
    it contains the pixel width (cols) of the mouse shape for the session.
  For text modes:
    will be equal to 1.

Row
  is a full word returned by the pointer draw device driver:
  For graphics modes:
    pixel height (row) of the mouse shape for the session. Must be greater than or equal to 1.
  For text modes:
    row must be equal to 1.

ColOffset
  is returned by the pointer draw device to indicate the relative column pixel offset for the pointer image used for coordinate tracking. This defines the column coordinate for the pointer image hotspot. This value is a signed number that represents character or pixel offset, depending on the display mode.

RowOffset
  is returned by the pointer draw device to indicate the relative row pixel offset for the pointer image used for coordinate tracking. This defines the row coordinate for the pointer image hotspot. This value is a signed number that represents character or pixel offset, depending on the display mode.

Note: For text modes, row and column offset will equal 0.

DeviceHandle
contains the handle of the mouse device obtained from a previous MouOpen.

MouGetPtrShape – Get Pointer Shape

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

The application passes a parameter list with the same meaning as defined for MouSetPtrShape, to the mouse device driver. The mouse device driver copies the parameters that describe the pointer shape and attributes into the pointer definition control block pointed to by the PtrDefRec parameter. The word 0 (buffer length = TotLength) pointer definition record parameter field must contain the size in bytes of the application buffer in which the device driver is to insert the sessions pointer image. All other words in the parameter list are returned to the application by MouGetPtrShape.

If the buffer size is insufficient, the TotLength field will contain the actual size in bytes of the returned pointer image.

The pointer shape may be set by the application via MouSetPtrShape or may be the default image provided by the installed Pointer Device Driver.

Refer to the OS/2 Technical Reference Volume I for more information concerning the Pointer Draw device driver.

Purpose

MouGetScaleFact returns a pair of 1-word scaling factors for the current mouse device.

Calling Sequence

```
EXTRN  MouGetScaleFact:FAR

PUSH@  OTHER   ScaleStruct   ;2-word structure
PUSH   WORD    DeviceHandle  ;Mouse device handle
CALL   MouGetScaleFact
```

Where

ScaleStruct

is where the mouse device driver's current row and column coordinate scaling factors are returned to the caller by the mouse device driver.

ScaleStruct is a control block structure that is written into by the mouse support. The format of the ScaleStruct structure is:

Word 0

=RowScale, the current row coordinate scaling factor. Rowscale falls within the following limits:

1 <= return value <= (32k - 1)

Word 1

= ColScale, the current column coordinate scaling factor. ColScale falls within the following limits:

1 <= return value <= (32k - 1)

See "MouSetScaleFact – Set Mouse Scaling Factor" on page 4-42 for more information.

DeviceHandle

contains the handle of the mouse device obtained from a previous MouOpen.

MouGetScaleFact — Get Mouse Scaling Factors

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

None

MouInitReal – Initialize DOS mode

Purpose
MouInitReal initializes the DOS mode mouse device driver.

Calling Sequence

```
EXTRN  MouInitReal:FAR

PUSH@  ASCIIZ  DriverName    ;Pointer draw driver name
CALL   MouInitReal
```

Where

DriverName

contains the name of the Mouse Pointer Draw Device Driver used as the pointer image drawing routine for the DOS mode session.

The name of the device driver must be included in the CONFIG.SYS file at system boot time. Applications that use the default Pointer Draw Device Driver supplied by the system, must push a double word of 0es in place of an address.

Currently the Shell uses the default image drawing routine supplied by the system and, consequently, places a double word of 0s in place of an address.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks
MouInitReal is issued by the Shell (System Session Manager) at shell initialization time.

The DOS mode mouse API (INT 33H), in contrast to the OS/2 mode Mouse API, does not contain an OPEN command. In addition, there may be only one session for DOS mode and, therefore, only one DOS mode mouse support.

MouInitReal – Initialize DOS mode

The default pointer draw routine for the DOS mode is located in the screen pointer draw module used in the OS/2 mode. The OS/2 mode mouse support knows the address of this screen pointer draw routine. Establishing addressability to the screen pointer draw routine for the DOS mode mouse driver must be done at run time. The entry point of the screen pointer draw routine must be passed to the DOS mode mouse device driver. MouInitReal passes the address of the default screen pointer draw routine to the DOS mode device driver at system initialization time so it is transparent to applications.

MouOpen – Open Mouse Device

Purpose

MouOpen opens the mouse device for the current session.

Calling Sequence

```
EXTRN  MouOpen:FAR

PUSH@  ASCIIZ  DriverName    ;Pointer draw driver name
PUSH@  WORD    DeviceHandle  ;Mouse device handle
CALL   MouOpen
```

Where

DriverName

contains the name of the Mouse Pointer Draw Device Driver used as the pointer image drawing routine for the session.

The name of the device driver must be included in the CONFIG.SYS file at system start-up time. Applications that use the default Pointer Draw Device Driver supplied by the system must push a double word of 0es in place of an address.

Applications that use the default image drawing routine supplied by the system must push a double word of 0s in place of an address.

DeviceHandle

is where the mouse support returns a 1 word value that represents the mouse handle to the application.

Returns

IF AX = 0 then NO error

ELSE AX = error code

MouOpen – Open Mouse Device

Remarks

MouOpen generates a new mouse device handle every time it is called.

MouOpen initializes the Mouse functions to a known state. The application may have to issue additional mouse functions to establish the environment it desires. For example, after the MouOpen, the collision area is defined to be the size of the entire display. Therefore, to get the pointer to be displayed, the application must issue a MouDrawPtr to remove the collision area.

The initial state of the mouse is:

- Row/Col scale factors set to 16/8. (See "MouSetScaleFact – Set Mouse Scaling Factor" on page 4-42.)
- all events reported. (See "MouSetEventMask – Set Mouse Event Mask" on page 4-34.)
- empty event queue. (See "MouReadEventQue – Read Mouse Event Queue" on page 4-23 and "MouGetNumQueEl – Get Event Queue Status" on page 4-12.)
- all user settable Device Status bits reset. (Set to 0. See "MouSetDevStatus – Set Mouse Device Status" on page 4-32.)
- pointer set to center of screen. (See "MouSetPtrPos – Set Mouse Pointer Position" on page 4-36.)
- pointer shape set to the default for the pointer device driver currently registered in the session. (See "MouSetPtrShape – Set Mouse Pointer Shape" on page 4-38.)
- collision area equal to full screen. (See "MouDrawPtr – Mouse Draw Pointer" on page 4-4 and "MouRemovePtr – Remove Mouse Pointer" on page 4-30.)

Purpose

MouReadEventQue reads an event from the mouse device FIFO event queue, and places it in a structure provided by the application.

Calling Sequence

```
EXTRN  MouReadEventQue:FAR

PUSH@  OTHER   Buffer        ;10 byte Structure address
PUSH@  WORD    ReadType      ;Read type
PUSH   WORD    DeviceHandle  ;Mouse device handle
CALL   MouReadEventQue
```

Where

Buffer

is a 10 byte structure in application storage where the *FIFO* event record element from the mouse event queue for the currently installed mouse device driver will be returned to the application. The return data has the following format:

```
MouState   WORD  (see below)
EventTime  DWORD (time since boot in milliseconds)
Row        WORD  (absolute/relative row position)
Col        WORD  (absolute/relative col position)
```

MouState

represents the state of the mouse at the time the event is reported. This word is defined as follows:

Bit #	**Meaning**
7-15 -	Reserved = 0
6 -	set if, button 3 down
5 -	set if, mouse motion, button 3 down
4 -	set if, button 2 down
3 -	set if, mouse motion, button 2 down
2 -	set if, button 1 down
1 -	set if, mouse motion, button 1 down
0 -	set if, mouse motion, no buttons down

MouReadEventQue – Read Mouse Event Queue

ReadType

indicates the action to take when MouReadEventQue is issued and the mouse event queue is empty. If the mouse event queue is not empty, this parameter is not examined by the mouse support. The ReadType values follow:

0 - No *WAIT* for data on empty queue (return a *NULL* record)
1 - *WAIT* for data on empty queue

DeviceHandle

contains the handle of the mouse device obtained from a previous MouOpen.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

The types of queued events are directly affected by the current value of the Mouse EventMask. MouSetEventMask is used to indicate the types of events desired, and MouGetEventMask is used to query the current value of the mask. Refer to these functions for further explanation of the masking of events.

Recognition of the mouse transition depends on the use of MouState returned in the event record. The application should focus on bit transitions which occur in this word. It is important to properly set the event mask with MouSetEventMask for reporting the state transitions.

MouState reports the state of the mouse which resulted from the action which caused the event. The action can be pressing or releasing a button, and/or moving the mouse. All status is given, regardless of the EventMask which was used to determine whether or not to report the event.

For example, assume the EventMask indicates that the application wishes only button 1 event. The EventMask will have only bits 1 and 2 set in this case. Also assume the current state of the mouse is no buttons down, and mouse is not moving. At this point, button 1 is pressed causing an event; the status will show button 1 down (bit 2 set). Next the mouse is moved, thereby causing more events; status will show bits 1 set. Finally, mouse is stopped and button 1 is released. The event will show status with no bits set.

Next, button 2 is pressed. No event occurs. Mouse is then moved; again, no event. Then, while mouse is still in motion, button 1 is pressed; an event is generated with bits 1 and 3 set in the state word. While mouse is still in motion, both buttons are released. Because button 1 changes states, an event occurs. The state word will have bit 0 set. Finally, mouse is stopped. No event occurs, again because no button 1 transition has taken place.

MouRegister – Register a Subsystem

Purpose

MouRegister registers a mouse subsystem within a session.

Calling Sequence

```
EXTRN   MouRegister:FAR

PUSH@  ASCIIZ  ModuleName     ;Module Name
PUSH@  ASCIIZ  EntryName      ;Entry Name
PUSH   DWORD   Mask           ;Function Mask
CALL   MouRegister
```

Where

ModuleName
contains the dynamic link module name. The maximum length is 129 bytes (including ASCIIZ terminator).

EntryName
contains the dynamic link entry point name of a routine that receives control when any of the registered functions are called. The maximum length is 33 bytes (including ASCIIZ terminator)

Mask
is a mask of bits, where each bit set to 1 identifies a Mouse function being registered. The bit mask format is shown below:

Bit #	Function Registered
31-21	-Reserved = 0
20	-MouSetDevStatus
19	-MouInitReal
18	-MouSetPtrPos
17	-MouGetPtrPos
16	-MouRemovePtr
15	-MouDrawPtr
14	-MouSetPtrShape
13	-MouGetPtrShape
12	-MouClose
11	-MouOpen
10	-Reserved

09	-Reserved
08	-MouSetEventMask
07	-MouSetScaleFact
06	-MouGetEventMask
05	-MouGetScaleFact
04	-MouReadEventQue
03	-MouGetNumQueEl
02	-MouGetDevStatus
01	-MouGetNumMickeys
00	MouGetNumButtons

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

The Base Mouse Subsystem is the default mouse subsystem. There can be only one MouRegister outstanding at a time without an intervening MouDeRegister. MouDeRegister must be issued by the same process that issued MouRegister.

When any registered function is called, control is routed to EntryName. When this routine is entered, four additional values are pushed onto the stack. The first is the index number (WORD) of the function being called. The second is a near pointer (WORD). The third is the caller's DS register (WORD). The fourth is the return address (DWORD) to the mouse router. For example, if MouGetNumMickeys were called and control routed to EntryName, the stack would appear as if the following instructions were executed:

```
PUSH@ WORD NumberOfMickeys
PUSH  WORD DeviceHandle
CALL  FAR  MouGetNumMickeys
PUSH  WORD Function Code
CALL  NEAR entry point in Mouse Router
PUSH  DS
CALL  FAR  EntryName
```

MouRegister – Register a Subsystem

When a registered function returns to the Mouse Router, AX is interpreted as follows:

AX = 0
: No error. Do not invoke the Base Mouse Subsystem routine. Return AX = 0.

AX = -1
: Invoke the Base Mouse Subsystem routine. Return AX = return code from the Base Mouse Subsystem.

AX = error (not 0 or -1)
: Do not invoke the Base Mouse Subsystem routine. Return AX = error.

When the mouse API router receives a mouse call, it routes it to the Base Mouse Subsystem unless an application or other mouse subsystem has previously issued MouRegister for that call. If the call was registered, the subsystem is entered at the EntryName specified, and provided with the applicable function code.

The registered function mask is used to determine whether a requested function will be performed by the registered mouse subsystem or default to the Base Mouse Subsystem.

The following table shows the relationship of the mouse API calls and the function code passed to either the Base Mouse Subsystem or a registered mouse subsystem.

MOU API Call	**Function Code**
MouGetNumButtons	00
MouGetNumMickeys	01
MouGetDevStatus	02
MouGetNumQueEl	03
MouReadEventQue	04
MouGetScaleFact	05
MouGetEventMask	06
MouSetScaleFact	07
MouSetEventMask	08
Reserved	09
Reserved	0A
MouOpen	0B

MouClose	0C
MouGetPtrShape	0D
MouSetPtrShape	0E
MouDrawPtr	0F
MouRemovePtr	10
MouGetPtrPos	11
MouSetPtrPos	12
MouInitReal	13
MouSetDevStatus	14

A registered mouse subsystem must leave the stack, on exit, in the exact state it was received.

MouRemovePtr – Remove Mouse Pointer

Purpose

MouRemovePtr allows a process to notify the mouse device driver that the area defined by the passed parameters is for the exclusive use of the application. This area is defined as the 'collision' area and is not available to the mouse device driver when drawing pointer images.

Calling Sequence

```
EXTRN MouRemovePtr:FAR

PUSH@  OTHER   PtrArea       ;Address of pointer data block
PUSH   WORD    DeviceHandle  ;Mouse device handle
CALL   MouRemovePtr
```

Where

PtrArea

is where the application provides a data structure to define the collision area. This structure is as follows:

- UpLeftRow WORD Upper left row (pixels or chars) coordinates (word)
- UpLeftCol WORD Upper left col (pixels or chars) coordinates (word)
- LowRightRow WORD Lower Right row (pixels or chars) coordinates (word)
- LowRightCol WORD Lower Right col (pixels or chars) coordinates (word)

which the mouse device driver uses to define the collision area for the session pointer shape.

Neither the upper left corner (UpLeftRow , UpLeftCol) nor the lower right corner (LoRightRow, LoRighCol) of the collision area coordinate pair may overrun the display. An error will result if either of these coordinate pairs logically extends beyond the screen boundaries.

Row and col values may be specified in either pixels or character units as long as the orientation is preserved across all values in the structure and the dimensional choice corresponds to the current display mode.

DeviceHandle
contains the handle of the mouse device obtained from a previous MouOpen.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

The application passes data to the mouse device driver to inform the mouse driver that the collision area described by the data is to be exclusively under the control of the application. This collision area is not to be modified by the mouse device driver. This causes the mouse device driver to remove the mouse pointer image from the screen if it is currently located within the collision area. The mouse device driver will not draw the pointer image any time the image's logical location lies within the collision area.

MouRemovePtr may be issued by any process in the session. However, only one collision area is active at a time. Each MouRemovePtr command has the effect of resetting the collision area to the location and area specified by the current command. Previously defined collision areas are replaced and no longer checked for.

If the logical pointer position is outside of the collision area specified by the latest MouRemovePtr command, the pointer image will be drawn.

The MouDrawPtr command effectively cancels the MouRemovePtr command and allows the pointer to be drawn anywhere on the screen, until a new MouRemovePtr command is issued.

MouSetDevStatus – Set Mouse Device Status

Purpose

MouSetDevStatus sets the mouse device driver status flags for the mouse device driver currently installed. The status flags are a 2 byte set of bit flags.

Calling Sequence

```
EXTRN  MouSetDevStatus:FAR

PUSH@  WORD    DeviceStatus  ;Status flags
PUSH   WORD    DeviceHandle  ;Mouse device handle
CALL   MouSetDevStatus
```

Where

DeviceStatus

is where the application places the desired status flag settings.

The passed parameter is a 2 byte set of flags. Only the high order byte has any meaning.

High byte

Bit #	***Meaning***
07-02	-Reserved = 0
01	-set if, mouse device is to return data in mickeys
00	-set if, the interrupt level pointer draw routine is not to be called at interrupt time

Low byte:

Bit #	**Meaning**
07-00	-Reserved = 0

DeviceHandle

contains the handle of the mouse device obtained from a previous MouOpen.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

MouSetDevStatus is the complement to MouGetDevStatus. However, not all status flags may be set with MouSetDevStatus. Only the flags corresponding to the following functions may be modified:

- return data in mickeys

  Normally, mouse data is returned to the application with the absolute display mode coordinates of the pointer image position on the display screen. By setting this status flag, mouse data will be returned in relative mickeys, a wait of mouse movement.
- don't call pointer draw device

  Normally, the interrupt level screen pointer draw routine is called at interrupt time. The pointer draw routine projects the pointer image on the screen and then returns to the mouse device driver. By setting this status flag, the mouse device driver will not call the pointer draw routine. The application must draw any required pointer image on the screen.

MouSetEventMask – Set Mouse Event Mask

Purpose

MouSetEventMask assigns a new event mask to the current mouse device driver.

Calling Sequence

```
EXTRN   MouSetEventMask:FAR

PUSH@  WORD    EventMask     ;Mouse device event mask ptr
PUSH   WORD    DeviceHandle  ;Mouse device handle
CALL   MouSetEventMask
```

Where

EventMask

indicates what mouse events are to be placed on the event queue.

The EventMask bit values are described below:

Bit #	**Meaning**
07-15	- Reserved = 0
06	- set to receive button 3 press/release events
05	- set to receive button 3 press/release events, with mouse motion
04	- set to receive button 2 press/release events
03	- set to receive button 2 press/release events, with mouse motion
02	- set to receive button 1 press/release events
01	- set to receive button 1 press/release events, with mouse motion
00	- set to mouse motion events with no button press/release events

DeviceHandle

contains the handle of the mouse device obtained from a previous MouOpen.

Returns

IF   AX = 0 then NO error

ELSE AX = error code

Remarks

Setting a bit in the event mask means that the associated event will be reported on the mouse FIFO event queue. See "MouReadEventQue — Read Mouse Event Queue" on page 4-23 for examples of event mask use.

MouSetPtrPos – Set Mouse Pointer Position

Purpose

MouSetPtrPos directs the mouse driver to set a new current row and column coordinate position for the mouse pointer shape.

Calling Sequence

```
EXTRN  MouSetPtrPos:FAR

PUSH@  OTHER   PtrPos        ;Double word structure
PUSH   WORD    DeviceHandle  ;Mouse device handle
CALL   MouSetPtrPos
```

Where

PtrPos

structure format follows:

Word 0 =
pointer row coordinate screen position.

Word 1 =
pointer column coordinate screen position.

Both parameters must be in pixel or character units, depending on the mode setting of the display in the session.

Coordinate positions are relative to physical displacement from the top left corner of the display screen.

DeviceHandle

contains the handle of the mouse device obtained from a previous MouOpen.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

The application must insure that the coordinate position specified conforms to the current display mode orientation for the session. Pixel values must be used for graphics modes and character values for text modes.

This function has no effect on the display's current collision area definition as specified by the MouDrawPtr call. If the mouse pointer image is directed into a defined collision area, the pointer image will not be drawn until either enough pointer movement has been generated to locate the pointer image beyond the collision area or the collision area is released by the MouDrawPtr call.

MouSetPtrShape – Set Mouse Pointer Shape

Purpose

MouSetPtrShape allows a process to set the pointer shape and size to be used as the mouse device driver pointer image for all applications in a session.

Calling Sequence

```
EXTRN  MouSetPtrShape:FAR

PUSH@  OTHER   PtrBuffer     ;Pointer shape buffer
PUSH@  OTHER   PtrDefRec     ;Pointer definition record
PUSH   WORD    DeviceHandle  ;Mouse device handle
CALL   MouSetPtrShape
```

Where

PtrBuffer

is where the application stores the bit image the mouse device driver uses as the pointer shape for that session. The buffer consists of AND and XOR pointer masks in a format meaningful to the Pointer Draw Device Driver.

For CGA compatible text modes (0, 1, 2, and 3) the following describes the *AND* and *XOR* pointer mask bit definitions for each character cell of the masks.

Bits	Meaning
15 -	Blinking
14-12 -	Background Color
11 -	Intensity
10-8 -	Foreground Color
7-0 -	Character

PtrDefRec

is a structure in application storage where the application stores the necessary data for the pointer device driver to build an Row by Column image for each bit plane for the currently running display mode.

The pointer definition record structure is described below:

Word 0	= TotLength
Word 1	= Col
Word 2	= Row
Word 3	= ColOffset
Word 4	= RowOffset

TotLength

contains the total length in bytes of the data necessary for the Mouse Pointer Draw Device Driver to build a Row by Column image for each bit plane for the currently running display mode.

The following example illustrates how to compute the TotLength for the system supplied Mouse Pointer Draw Device Driver:

```
Mono and Text -

   TotLength = (height in characters) *
                     (width in characters) * 2 * 2
             = 1 * 1 * 2 * 2
             = 4

Note:  as stated above, for text mode height and width must be 1,
so length is always 4.

   Graphics -

   TotLength = (height in pixels)*
               (width in pixels)*(bits per pixel) * 2 / 8.

Note:  width-in-pixels must be a multiple of 8.

Modes 4 and 5 (320 x 200)

   TotLength = (height) * (width) * 2 * 2 / 8.

Mode 6 (640 x 200)

   TotLength = (height) * (width) * 1 * 2 / 8.
```

Col

for graphics modes: is a full-word that contains the pixel width (columns) of the mouse shape for the session. This width must be greater than or = to 1.

for text modes, column must = 1.

Row

for graphics modes: is a full-word that contains the pixel height (row) of the mouse shape for the session. Row must be greater than or = to 1.

For Text modes, Row must = 1.

ColOffset

is the relative column pixel offset for the pointer image which is used for coordinate tracking. This defines the column coordinate for the pointer image hotspot. This value is a signed number that represents character or pixel offset, depending on the display mode.

RowOffset

is the relative row pixel offset for the pointer image which is used for coordinate tracking. This defines the row coordinate for the pointer image hotspot. This value is a signed number that represents pixel offset, depending on the display. Length calculations produce byte boundary buffer sizes.

For other custom displays and for the extended modes of the EGA attachment, it is possible to initialize the display to modes that require multiple bit planes. In these cases, the area sized by the Row and Col limits must be repeated for each bit plane supported in that mode. Consequently, the calling process must supply enough data to allow the mouse device driver to draw the pointer shape on all currently supported bit planes in that session.

Note: For text modes, row and column offset must equal 0.

DeviceHandle

contains the handle of the mouse device obtained from a previous MouOpen.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

An application passes a data image to the mouse device driver, which the mouse driver applies to the screen whenever the logical pointer position is not located in the application-defined collision area. The application synchronizes use of the screen with the mouse driver by way of MouRemovePtr and MouDrawPtr.

The pointer shape is dependent on the display device driver used to support the display device. OS/2 supports text and graphics modes. These modes are restricted to modes 0 through 7, depending on the display device. Character modes (modes 0, 1, 2, 3, and 7) support the pointer cursor only as a reverse block character. This reverse block character has a character height and width = 1.

The pointer shape is mapped by the Pointer Draw Device Driver and determined completely by the application. The height and width may vary from 1 through the pixel size of the display screen. For restrictions concerning the Pointer Draw Device Driver, see OS/2 Technical Reference, Volume 1.

Note: The current pointer shape in effect for the session may be determined with "MouGetPtrShape – Get Pointer Shape" on page 4-14.

MouSetScaleFact — Set Mouse Scaling Factor

Purpose

MouSetScaleFact assigns to the current mouse device driver a new pair of 1-word scaling factors.

Calling Sequence

```
EXTRN  MouSetScaleFact:FAR

PUSH@  OTHER   ScaleStruct   ;2-word structure
PUSH   WORD    DeviceHandle  ;Mouse device handle
CALL   MouSetScaleFact
```

Where

ScaleStruct

is where the mouse device driver's new row and column coordinate scaling factors are obtained by the mouse device driver.

ScaleStruct is a data structure supplied by the application. The format of the ScaleStruct structure follows:

Word 0 = RowScale

the new row coordinate scaling factor. RowScale must conform to the following limits:

1 <= value <= (32k-1)

Word 1 = ColScale

the new column coordinate scaling factor. ColScale must conform to the following limits:

1 <= value <= (32k-1)

DeviceHandle

contains the handle of the mouse device obtained from a previous MouOpen.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

MouSetScaleFact sets the mickey to pixel ratio for mouse motion. The row scale and column scale ratios specify a number of mickeys per 8 pixels. The default value for the row scale is 8 mickeys per 8 pixels. The default value for the column scale is 16 mickeys to 8 pixels.

The number of pixels moved, does not have to correspond 1 to 1 with the number of mickeys the mouse moves. The scaling factor defines a sensitivity for the mouse, which is a ratio of the number of mickeys required to move the cursor 8 pixels on the screen. The sensitivity determines at what rate the cursor moves on the screen.

MouSynch — Get Synchronous Access

Purpose

MouSynch provides synchronous (serial) access for a mouse subsystem to the mouse device driver.

Calling Sequence

```
EXTRN  MouSynch:FAR

PUSH   WORD    IOWait        ;Indicate wait/no wait
CALL   MouSynch
```

Where

IOWait

indicates whether to wait for access.

The flagword bits are defined as follows:

	Description
15-1 -	reserved equal to 0
0 -	indicates whether caller wants to wait if mouse device is busy
If bit 0 = 1	requestor waits until mouse device driver is free
If bit 0 = 0	control immediately returned to caller

Returns

IF   AX = 0 then NO error

ELSE AX = error code

Remarks

MouSynch requests an exclusive system semaphore (See "DosCloseSem — Close System Semaphore" on page 2-19). This semaphore request clears when the subsystem returns to the Mouse Router. MouSynch blocks all other threads within a session until the semaphore clears (returns from the subsystem to the router). To ensure proper synchronization, MouSynch should be issued by a mouse subsystem if it intends to access dynamically modifiable per-session shared data or if it intends to issue a DosDevIOCtl. MouSynch will not protect globally shared data from threads in other sessions.

Chapter 5. OS/2 Video Function Calls

This chapter reflects the video API interface only. Each function call reflects the most frequently occurring error codes only. For an extensive listing of all error codes, refer to the Appendix at the back of this book.

For information regarding other functional characteristics of the video API, refer to the IBM Operating System/2 Technical Reference, Volume 1

VioDeRegister — DeRegister Video Subsystem

Purpose

VioDeRegister deregisters a video subsystem previously registered within a session. VioDeRegister must be issued by the same process that issued the previous VioRegister. After VioDeRegister is issued, subsequent video calls are processed by the Base Video Subsystem.

Calling Sequence

```
EXTRN  VioDeRegister:FAR

CALL   VioDeRegister
```

Where

none

Returns

```
IF     AX = 0 then NO error code

ELSE AX = error code
```

Remarks

none

Purpose

VioEndPopUp is issued by the application when it no longer requires the temporary screen obtained through a previous VioPopUp call.

Calling Sequence

```
EXTRN   VioEndPopUp:FAR

PUSH    WORD     VioHandle        ;Vio device handle
CALL    VioEndPopUp
```

Where

VioHandle
is a reserved word of 0s.

Returns

IF   AX = 0 then NO error

ELSE AX = error code

Remarks

When the application issues a VioEndPopUp call, all video calls are directed to the application's normal video buffer.

VioGetAnsi — Get ANSI Status

Purpose

VioGetAnsi returns the current ANSI status On/Off state

Calling Sequence

```
EXTRN  VioGetAnsi:FAR

PUSH@  WORD   Indicator   ;On/Off indicator (returned)
PUSH   WORD   VioHandle   ;Vio handle
CALL   VioGetAnsi
```

Where

Indicator
is where the current ANSI status is returned. A value of 1 indicates *ANSI* is active, and a value of 0 indicates *ANSI* is not active.

VioHandle
is a reserved word of 0s.

Returns

IF   AX = 0 then NO error

ELSE AX = error code

Remarks

None

VioGetBuf – Get Logical Video Buffer

Purpose

VioGetBuf returns the address of the logical video buffer (*LVB*).

Calling Sequence

```
EXTRN  VioGetBuf:FAR

PUSH@  DWORD   LVBPtr         ;Points to LVB
PUSH@  WORD    Length         ;Length of buffer
PUSH   WORD    VioHandle      ;Vio handle
CALL   VioGetBuf
```

Where

LVBPtr
contains the selector and offset of the logical video buffer. Applications should not assume the offset portion of this far address will be 0.

Length
is the length of the returned buffer in bytes. The length is

```
Number of rows * Number of columns * 2
```

VioHandle
is a reserved word of 0s.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

With VioGetBuf, an application can prepare a screen in the application's own logical video buffer (LVB) offline. When the application is in the foreground, the physical screen buffer is updated from the LVB when VioShowBuf is issued. When the application runs in the background, the physical screen buffer is updated when the application is switched to the foreground.

VioGetBuf – Get Logical Video Buffer

Once VioGetBuf is issued, all VioWrtXX calls issued while the application is running in the foreground are written to the physical display buffer and LVB.

VioGetBuf is not supported in graphics modes.

Use VioGetMode to determine the dimensions of the buffer.

Purpose

VioGetConfig returns the video display configuration.

Calling Sequence

```
EXTRN  VioGetConfig:FAR

PUSH   WORD    Reserved    ;Reserved (must be 0)
PUSH@  OTHER   ConfigData  ;Configuration data
PUSH   WORD    VioHandle   ;Vio handle
CALL   VioGetConfig
```

Where

Reserved
is a word of 0s.

ConfigData
is a structure where the display configuration is returned.

Size	**Description**
WORD	Length
WORD	Adapter type
WORD	Display type
DWORD	Memory

Length
is an input parameter to VioGetConfig. It specifies the length of the data structure in bytes including itself. For OS/2 the maximum size structure required is 10 bytes.

Adapter Type
is the display adapter type.

- 0 = reserved
- 1 = color graphics adapter
- 2 = enhanced graphics adapter
- 3 = VGA or IBM Personal System/2™ Display Adapter
- 4-6 = reserved
- 7 = IBM Personal System/2™ Display Adapter 8514/A

VioGetConfig – Get Video Configuration

Display Type
  is the display/monitor type.
  - 00 = reserved
  - 01 = color display
  - 02 = enhanced color display
  - 03 = IBM Personal System/2 Monochrome Display 8503
  - 04 = IBM Personal System/2 Color Displays 8512 and 8513
  - 05-08 = reserved
  - 09 = IBM Personal System/2 Color Display 8514

Memory
  is the amount of memory on the adapter in bytes. It is returned as a 32-bit value.

VioHandle
is a reserved WORD of 0s.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

The values returned for Adapter Type and Display Type specify the configuration the Base Video Subsystem assumes is present. The Base Video Subsystem determines this information by making various tests, for example, testing the switch settings on the card. This interface does not guarantee that the display corresponding to the display type returned is actually present. There may be no monitor attached to the adapter. Also, if the switch settings on the card are set inappropriately, the Base Video Subsystem (BVS) may assume that one display type is present when another is in its place.

Purpose

VioGetCp allows a process to query the code page currently used to display text data.

Calling Sequence

```
EXTRN  VioGetCp:FAR

PUSH   WORD    Reserved      ;
PUSH@  WORD    CodePageID    ;Code page ID
PUSH   WORD    VioHandle     ;Video handle
CALL   VioGetCp
```

Where

Reserved
is a reserved word of 0s.

CodePageID
is a word in the application's data area. The current video code page is returned in this word.

VioHandle
is a reserved word of 0s.

Returns

IF   AX = 0 then NO error

ELSE AX = error code

Remarks

The display code page ID previously set by VioSetCp or inherited from the requesting process is returned to the caller.

The code page tag returned will be the currently active code page. A value of 0000 indicates that the code page in use is the ROM code page provided by the hardware.

VioGetCurPos – Get Cursor Position

Purpose
VioGetCurPos returns the cursor position.

Calling Sequence
```
EXTRN  VioGetCurPos:FAR

PUSH@  WORD    Row            ;Row return data
PUSH@  WORD    Column         ;Column return data
PUSH   WORD    VioHandle      ;Vio handle
CALL   VioGetCurPos
```

Where

Row
is the current row position of the cursor where 0 is the top row.

Column
is the current column position of the cursor where 0 is the leftmost column.

VioHandle
is a reserved word of 0s.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks
None

Purpose

VioGetCurType returns the cursor type.

Calling Sequence

```
EXTRN  VioGetCurType:FAR

PUSH@  OTHER   CursorData    ;Cursor characteristics
PUSH   WORD    VioHandle     ;Vio handle
CALL   VioGetCurType
```

Where

CursorData
is where characteristics of the cursor are returned.

Size	**Description**
WORD	Cursor start line
WORD	Cursor end line
WORD	Cursor width
WORD	Cursor attribute

CursorStartLine
is the horizontal scan line in the character cell which marks the top line of the cursor. If the character cell has "n" scan lines, 0 is the top scan line of the character cell and"n"-1 is the bottom scan line.

CursorEndLine
is the horizontal scan line in the character cell which marks the bottom line of the cursor. Scan lines within a character cell are numbered as defined under CursorStartLine.

CursorWidth
is the width of the cursor in columns. The maximum value supported by the OS/2 Base Video Subsystem is 1. CursorWidth = 0, specifies the default width (one column)

CursorAttrib
is the attribute of the cursor.

- −1 = hidden

- Any other value is normal.

VioHandle
is a reserved word of 0s.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Family API Considerations

Some options operate differently in the DOS mode than they do in the OS/2 mode. Therefore, the following restriction applies to VioGetCurType when coding in the DOS mode :

VioGetCurType returns only two values for CursorAttrib; 0 = visible cursor, and -1 = hidden cursor.

Remarks

None

Purpose

VioGetFont returns either the font table of the size specified, or the font currently in use.

Calling Sequence

```
EXTRN  VioGetFont:FAR

PUSH@  OTHER   RequestBlock  ;Request block
PUSH   WORD    VioHandle     ;Vio handle
CALL   VioGetFont
```

Where

RequestBlock

is a data structure that contains the request. The content of the structure varies depending upon the request type. The request type is in the second word. The formats of the supported request blocks are shown below. The symbols in the rightmost column are defined as follows:

- I = input parameter
- O = output parameter

	Get Current Font (EGA, VGA, or IBM Personal System/2 Display Adapter	
WORD	Length of structure (in bytes including length itself) = 14	I
WORD	Request type = 0, get current font	I
WORD	Pel columns in character cell	O
WORD	Pel rows in character cell	O
DWORD	A caller-supplied data area where the requested font table is returned. If the specified address is 0, a system-supplied segment that contains the requested font table is returned.	I/O

VioGetFont – Get Font

	Get Current Font (EGA, VGA, or IBM Personal System/2 Display Adapter	
WORD	Length in bytes of the caller-supplied data area where the font table is returned.	I/O

	Get ROM Font (CGA, EGA, VGA, or or IBM Personal System/2 Display Adapter)	
WORD	Length of structure (in bytes including length itself) = 14.	I
WORD	Request type = 1, get ROM font.	I
WORD	Pel columns in character cell	I
WORD	Pel rows in character cell	I
DWORD	a caller- supplied data area where the requested font table is returned. If the specified address is 0, a system-supplied segment that contains the requested font table is returned.	I/O
WORD	Length in bytes of the caller-supplied data area where the font table is returned.	I/O

VioHandle
is a reserved word of 0s.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

For request type = one, return ROM font, the font size requested must be supported by the display adapter installed. The 8x8, 8x14, 9x14, 8x16, or 9x16 character font may be requested for the VGA or IBM Personal System/2 Display Adapters adapters. The 8x8, 8x14, or 9x14 font may be requested for the enhanced graphics adapter. The 8x8 font may be requested for the color graphics adapter.

For request type = one, return ROM font, the far address returned will be a ROM pointer only for those fonts where the font table for the full 256-character set is actually contained in ROM. Otherwise, the far address returned will be a RAM pointer. Note that for 8x8 the font table for the full 256-character set is returned. For 9x14 or 9x16 the font table for the full 256-character set is also returned. Partial fonts are not returned. The 9x14 and 9x16 fonts are derived from variations of the 8x14 and 8x16 fonts, respectively, where the fonts for those characters which are different are replaced.

For VioGetFont specifying request type = one, return ROM font, the font returned is derived from the fonts contained in the system, EGA, VGA, and IBM Personal System/2 Display Adapter BIOS data areas as applicable. One exception is: for the EGA, VGA and IBM Personal System/2 Display Adapter, if VioSetCp has been issued, the font of the size requested from the active code page is returned.

VioGetMode – Get Display Mode

Purpose

VioGetMode returns the mode of the display.

Calling Sequence

```
EXTRN  VioGetMode:FAR

PUSH@  OTHER   ModeData      ;Mode characteristics
PUSH   WORD    VioHandle     ;Vio handle
CALL   VioGetMode
```

Where

ModeData

is the structure where mode characteristics are returned.

Size	**Description**
WORD	Length
BYTE	Type
BYTE	Color
WORD	Text Columns
WORD	Text Rows
WORD	Horizontal Resolution
WORD	Vertical Resolution

Length

is an input parameter to VioGetMode. Length specifies the length of the data structure in bytes including length itself. The value specified on input controls the amount of mode data returned. The minimum structure size required is 3 bytes, and the maximum structure size required is 12 bytes. Any value specified for Length other than 3 must be an even number.

Type

is a bit mask of mode characteristics. The definitions of the bits follows:

```
xxxxxxxb b = 0 monochrome compatible mode
         b = 1 other
xxxxxxbx b = 0 text mode
         b = 1 graphics mode
xxxxxbxx b = 0 enable color burst
         b = 1 disable color burst
```

Color
is the number of colors defined as a power of 2. This is equivalent to the number of color bits that define the color. For example:

- Color = 1 specifies 2 colors
- Color = 2 specifies 4 colors
- Color = 4 specifies 16 colors

Text Columns
are the number of text columns.

Text Rows
are the number of text rows.

Horizontal Resolution
is the number of pel columns.

Vertical Resolution
is the number of pel rows.

VioHandle
is a reserved word of 0s.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

Refer to "VioSetMode — Set Display Mode" on page 5-67 for examples.

VioGetPhysBuf — Get Physical Display Buffer

Purpose

VioGetPhysBuf gets addressability to the physical display buffer.

Calling Sequence

```
EXTRN  VioGetPhysBuf:FAR

PUSH@  OTHER   Structure     ;Data structure
PUSH   WORD    Reserved      ;Reserved (must be 0)
CALL   VioGetPhysBuf
```

Where

Structure

is a data structure that contains the physical display buffer address and length on input and the selectors, used to address the display buffer, on output. The data structure is defined as follows:

Size	Description
DWORD	Buffer start address
DWORD	Buffer length
OTHER	Selector list

Buffer Start Address

is the physical display buffer specified as a 32-bit physical address.

Buffer Length

is the 32-bit length of the physical display buffer.

Selector List

is where the selectors (each of word-length) used to address the physical display buffer are returned. The first selector returned in the list addresses the first 64K of the physical display buffer or Buffer Length, whichever is smaller. If Buffer Length is greater than 64K bytes, the second selector addresses the second 64K bytes, and so on. The last selector returned in the list addresses the remainder of the display buffer. The application is responsible for ensuring enough space is reserved for Selector List to accommodate the specified Buffer Length.

Reserved
is a word of 0s.

Returns

IF   AX = 0 then NO error

ELSE AX = error code

Remarks

An application uses VioGetPhysBuf to get addressability to the physical display buffer. The selector returned by VioGetPhysBuf may be used only when an application program is executing in the foreground. When an application wants to access the physical display buffer, the application must call VioScrLock. VioScrLock either waits until the program is running in the foreground or returns a warning when the program is running in the background. For more information refer to "VioScrLock – Lock Screen" on page 5-48 and "VioScrUnLock – Unlock Screen" on page 5-58

The buffer range specified for the physical screen buffer must fall between A0000 and BFFFF inclusive. An application may issue VioGetPhysBuf only when it is running in the foreground. An application may issue VioGetPhysBuf more than once.

VioGetState — Get Video State

Purpose

VioGetState returns the current settings of the palette registers, over-scan (border) color or blink/background intensity switch.

Calling Sequence

```
EXTRN  VioGetState:FAR

PUSH@  OTHER   RequestBlock  ;Request block
PUSH   WORD    VioHandle     ;Vio handle
CALL   VioGetState
```

Where

RequestBlock

is a data structure that contains the request. The content of the structure varies depending on the request type. The request type is in the second word. The formats of the request blocks supported are shown below. The symbols in the right column have the following meanings:

- I = input parameter
- O = output parameter

	Get Palette Registers (EGA, VGA, or IBM Personal System/2 Display Adapter)	
WORD	Length of structure (in bytes including length itself) maximum length = 38.	I
WORD	Request type = 0, Get palette registers	I
WORD	First palette register to return. Must be in range 0 to 15. The palette registers are returned in sequential order. The number of palette registers returned is based upon the length of the structure.	I

	Get Palette Registers (EGA, VGA, or IBM Personal System/2 Display Adapter)	
1 or more	1 WORD that contains the color value for each palette register returned.	O

	Get Overscan (Border) Color (CGA, VGA, or IBM Personal System/2 Display Adapter)	
WORD	Length of structure (in bytes including length itself) = 6	I
WORD	Request type = 1, get overscan (border) color	I
WORD	Color value	O

	Get Blink/Background Intensity Switch (CGA, EGA, VGA, or IBM Personal System/2 Display Adapter)	
WORD	Length of structure (in bytes including length itself) = 6	I
WORD	Request type = 2, get blink/ background intensity switch	I
WORD	Value = 0, blinking foreground colors enabled. Value = 1, high intensity background colors enabled.	O

VioHandle
is a reserved word of 0s.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

None

Purpose

VioModeUndo allows one thread within a process to cancel a VioModeWait issued by another thread within the same process.

Calling Sequence

```
EXTRN  VioModeUndo:FAR

PUSH   WORD    OwnerIndic     ;Ownership indicator
PUSH   WORD    KillIndic      ;Terminate indicator
PUSH   WORD    Reserved       ;Reserved (must be 0)
CALL   VioModeUndo
```

Where

OwnerIndic

indicates whether the thread issuing VioModeUndo wants ownership of VioModeWait to be reserved for its process.

- If OwnerIndic = 0, reserve ownership
- If OwnerIndic = 1, give up ownership.

KillIndic

indicates whether the thread (with the outstanding VioModeWait) should be returned an error code or be terminated.

- If KillIndic = 0, return error code
- If KillIndic = 1, terminate thread.

Reserved

is a reserved word of 0s.

Returns

IF AX = 0 then NO error

ELSE AX = error code

VioModeUndo — Restore Mode Undo

Remarks

VioModeUndo may be issued only by a thread within the process which owns VioModeWait. The thread issuing VioModeUndo can either reserve ownership of the VioModeWait function for its process or give up ownership. The thread whose VioModeWait is cancelled is optionally terminated.

Purpose

VioModeWait allows a graphics mode application to be notified when it must restore its video mode, state, and modified display adapter registers. The return from this function call provides the notification.

Calling Sequence

```
EXTRN  VioModeWait:FAR

PUSH   WORD    RequestType    ;Request type
PUSH@  WORD    NotifyType     ;Notify type (returned)
PUSH   WORD    Reserved       ;Reserved (must be 0)
CALL   VioModeWait
```

Where

RequestType
indicates the event the application is waiting for. RequestType = 0 indicates the application wants to be notified at the end of a pop-up to restore its mode. RequestType = 0 is the only event supported by VioModeWait.

NotifyType
specifies the operation to be performed by the application upon return from VioModeWait. NotifyType = 0, indicating restore mode, is the only type of notification returned.

Reserved
is a reserved word of 0s.

Returns

IF AX = 0 then NO error

ELSE AX = error code

VioModeWait — Restore Mode Wait

Remarks

A VioModeWait thread is notified to perform a restore at the completion of an application or hard error pop-up. Refer to "VioPopUp — Allocate a pop-up Display Screen" on page 5-28 for further discussion. The VioModeWait thread of the session that was originally interrupted for the pop-up is notified. The VioModeWait thread must restore the video mode, state, and modified display adapter registers and immediately re-issue VioModeWait. The VioModeWait thread does not restore the physical display buffer. OS/2 saves/restores the physical display buffer over a pop-up.

Only one process for a session can issue VioModeWait. The first process that issues VioModeWait becomes the owner of this function. (Refer to "VioModeUndo — Restore Mode Undo" on page 5-23.)

An application must issue VioModeWait only if it writes directly to the registers on the display adapter. Otherwise, if VioModeWait is not issued, OS/2 restores the physical display buffer, mode, and state at the completion of a pop-up.

A graphics mode application (or a text mode application which writes directly to the registers on the display adapter) must issue VioSavRedrawWait. (Refer to "VioSavRedrawWait — Screen Save Redraw Wait" on page 5-45.)

A VioModeWait thread should not issue any file system or loader DOS calls (or calls to any dynamic link routines which issue these file system or loader DOS calls). Otherwise, a system lockout will occur in the following scenario:

1. One of the threads of a process running in the foreground causes a hard error.
2. A hard error pop-up is displayed. At the completion of the pop-up, a VioModeWait thread (in the same process which caused the hard error) is notified to perform a restore.
3. The VioModeWait thread issues a file system or loader DOS call.

An application that contains a VioModeWait thread should be designed to avoid any hard errors while the VioModeWait thread is running. If hard errors occur, there is a potential for a system lockout to occur.

VioPopUp — Allocate a pop-up Display Screen

Purpose

VioPopUp is issued by an application process when it requires a temporary screen to display a momentary message to the user.

Calling Sequence

```
EXTRN   VioPopUp:FAR

PUSH@  WORD    Options       ;Option Flags
PUSH   WORD    VioHandle     ;Vio handle
CALL   VioPopUp
```

Where

Options

contain bit flags that indicate which of the various options available to the application are being selected. The flags bits are described below:

High byte = 0

Low byte=:

Bit	**Meaning**
7-2 -	Reserved = 0
1 -	0=non-transparent operation. The video mode is set to text — mode 3, 3*, 3+, 7 or 7+. The highest resolution supported by the primary display adapter configured in the system is selected. The screen is cleared, and the cursor is positioned in the upper left corner of the display. 1=transparent operation. If the video mode of the outgoing foreground session is text (mode 2, 3, 7 or 1 of the * or + variations of these modes), no mode change occurs. The screen is not cleared, and the cursor remains in its current position. If transparent operation is selected, and if the video mode of the outgoing foreground session is not text (or if the outgoing foreground session has a VioSavRedrawWait thread), the pop-up request is refused. A unique error code is returned in this case.

OS/2 is responsible for saving and restoring the physical display buffer of the previous foreground session around a pop-up. This is true whether transparent or non-transparent operation is selected.

0 - 0=return with unique error code if pop-up is not immediately available and 1=wait if pop-up is not immediately available.

VioHandle
is a reserved word of 0s.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

VioPopUp is normally issued by the application when it is running in the background and wishes to immediately display a message to the user without waiting to become the active foreground session.

When an application process issues VioPopUp, it should wait for the return from the request. If the process allows any of its threads to write to the screen before VioPopUp returns a successful return code, the screen output may be directed to the application's normal video buffer rather than to the pop-up screen. If the process allows any of its threads to issue keyboard or mouse calls before VioPopUp returns a successful return code, the input will be directed from the application's normal session. Once the process which issued VioPopUp receives a successful return code, video and keyboard calls issued by any of the threads in the pop-up process are directed to the pop-up screen. This will continue until the process issues VioEndPopUp phen.up. At that time video and keyboard calls resume being directed to the application's normal video buffer.

There may be only one pop-up in existence at any one time. If a process requests a pop-up and a pop-up already exists, the process has the choice of waiting for the prior pop-up to complete or receiving an immediate return with an error code. The error code will indicate that the operation failed due to an existing pop-up having captured the screen.

Vio pop-ups provide a mechanism for a background application to notify the operator of an abnormal event upon which the operator must take some action. When considering the suitability of using pop-ups in a particular situation, the possible disruptive effect of pop-ups to the operator should be considered. If the operator were interrupted frequently by pop-ups issued by background applications, the operator would not be able to work effectively with the foreground application.

While a video pop-up is in the foreground, the operator cannot use the hot key to switch to another application or the shell. Before the operator can switch another application or the shell to the foreground, the pop-up application must issue VioEndPopUp

While a video pop-up is in effect, all video calls from the previous foreground session are blocked until the process that issued VioPopUp issues VioEndPopUp

When VioPopUp is issued, only the process within the session that issued VioPopUp is brought to the foreground. Assuming the session was already the foreground session, any video calls issued by other processes in that session are blocked until the process that issued VioPopUp issues VioEndPopUp

DosExecPgm may not be issued by a process during a pop-up. The following video calls are the only calls that may be issued by a process that issued VioPopUp during the pop-up:

VioEndPopUp	VioScrollDn
VioGetConfig	VioScrollLf
VioGetCp	VioSetCurPos
VioGetFont	VioSetCurType
VioGetAnsi	VioSetCp
VioGetState	VioSetFont
VioGetCurPos	VioSetState
VioGetCurType	VioWrtNChar
VioGetMode	VioWrtNAttr
VioReadCharStr	VioWrtNCell
VioReadCellStr	VioWrtCharStr
VioScrollRt	VioWrtCharStrAtt
VioScrollUp	VioWrtCellStr
	VioWrtTTY

Selectors to the physical display buffer, which the issuing process obtained on a prior VioGetPhysBuf call, may not be used during the pop-up.

When an application registers a replacement for VioPopUp within a session, the registered routine is only invoked when that session is in the foreground. If VioPopUp is issued when that session is in the background, the OS/2 default routine is invoked. If the application's session is using a keyboard or mouse monitor, the monitor will not intercept data while the pop-up is active.

VioPrtSc – Print Screen

Purpose

VioPrtSc copies the screen to the printer.

Calling Sequence

```
EXTRN  VioPrtSc:FAR

PUSH   WORD    VioHandle     ;Vio handle
CALL   VioPrtSc
```

Where

VioHandle
is a reserved word of 0s.

Returns

IF   AX = 0 then NO error

ELSE AX = error code

Remarks

VioPrtSc supports text modes 0 through 3, 7, and the + and * variations of these modes. An Alternate Video Subsystem may want to register a replacement for VioPrtSc. The Base Video Subsystem does not support PrtSc in graphics modes.

VioPrtSc is reserved for use by the session manager. Application programs may not issue VioPrtSc.

VioPrtScToggle – Toggle Print Screen

Purpose
VioPrtScToggle is called by the Session Manager when the operator presses Ctrl-PrtSc.

Calling Sequence
```
EXTRN  VioPrtScToggle:FAR

PUSH   WORD    VioHandle     ;Vio handle
CALL   VioPrtScToggle
```

Where
VioHandle
is a reserved word of 0s.

Returns
IF AX = 0 then NO error

ELSE AX = error code

Remarks
VioPrtScToggle toggles the Ctrl-PrtSc state of the foreground session. When the Ctrl-PrtSc state is on, all VioWrtTTY calls from that session are echoed to the print device.

VioPrtScToggle can only be called by the session manager. If an application issues this call, it will receive an error code.

Three beeps are generated if a hard error is detected while writing to the printer. When Ctrl-PrtSc is turned off, the operator may have to press the Enter key to cause output spooled while Ctrl-PrtSc was active to be printed.

VioReadCellStr – Read Char/Attr String

Purpose

VioReadCellStr reads a string of character-attribute pairs (or cells) from the screen starting at the specified location.

Calling Sequence

```
EXTRN  VioReadCellStr:FAR

PUSH@  OTHER   CellStr        ;Cell string buffer
PUSH@  WORD    Length         ;Length of cell string buffer
PUSH   WORD    Row            ;Starting row location
PUSH   WORD    Column         ;Starting column location
PUSH   WORD    VioHandle      ;Video handle
CALL   VioReadCellStr
```

Where

CellStr
is the buffer into which the cell string is read.

Length
is the length of the buffer in bytes. Length must take into account that each character-attribute entry in the buffer is 2-bytes. If the length of the buffer is not sufficient, the last entry will not be complete.

Row
is the starting row of the field to read where 0 is the top row.

Column
is the starting column of the field to read where 0 is the leftmost column.

VioHandle
is a reserved word of 0s.

Returns

IF   AX = 0 then NO error

ELSE AX = error code

Remarks

If a string read comes to the end of the line and is not complete, the string read continues at the beginning of the next line. If the read comes to the end of the screen and is not complete, the read terminates and the length is set to the length of the buffer that was filled.

VioReadCharStr – Read Character String

Purpose

VioReadCharStr reads a character string from the display starting at the specified location.

Calling Sequence

```
EXTRN   VioReadCharStr:FAR

PUSH@   OTHER    CharStr        ;Character buffer
PUSH@   WORD     Length         ;Length of buffer
PUSH    WORD     Row            ;Starting row location
PUSH    WORD     Column         ;Starting column location
PUSH    WORD     VioHandle      ;Video handle
CALL    VioReadCharStr
```

Where

CharStr
is the buffer where the character string is read into.

Length
is the length of the buffer in bytes.

Row
is the starting row of the field to read where 0 is the top row.

Column
is the starting column of the field to read where 0 is the leftmost column.

VioHandle
is a reserved word of 0s.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

If a string read comes to the end of the line and is not complete, then the string read continues at the beginning of the next line. If the read comes to the end of the screen and is not complete, the read terminates and the length is set to the number of characters read.

VioRegister – Register Video Subsystem

Purpose

VioRegister registers an Alternate Video Subsystem within a session.

Calling Sequence

```
EXTRN  VioRegister:FAR

PUSH@  ASCIIZ  ModuleName     ;Module name
PUSH@  ASCIIZ  EntryPoint     ;Entry point name
PUSH   DWORD   FunctionMask1  ;Function mask 1
PUSH   DWORD   FunctionMask2  ;Function mask 2
CALL   VioRegister
```

Where

ModuleName
contains the dynamic link module name. The maximum length is 129 bytes including the terminating byte of 0.

EntryPoint
contains the dynamic link entry point name of a routine that receives control when any of the registered functions are called. The maximum length is 33 bytes including the terminating byte of 0.

FunctionMask1
is a bit mask where each bit identifies a video function. The bit definitions are shown below. The first word pushed onto the stack contains the high order 16 bits of the function mask, and the second word contains the low order 16 bits.

Bit	Registered Function
31	VioPrtScToggle
30	VioEndPopUp
29	VioPopUp
28	VioSavRedrawUndo
27	VioSavRedrawWait
26	VioScrUnLock
25	VioScrLock
24	VioPrtSc
23	VioGetAnsi
22	VioSetAnsi
21	VioScrollRt
20	VioScrollLf
19	VioScrollDn
18	VioScrollUp
17	VioWrtCellStr
16	VioWrtCharStrAtt
15	VioWrtCharStr
14	VioWrtTTY
13	VioWrtNCell
12	VioWrtNAttr
11	VioWrtNChar
10	VioReadCellStr
09	VioReadCharStr
08	VioShowBuf
07	VioSetMode
06	VioSetCurType
05	VioSetCurPos
04	VioGetPhysBuf
03	VioGetBuf
02	VioGetMode
01	VioGetCurType
00	VioGetCurPos

FunctionMask2

is a bit mask where each bit identifies a video function. The bit mask has the format shown below. The first word pushed onto the stack contains the high order 16 bits of the function mask, and the second word contains the low order 16 bits. Unused bits are reserved and must be 0.

Bit	Registered Function
31-09	Reserved = 0
08	VioSetState
07	VioGetState
06	VioSetFont
05	VioGetCp
04	VioSetCp
03	VioGetConfig
02	VioGetFont
01	VioModeUndo
00	VioModeWait

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

An Alternate Video Subsystem must register which video calls it handles. The default OS/2 video subsystem is the Base Video Subsystem.

When any of the registered functions are called, control is routed to EntryPoint. When this routine is entered, four additional values (5 words) are pushed onto the stack.

The first value is the index number (WORD) of the routine being called. The second value is a near pointer (WORD). The third value is the caller's DS register (WORD). The fourth value is the return address (DWORD) to the VIO router.

For example, if VioSetCurPos were a registered function, the stack would appear as if the following instruction sequence were executed if VioSetCurPos were called and control routed to EntryPoint:

```
PUSH   WORD   Row
PUSH   WORD   Column
PUSH   WORD   VioHandle
CALL   FAR    VioSetCurPos
PUSH   WORD   Index
CALL   NEAR   Entry point in Vio router
PUSH   WORD   Caller's DS
CALL   FAR    Dynamic link entry point
```

The index numbers that correspond to the registered functions are listed below:

0	VioGetPhysBuf	21	VioScrollRt
1	VioGetBuf	22	VioSetAnsi
2	VioShowBuf	23	VioGetAnsi
3	VioGetCurPos	24	VioPrtSc
4	VioGetCurType	25	VioScrLock
5	VioGetMode	26	VioScrUnLock
6	VioSetCurPos	27	VioSavRedrawWait
7	VioSetCurType	28	VioSavRedrawUndo
8	VioSetMode	29	VioPopUp
9	VioReadCharStr	30	VioEndPopUp
10	VioReadCellStr	31	VioPrtScToggle
11	VioWrtNChar	32	VioModeWait
12	VioWrtNAttr	33	VioModeUndo
13	VioWrtNCell	34	VioGetFont
14	VioWrtCharStr	35	VioGetConfig
15	VioWrtCharStrAtt	36	VioSetCp
16	VioWrtCellStr	37	VioGetCp
17	VioWrtTTY	38	VioSetFont
18	VioScrollUp	39	VioGetState
19	VioScrollDn	40	VioSetState
20	VioScrollLf		

When a registered function returns to the video router, the contents of AX are interpreted as follows:

AX = 0
: No error. Do not invoke the corresponding Base Video Subsystem routine. Return to caller with AX = 0.

VioRegister — Register Video Subsystem

AX = -1
: No error. Invoke the corresponding Base Video Subsystem routine. Return to caller with AX = return code from Base Video Subsystem.

AX = error (not 0 or -1)
: Do not invoke the corresponding Base Video Subsystem routine. Return to caller with AX = error.

When an application registers a replacement for VioPopUp within a session, the registered routine is only invoked when that session is in the foreground. If VioPopUp is issued when that session is in the background, the OS/2 default routine is invoked.

An Alternate Video Subsystem should be designed so that the routines registered do not cause any hard errors when they are invoked. Otherwise, a system lockout will occur. Code and data segments of registered routines, which might potentially be loaded from diskette, must be preloaded.

Purpose

VioSavRedrawUndo allows one thread within a process to cancel a VioSavRedrawWait issued by another thread within the same process.

Calling Sequence

```
EXTRN  VioSavRedrawUndo:FAR

PUSH   WORD    OwnerIndic     ;Ownership indicator
PUSH   WORD    KillIndic      ;Terminate indicator
PUSH   WORD    VioHandle      ;Video handle
CALL   VioSavRedrawUndo
```

Where

OwnerIndic

indicates whether the thread issuing VioSavRedrawUndo wants ownership of VioSavRedrawWait to be reserved for its process.

- If OwnerIndic = 0, reserve ownership
- If OwnerIndic = 1, give up ownership

KillIndic

indicates whether the thread with the outstanding VioSavRedrawWait should be returned an error code or be terminated.

- If KillIndic = 0, return error code
- If KillIndic = 1, terminate thread

VioHandle

is a reserved word of 0s.

Returns

IF AX = 0 then NO error

ELSE AX = error code

VioSavRedrawUndo — Screen Save Redraw Undo

Remarks

The issuing thread can reserve ownership of VioSavRedrawWait for its process or give it up. If a thread's VioSavRedrawWait is cancelled, it is optionally terminated. VioSavRedrawUndo may be issued only by a thread within the same process that owns VioSavRedrawWait.

VioSavRedrawWait – Screen Save Redraw Wait

Purpose

VioSavRedrawWait notifies a graphics mode application when it must save or redraw its screen image. The return from this function call provides the notification. The thread that issues the call performs the save or redraw and then re-issues VioSavRedrawWait to wait until its screen image must be saved or redrawn again.

Calling Sequence

```
EXTRN  VioSavRedrawWait:FAR

PUSH   WORD  SavRedrawIndic  ;Save/redraw indicator
PUSH@  WORD  NotifyType      ;Notify type (returned)
PUSH   WORD  VioHandle       ;Video handle
CALL   VioSavRedrawWait
```

Where

SavRedrawIndic
indicates which events the application is waiting for:

If SavRedrawIndic = 0
the session manager notifies the application for both save and redraw operations.

If SavRedrawIndic = 1
the session manager notifies the application for redraw operations only.

NotifyType
specifies the operation to be performed by the application upon return from VioSavRedrawWait:

0 = save screen image
1 = restore screen image

VioHandle
is a reserved word of 0s.

VioSavRedrawWait — Screen Save Redraw Wait

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

OS/2 uses VioSavRedrawWait to notify a graphics mode application to save or restore its screen image at screen switch time. The application in the outgoing foreground session is notified to perform a save. The application in the incoming foreground session is notified to perform a restore. The application must perform the action requested and immediately re-issue VioSavRedrawWait. When an application performs a save, it saves its physical display buffer, video mode, and any other information the application needs to completely redraw its screen at restore time.

Only one process per session can issue VioSavRedrawWait. The process that issues VioSavRedrawWait first, becomes the owner of the function.

A text mode application must issue VioSavRedrawWait only if the application writes directly to the registers on the display adapter. Assuming VioSavRedrawWait is not issued by a text mode application, OS/2 performs the required saves and restores.

An application that issues VioSavRedrawWait may also need to issue VioModeWait. This would allow the application to be notified when it must restore its mode at the completion of an application or hard error popup. Refer to “VioModeWait — Restore Mode Wait” on page 5-25 for more information. Two application threads would be required to perform these operations in this case.

At the time a VioSavRedrawWait thread is notified, the session is in transition to/from the background. Although the session's official status is background, any selector to the physical display buffer previously obtained by the VioSavRedrawWait process (through VioGetPhysBuf) is valid at this time. The physical display buffer must be accessed without issuing VioScrLock. Since the session's official status is background, any thread which issues VioScrLock with the "wait if unsuccessful" option will in fact wait.

An application containing a VioSavRedrawWait thread should be designed so that the process does not cause any hard errors while the VioSavRedrawWait thread is running. Otherwise, there is a potential for a system lockout situation to occur.

An application's VioSavRedrawWait thread may be notified to perform a restore before it is notified to perform a save. This would happen if the application was running in the background the first time it issued VioSavRedrawWait.

Note: that the OS/2 Start command starts an application in the background.

VioScrLock — Lock Screen

Purpose

VioScrLock requests ownership of (locks) the physical display buffer.

Calling Sequence

```
EXTRN  VioScrLock:FAR

PUSH   WORD    WaitFlag      ;Block or not
PUSH@  BYTE    Status        ;Lock status (returned)
PUSH   WORD    VioHandle     ;Video handle
CALL   VioScrLock
```

Where

WaitFlag

indicates whether the process should block until the screen I/O can take place.

- 0 = return if screen I/O not available
- 1 = wait until screen I/O is available

Status

indicates whether the lock is successful.

- 0 = lock successful
- 1 = lock unsuccessful (in the case of no wait)
- Status is returned only when AX = 0.
- Status = 1 may be returned only when WaitFlag = 0.

VioHandle

is a reserved word of 0s.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Family API Considerations

Some options operate differently in the DOS mode than they do in the OS/2 mode. Therefore, the following restriction applies to VioScrLock when coding in the DOS mode :

The status will always indicate the lock is successful (AX = 0)

Remarks

This function call permits a process to determine if I/O to the physical screen buffer can take place. This prevents the process from writing to the physical buffer when the process is in the background. Processes must cooperate with the system in coordinating screen accesses.

Screen switching is disabled while the screen lock is in place. If a screen switch is suspended by a screen lock, and if the application holding the lock does not issue VioScrUnLock within a system-defined time limit, the screen switch occurs, and the process holding the lock is frozen in the background. A process should yield the screen lock as soon as possible to avoid being frozen when running in the background. The timeout on the lock does not begin until a screen switch is requested.

When the screen lock is in effect and another thread in the same or different process (in the same session) issues VioScrLock, the second thread receives an error code. VioScrUnLock must be issued by a thread within the same process that issued VioScrLock.

VioScrollDn – Scroll Screen Down

Purpose

VioScrollDn scrolls the entire display buffer (or area specified within the display buffer) down.

Calling Sequence

```
EXTRN  VioScrollDn:FAR

PUSH   WORD    TopRow         ;Top row
PUSH   WORD    LeftCol        ;Left column
PUSH   WORD    BotRow         ;Bottom row
PUSH   WORD    RightCol       ;Right column
PUSH   WORD    Lines          ;Number of lines
PUSH@  OTHER   Cell           ;Cell to be written
PUSH   WORD    VioHandle      ;Video handle
CALL   VioScrollDn
```

Where

TopRow
is the top row of the area to scroll.

LeftCol
is the left column of the area to scroll.

BotRow
is the bottom row of the area to scroll.

RightCol
is the right column of the area to scroll.

Lines
is the number of lines to be inserted at the top of the screen area being scrolled. If 0 is specified, no lines are scrolled.

Cell
is a character-attribute pair (two bytes) used as a fill character on inserted lines.

VioHandle
is a reserved word of 0s.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

TopRow = 0 and LeftCol = 0 identifies the top left corner of the screen.

If a value greater than the maximum value is specified for TopRow, LeftCol, BotRow, RightCol, or Lines, the maximum value for that parameter is used.

If TopRow and LeftCol = 0 and if BotRow, RightCol, and Lines = 65535 (or -1 in Assembler language), the entire screen will be filled with the character defined by Cell.

VioScrollLf – Scroll Screen Left

Purpose

VioScrollLf scrolls the entire display buffer (or area specified within the display buffer) left.

Calling Sequence

```
EXTRN  VioScrollLf:FAR

PUSH   WORD    TopRow        ;Top row
PUSH   WORD    LeftCol       ;Left column
PUSH   WORD    BotRow        ;Bottom row
PUSH   WORD    RightCol      ;Right column
PUSH   WORD    Lines         ;Number of lines
PUSH@  OTHER   Cell          ;Cell to be written
PUSH   WORD    VioHandle     ;Video Handle
CALL   VioScrollLf
```

Where

TopRow
is the top row of the area to scroll.

LeftCol
is the left column of the area to scroll.

BotRow
is the bottom row of the area to scroll.

RightCol
is the right column of the area to scroll.

Lines
is the number of columns to be inserted at the right of the screen area being scrolled. If 0 is specified, no lines are scrolled.

Cell
is a character attribute pair (two bytes) used as a fill character on inserted columns.

VioHandle
is a reserved word of 0s.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

TopRow = 0 and LeftCol = 0 identifies the top left corner of the screen.

If a value greater than the maximum value is specified for TopRow, LeftCol, BotRow, RightCol, or Lines, the maximum value for that parameter is used.

If TopRow and LeftCol = 0 and if BotRow, RightCol, and Lines = 65535 (or -1 in Assembler language), the entire screen will be filled with the character defined by Cell.

VioScrollRt – Scroll Screen Right

Purpose

VioScrollRt scrolls the entire display buffer (or area specified within the display buffer) right.

Calling Sequence

```
EXTRN  VioScrollRt:FAR

PUSH   WORD    TopRow        ;Top row
PUSH   WORD    LeftCol       ;Left column
PUSH   WORD    BotRow        ;Bottom row
PUSH   WORD    RightCol      ;Right column
PUSH   WORD    Lines         ;Number of lines
PUSH@  OTHER   Cell          ;Cell to be written
PUSH   WORD    VioHandle     ;Video handle
CALL   VioScrollRt
```

Where

TopRow
is the top row of the area to scroll.

LeftCol
is the left column of the area to scroll.

BotRow
is the bottom row of the area to scroll.

RightCol
is the right column of the area to scroll.

Lines
is the number of columns to be inserted at the left of the screen area being scrolled. If 0 is specified, no lines are scrolled.

Cell
is a character attribute pair (two bytes) used as a fill character on inserted columns.

VioHandle
is a reserved word of 0s.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

TopRow = 0 and LeftCol = 0 identifies the top left corner of the screen.

If a value greater than the maximum value is specified for TopRow, LeftCol, BotRow, RightCol, or Lines, the maximum value for that parameter is used.

If TopRow and LeftCol = 0 and if BotRow, RightCol, and Lines = 65535 (or -1 in Assembler language), the entire screen will be filled with the character defined by Cell.

VioScrollUp – Scroll Screen Up

Purpose

VioScrollUp scrolls the entire display buffer (or area specified within the display buffer) up.

Calling Sequence

```
EXTRN  VioScrollUp:FAR

PUSH   WORD     TopRow         ;Top row
PUSH   WORD     LeftCol        ;Left column
PUSH   WORD     BotRow         ;Bottom row
PUSH   WORD     RightCol       ;Right column
PUSH   WORD     Lines          ;Number of lines
PUSH@  OTHER    Cell           ;Fill character
PUSH   WORD     VioHandle      ;Video handle
CALL   VioScrollUp
```

Where

TopRow
is the top row of the area to scroll.

LeftCol
is the left column of the area to scroll.

BotRow
is the bottom row of the area to scroll.

RightCol
is the right column of the area to scroll.

Lines
is the number of lines to be inserted at the bottom of the screen area being scrolled. If 0 is specified, no lines are scrolled.

Cell
is a character attribute pair (two bytes) used as a fill character on inserted lines.

VioHandle
is a reserved word of 0s.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

TopRow=0 and LeftCol=0 identifies the top left corner of the screen.

If a value greater than the maximum value is specified for TopRow, LeftCol, BotRow, RightCol, or Lines, the maximum value for that parameter is used.

If TopRow and LeftCol=0 and if BotRow, RightCol, and Lines=65535 (or -1 in Assembler language), the entire screen will be filled with the character defined by Cell.

VioScrUnLock – Unlock Screen

Purpose

VioScrUnLock releases ownership of (unlocks) the physical display buffer.

Calling Sequence

```
EXTRN  VioScrUnLock:FAR

PUSH   WORD    VioHandle     ;Video handle
CALL   VioScrUnLock
```

Where

VioHandle
is a reserved word of 0s.

Returns

IF   AX = 0 then NO error

ELSE AX = error code

Family API Considerations

Some options operate differently in the DOS mode than they do in the OS/2 mode. Therefore, the following restriction applies to VioScrUnLock when coding in the DOS mode :

The status will always indicate the unlock is successful (AX = 0).

Remarks

None.

Purpose

VioSetAnsi activates or deactivates ANSI support.

Calling Sequence

```
EXTRN  VioSetAnsi:FAR

PUSH   WORD    Indicator      ;On/Off indicator
PUSH   WORD    VioHandle      ;Video handle
CALL   VioSetAnsi
```

Where

Indicator
equals 1 to activate ANSI support or 0 to deactivate ANSI.

VioHandle
is a reserved word of 0s.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

For ANSI support, "ON" is the default.

VioSetCp – Set Code Page

Purpose

VioSetCp allows a process to set the code page used to display text data on the screen.

Calling Sequence

```
EXTRN   VioSetCp:FAR

PUSH   WORD    Reserved       ;
PUSH   WORD    CodePageID     ;CodePage Id
PUSH   WORD    VioHandle      ;Video handle
CALL   VioSetCp
```

Where

Reserved
is a reserved word of 0s.

CodePageID
must be equivalent to one of the code page ID's specified on the CONFIG.SYS CODEPAGE = statement or must specify the default ROM code page (0000).

If the code page ID does not match one of the ID's on the CODEPAGE = statement, an error will result. Refer to IBM Operating System/2 User's Reference for a complete description of CODEPAGE.

VioHandle
is a reserved word of 0s.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

The code page tag specified must be either 0000 or have been specified on the CONFIG.SYS CODEPAGE = statement. A value of 0000 indicates that the code page is to be set to the ROM code page provided by the hardware.

VioSetCurPos – Set Cursor Position

Purpose
VioSetCurPos sets the cursor position.

Calling Sequence

```
EXTRN  VioSetCurPos:FAR

PUSH   WORD    Row            ;Row data
PUSH   WORD    Column         ;Column data
PUSH   WORD    VioHandle      ;Video handle
CALL   VioSetCurPos
```

Where

Row
is the new cursor row position where 0 is the top row.

Column
is the new cursor column position where 0 is the left column.

VioHandle
is a reserved word of 0s.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks
None

Purpose

VioSetCurType sets the cursor type.

Calling Sequence

```
EXTRN  VioSetCurType:FAR

PUSH@  OTHER   CursorData    ;Cursor characteristics
PUSH   WORD    VioHandle     ;Video handle
CALL   VioSetCurType
```

Where

CursorData
is a structure that contains the characteristics of the cursor.

Size	*Description*
WORD	Cursor start line
WORD	Cursor end line
WORD	Cursor width
WORD	Cursor attribute

CursorStartLine
is the horizontal scan line in the character cell which marks the top line of cursor. Note that if the character cell has N scan lines, 0 is the top scan line of the character cell and N-1 is the bottom scan line.

CursorEndLine
is the horizontal scan line in the character cell which marks the bottom line of the cursor. Scan lines within a character cell are numbered as defined under CursorStartLine. The maximum value which can be specified for CursorEndLine is 31. The appearance of the cursor when the number of pel rows defined in the cursor is greater than the number of pel rows in a character cell is variable depending upon the display adapter included in the configuration.

CursorWidth
is the width of the cursor in columns. The maximum value supported by the OS/2 Base Video Subsystem is 1. CursorWidth = 0 specifies the default width (one column).

VioSetCurType — Set Cursor Type

CursorAttrib
is the attribute of the cursor.

Value = -1, is hidden. Any other value is normal.

VioHandle
is a reserved word of 0s.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

None

Purpose

VioSetFont downloads a display font. The font being set must be compatible with the current mode.

Calling Sequence

```
EXTRN VioSetFont:FAR

PUSH@  OTHER   RequestBlock  ;Request block
PUSH   WORD    VioHandle     ;Video handle
CALL   VioSetFont
```

Where

RequestBlock

is a data structure that contains a request. The request type is contained in the second word. The format of the request block is shown below. The symbol in the right column has the following meaning:

I - input parameter

VioSetFont – Set Font

	Set Current Font (EGA, VGA, or IBM Personal System/2™ Display Adapter)	
WORD	Length of structure (in bytes including length itself) = 14	I
WORD	Request type = 0, set current font	I
WORD	Pel columns in character cell	I
WORD	Pel rows in character cell	I
DWORD	Far address of a data area that contains the font table to set.	I
WORD	Length in bytes of the data area that contains the font table to set.	I

VioHandle
is a reserved word of 0s.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

When VioSetFont is issued, the current code page is reset. If VioGetCp is subsequently issued, a unique error code is returned in AX. VioSetFont is applicable only for the enhanced graphics adapter, VGA or IBM Personal System/2 Display Adapter.

Note: Return code, ERROR_VIO_USER_FONT represents a warning. It indicates that although the font could not be loaded into the adapter using the current mode, the font was saved for use with a later VioSetMode.

Purpose

VioSetMode sets the mode of the display.

Calling Sequence

```
EXTRN  VioSetMode:FAR

PUSH@  OTHER   ModeData      ;Mode characteristics
PUSH   WORD    VioHandle     ;Video handle
CALL   VioSetMode
```

Where

ModeData

is a structure that contains the characteristics of the mode being set.

Size	**Description**
WORD	Length
BYTE	Type
BYTE	Color
WORD	Text Columns
WORD	Text Rows
WORD	Horizontal Resolution
WORD	Vertical Resolution

Length

is an input parameter to VioSetMode. Length specifies the length of the data structure in bytes including Length itself. The minimum structure size required is three bytes, and the maximum structure size required is 12 bytes. Any value specified for Length other than 3 must be an even number. If a structure of length less than the maximum is specified, OS/2 will use default values for the remaining fields.

Type

is a bit mask that contain specifications for the mode being set. The definitions of the bits follow:

```
xxxxxxxb b = 0 monochrome compatible mode
         b = 1 other

xxxxxxbx b = 0 text mode
         b = 1 graphics mode
xxxxxbxx b = 0 enable color burst
         b = 1 disable color burst
```

Color

defines the number of colors as a power of 2. This is equivalent to the number of color bits which define the color. For example,

```
Color = 1 specifies 2 colors
Color = 2 specifies 4 colors
Color = 4 specifies 16 colors
Color = 8 specifies 256 colors

Color = 0 should be specified for
                monochrome modes 7, 7+, and F.
```

Text Columns

are the number of text columns.

Text Rows

are the number of text rows. are supported for The color graphics adapter supports 25 rows. The enhanced graphics adapter supports 25 and 43 rows. The VGA adapter and the IBM Personal System/2™ Display Adapter support 25 and 50 rows.

Horizontal Resolution

is the number of pel columns.

Vertical Resolution

is the number of pel rows.

VioHandle

is a reserved word of 0s.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

VioSetMode initializes the cursor position and type. VioSetMode will clear the screen in the DOS mode and in DOS 3.3. For all other environments, to clear the screen, use one of the VioScrollxx calls.

The disable color burst bit in the Type field in the VioSetMode data structure is functional only for the color graphics adapter. For all other display adapters the setting of this bit is returned on any subsequent VioGetMode call but is otherwise ignored.

EXAMPLES

```
Mode 2
Type                  = 00000101
Color                 = 4
Text Columns          = 80
Text Rows             = 25
Horizontal Resolution = 640
Vertical Resolution   = 200

Mode 3

Type                  = 00000001
Color                 = 4
Text Columns          = 80
Text Rows             = 25
Horizontal Resolution = 640
Vertical Resolution   = 200

Mode 3*

Type                  = 00000001
Color                 = 4
Text Columns          = 80
Text Rows             = 25
Horizontal Resolution = 640
Vertical Resolution   = 350

Mode 3+

Type                  = 00000001
Color                 = 4
Text Columns          = 80
Text Rows             = 25
Horizontal Resolution = 720
Vertical Resolution   = 400

Mode 5
Type                  = 00000111
Color                 = 2
Text Columns          = 40
Text Rows             = 25
Horizontal Resolution = 320
Vertical Resolution   = 200
```

```
Mode 6
Type                  = 00000011
Color                 = 1
Text Columns          = 80
Text Rows             = 25
Horizontal Resolution = 640
Vertical Resolution   = 200

Mode 7
Type                  = 00000000
Color                 = 0
Text Columns          = 80
Text Rows             = 25
Horizontal Resolution = 720
Vertical Resolution   = 350

Mode 7+

Type                  = 00000000
Color                 = 0
Text Columns          = 80
Text Rows             = 25
Horizontal Resolution = 720
Vertical Resolution   = 400
Mode E

Type                  = 00000011
Color                 = 4
Text Columns          = 80
Text Rows             = 25
Horizontal Resolution = 640
Vertical Resolution   = 200

Mode F

Type                  = 00000010
Color                 = 0
Text Columns          = 80
Text Rows             = 25
Horizontal Resolution = 640
Vertical Resolution   = 350
```

VioSetMode – Set Display Mode

```
Mode 10

Type                  = 00000011
Color                 = 4
Text Columns          = 80
Text Rows             = 25
Horizontal Resolution = 640
Vertical Resolution   = 350

Mode 11

Type                  = 00000011
Color                 = 1
Text Columns          = 80
Text Rows             = 30
Horizontal Resolution = 640
Vertical Resolution   = 480

Mode 12

Type                  = 00000011
Color                 = 4
Text Columns          = 80
Text Rows             = 30
Horizontal Resolution = 640
Vertical Resolution   = 480

Mode 13

Type                  = 00000011
Color                 = 8
Text Columns          = 40
Text Rows             = 25
Horizontal Resolution = 320
Vertical Resolution   = 200
```

Purpose

VioSetState performs one of the following functions; sets palette registers, sets the overscan (border) color or sets the blink/background intensity switch.

Calling Sequence

```
EXTRN VioSetState:FAR

PUSH@  OTHER   RequestBlock  ;Request block
PUSH   WORD    VioHandle     ;Video handle
CALL   VioSetState
```

Where

RequestBlock

is a data structure that contains the request. The content of the structure varies depending on the request type. The request type is contained in the second word. The formats of the supported request blocks are shown below. The symbol in the right column has the following meaning:

- I - input parameter

	Set Palette Registers (EGA, VGA, or IBM Personal System/2 Display Adapter)	
WORD	Length of structure (in bytes including length itself) maximum length = 38.	I
WORD	Request type = 0, set palette registers	I
WORD	First palette register to set. Must be in range 0 to 15. The palette registers are set in sequential order. The number of palette registers set is based upon the length of the structure.	I
1 or more WORDs	One WORD that contains the color value for each palette register being set.	I

	Set Overscan (Border) Color (CGA, VGA, or IBM Personal System/2 Display Adapter)	
WORD	Length of structure (in bytes including length itself) = 6	I
WORD	Request type = 1, set overscan (border) color	I
WORD	Color value	I

	Set Blink/Background Intensity Switch (CGA, EGA, VGA, or IBM Personal System/2 Display Adapter)	
WORD	Length of structure (in bytes including length itself) = 6	I
WORD	Request type = 2, set blink/ background intensity switch	I

	Set Blink/Background Intensity Switch (CGA, EGA, VGA, or IBM Personal System/2 Display Adapter)	
WORD	Value = 0, enables blinking foreground colors. Value = 1, enables high intensity background colors	I

VioHandle
is a reserved word of 0s.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

None

VioShowBuf – Display Logical Buffer

Purpose

VioShowBuf updates the physical display buffer with the logical video buffer (LVB).

Calling Sequence

```
EXTRN  VioShowBuf:FAR

PUSH   WORD    Offset        ;Offset into LVB
PUSH   WORD    Length        ;Length
PUSH   WORD    VioHandle     ;Video handle
CALL   VioShowBuf
```

Where

Offset
is the starting offset within the logical video buffer where the screen update begins.

Length
is the length of the area to be updated to the screen.

VioHandle
is a reserved word of 0s.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

VioShowBuf is ignored unless the session is running in the foreground and some process within the session has previously called VioGetBuf.

VioShowBuf is not supported in graphics modes.

Purpose

VioWrtCellStr writes a string of character-attribute pairs (cells) to the display.

Calling Sequence

```
EXTRN  VioWrtCellStr:FAR

PUSH@  OTHER   CellStr        ;String to be written
PUSH   WORD    Length         ;Length of string
PUSH   WORD    Row            ;Starting row position for output
PUSH   WORD    Column         ;Starting column position for output
PUSH   WORD    VioHandle      ;Video handle
CALL   VioWrtCellStr
```

Where

CellStr
is a string of character-attribute cells to be written.

Length
is the length of the string to be written in bytes. Each character-attribute cell is two bytes.

Row
is the starting cursor row to be written into where 0 is the top row.

Column
is the starting cursor column to be written into where 0 is the left column.

VioHandle
is a reserved word of 0s.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

If a string write comes to the end of the line and is not complete, the string write continues at the beginning of the next line. If the write comes to the end of the screen, the write terminates.

Purpose

VioWrtCharStr writes a character string to the display.

Calling Sequence

```
EXTRN  VioWrtCharStr:FAR

PUSH@  OTHER   CharStr        ;String to be written
PUSH   WORD    Length         ;Length of character string
PUSH   WORD    Row            ;Starting row position for output
PUSH   WORD    Column         ;Starting column position for output
PUSH   WORD    VioHandle      ;Video handle
CALL   VioWrtCharStr
```

Where

CharStr
is the character string to be written.

Length
is the length of the character string in bytes.

Row
is the starting cursor row to be written into where 0 is the top row.

Column
is the starting cursor column to be written into where 0 is the left column.

VioHandle
is a reserved word of 0s.

Returns

IF AX = 0 then NO error

ELSE AX = error code

VioWrtCharStr – Write Character String

Remarks

If a string write comes to the end of the line and is not complete, the string write continues at the beginning of the next line. If the write comes to the end of the screen, the write terminates.

Note: The string is written to the display without changing any attributes.

Purpose

VioWrtCharStrAtt writes a character string with repeated attribute to the display.

Calling Sequence

```
EXTRN  VioWrtCharStrAtt:FAR

PUSH@  OTHER   CharStr      ;String to be written
PUSH   WORD    Length       ;Length of string
PUSH   WORD    Row          ;Starting row position for output
PUSH   WORD    Column       ;Starting column position for output
PUSH@  OTHER   Attr         ;Attribute to be replicated
PUSH   WORD    VioHandle    ;Video handle
CALL   VioWrtCharStrAtt
```

Where

CharStr
is the character string to be written.

Length
is the length of the character string in bytes.

Row
is the starting cursor row to be written into where 0 is the top row.

Column
is the starting cursor column to be written into where 0 is the left column.

Attr
is the attribute to be used in the display buffer for each character of the string written.

VioHandle
is a reserved word of 0s.

VioWrtCharStrAtt – Write Char String with Attr

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

If a string write comes to the end of the line and is not complete, the string write continues at the beginning of the next line. If the write comes to the end of the screen, the write terminates.

Purpose

VioWrtNAttr writes an attribute to the display a specified number of times.

Calling Sequence

```
EXTRN   VioWrtNAttr:FAR

PUSH@   OTHER    Attr           ;Attribute to be written
PUSH    WORD     Times          ;Repeat count
PUSH    WORD     Row            ;Starting row position for output
PUSH    WORD     Column         ;Starting column position for output
PUSH    WORD     VioHandle      ;Video handle
CALL    VioWrtNAttr
```

Where

Attr
is the attribute to be written.

Times
is the number of times to write the attribute.

Row
is the starting cursor row to be written into where 0 is the top row.

Column
is the starting cursor column to be written into where 0 is the left column.

VioHandle
is a reserved word of 0s.

Returns

IF AX = 0 then NO error

ELSE AX = error code

VioWrtNAttr – Write N Attributes

Remarks

If a repeated write comes to the end of the line and is not complete, the write continues at the beginning of the next line. If the write comes to the end of the screen, the write terminates.

Purpose

VioWrtNCell writes a cell (or character-attribute pair) to the display a specified number of times.

Calling Sequence

```
EXTRN  VioWrtNCell:FAR

PUSH@  OTHER   Cell           ;Cell to be written
PUSH   WORD    Times          ;Repeat count
PUSH   WORD    Row            ;Starting row position for output
PUSH   WORD    Column         ;Starting column position for output
PUSH   WORD    VioHandle      ;Video handle
CALL   VioWrtNCell
```

Where

CellStr
is the character-attribute cell (two bytes) to be written.

Times
is the number of times to write the cell.

Row
is the starting cursor row to be written into where 0 is the top row.

Column
is the starting cursor column position to be written, where 0 is the left column.

VioHandle
is a reserved word of 0s.

Returns

IF   AX = 0 then NO error

ELSE AX = error code

VioWrtNCell – Write N Char/Attrs

Remarks

If a repeated write comes to the end of the line and is not complete, the write continues at the beginning of the next line. If the write comes to the end of the screen, the write terminates.

Purpose

VioWrtNChar writes a character to the display a specified number of times.

Calling Sequence

```
EXTRN   VioWrtNChar:FAR

PUSH@   OTHER   Char            ;Character to be written
PUSH    WORD    Times           ;Repeat count
PUSH    WORD    Row             ;Starting row position for output
PUSH    WORD    Column          ;Starting column position for output
PUSH    WORD    VioHandle       ;Video handle
CALL    VioWrtNChar
```

Where

Char
is the character to be written.

Times
is the number of times to write the character.

Row
is the starting cursor row to be written into where 0 is the top row.

Column
is the starting cursor column to be written into where 0 is the left column.

VioHandle
is a reserved word of 0s.

Returns

IF AX = 0 then NO error

ELSE AX = error code

VioWrtNChar – Write N Characters

Remarks

If a repeated write comes to the end of the line and is not complete, the write continues at the beginning of the next line. If the write comes to the end of the screen, the write terminates.

Purpose

VioWrtTTY writes a character string to the display starting at the current cursor position. At the completion of the write, the cursor is positioned at the first position beyond the end of the string.

Calling Sequence

```
EXTRN  VioWrtTTY:FAR

PUSH@  OTHER   CharStr       ;String to be written
PUSH   WORD    Length        ;Length of string
PUSH   WORD    VioHandle     ;Video handle
CALL   VioWrtTTY
```

Where

CharString
is the string to be written.

Length
is the length of the character string in bytes.

VioHandle
is a reserved word of 0s.

Returns

IF AX = 0 then NO error

ELSE AX = error code

Remarks

If a string write comes to the end of the line and is not complete, the string write continues at the beginning of the next line. If the write comes to the end of the screen, the screen is scrolled, and the write continues until completed.

The characters carriage return, line feed, backspace, tab and bell are treated as commands rather than printable characters. Backspace is a non-destructive backspace. Tabs are expanded to provide standard

8-byte-wide fields. VioWrtTTY is the only video call affected by Ctrl-PrtSc and ANSI.

Characters are written using the current attribute defined by ANSI or the default value of X'07'.

Chapter 6. Generic IOCtl Commands

OS/2 device drivers are used by OS/2 to access the I/O hardware. The IOCtl functions provide a method for an application, or subsystem, to send device-specific control commands to a device driver. The IOCtl functions are issued through the DosDevIOCtl API function request. The IOCtl functions are subfunctions of the DosDevIOCtl API function request. Applications should use the DosDevIOCtl function request for OS/2 Applications and the INT 21H IOCtl request for DOS applications. See "DosDevIOCtl — I/O Control for Devices" on page 2-34 for additional information.

The category and function fields are determined as follows. Each code is contained in a byte.

Category Code

Category	*Code*
0...	OS/2 Defined
1...	User Defined
.xxx xxxx	Code

Function Code

Function	*Code*
0...	Return error if unsupported
1...	Ignore if unsupported
.0..	Intercepted by OS/2
.1..	Passed to driver
..0.	Sends data and commands to device
..1.	Queries data and information from device
...x xxxx	Subfunction

Note that the sends/queries data bit is intended only to regularize the function set. It plays no critical role; some functions may contain both command and query elements. The convention is that such commands are defined as sends data.

Generic IOCtl Example

Following is the calling sequence for the DosDevIOCtl call:

```
EXTRN   DosDevIOCtl:Far

PUSH@   OTHER  Data        ;Data Packet
PUSH@   OTHER  ParmList    ;Parameter Packet
PUSH    WORD   Function    ;Function Code
PUSH    WORD   Category    ;Category Code
PUSH    WORD   DevHandle   ;User's Device Driver File Handle

CALL    DosDevIOCtl
```

The DosDevIOCtl call sends the request to the device driver request packet. The device driver receives the request packet, and looks for the Command Code (Command 16 is the Generic IOCtl command) to identify the request.

Note that each device driver can define the structure of the Data Packet and the Parameter Packet but all device drivers use the same request header. Refer to "DosDevIOCtl — I/O Control for Devices" on page 2-34 for more information.

The list of categories and functions for the GENERIC IOCtl request are summarized below.:

CAT	**FUNCTION**	**DESCRIPTION**
01		Serial Device Control
	14H	Reserved
	34H	Reserved
	41H	Set baud (bit) rate
	42H	Set line characteristics (stop, parity, data bits)
	44H	Transmit Byte Immediate
	45H	Break off
	46H	Set modem control signals
	47H	Behave as if XOFF received (stop transmit)
	48H	Behave as if XON received (start transmit)
	49H	Reserved
	4BH	Break on

CAT	FUNCTION	DESCRIPTION
	53H	Set Device Control Block (DCB) parameters
	61H	Return current baud (bit) rate
	62H	Return line characteristics
	64H	Return COM status
	65H	Return transmit data status
	66H	Return modem control output signals
	67H	Return current modem input signals
	68H	Return number of chars in receive queue
	69H	Return number of chars in transmit queue
	6DH	Return COM error
	72H	Return COM event information
	73H	Return Device Control Block (DCB) parameters
02		Reserved
03		Pointer Draw Control
	72H	Get pointer draw address (pointer draw DD)
04		Keyboard Control
	50H	Set code page
	51H	Set input mode (default ASCII)
	52H	Set interim character flags
	53H	Set shift state
	54H	Set typamatic rate and delay
	55H	Notify of change of foreground session
	56H	Set session manager Hot Key
	57H	Set KCB
	58H	Set code page ID
	5BH	Reserved
	5CH	Set NLS & custom code page
	71H	Get input mode
	72H	Get interim character flags
	73H	Get shift state
	74H	Read character data record(s)
	75H	Peek character data record
	76H	Get session manager Hot Key
	77H	Get keyboard type
	78H	Get code page ID
	79H	Translate scan code to ASCII
05		Printer Control

CAT	FUNCTION	DESCRIPTION
	42H	Set frame control (CPL, LPI)
	44H	Set infinite retry
	45H	Reserved
	46H	Initialize printer
	48H	Activate Font
	62H	Get frame control
	64H	Get infinite retry
	66H	Get printer status
	69H	Query Active Font
	6AH	Verify Font
06		Light Pen Control
07		Mouse Control
	50H	Allow ptr drawing after screen switch
	51H	Update screen display mode
	52H	Screen switcher call
	53H	Set scaling factors
	54H	Set event mask
	55H	Reserved
	56H	Set pointer shape
	57H	Unmark collision area
	58H	Mark collision area
	59H	Set pointer screen position
	5AH	Set OS/2 mode pointer draw address
	5BH	Set DOS mode pointer draw address
	5CH	Set device status flags
	60H	Get number of buttons
	61H	Get number of mickeys/centimeter
	62H	Get device status flags
	63H	Read event queue
	64H	Get event queue status
	65H	Get event mask
	66H	Get scaling factors
	67H	Get pointer screen position
	68H	Get pointer shape image
	69H	Reserved
08		Logical Disk Control
	00H	Lock drive

CAT	FUNCTION	DESCRIPTION
	01H	Unlock drive
	02H	Redetermine media
	03H	Set logical map
	20H	Block removable
	21H	Get logical map
	22H	Reserved
	43H	Set device parameters
	44H	Write track
	45H	Format and verify track
	5EH	Reserved
	5FH	Reserved
	63H	Get device parameters
	64H	Read track
	65H	Verify track
09		Physical Disk Control
	00H	Lock physical drive
	01H	Unlock physical drive
	44H	Physical write track
	63H	Get physical device parameters
	64H	Physical read track
	65H	Physical verify track
10		Character Device Monitor Control
	40H	Register
11		General Device Control
	01H	Flush input buffer
	02H	Flush output buffer
	60H	Query monitor support
12-127		Reserved Category Codes

ASYNC (RS232-C) Generic IOCtl

Wherever null pointer appears, it is the application's responsibility to set up a null pointer for the appropriate packet pointer before calling the device driver. IOCtls may be interpreted differently by future releases if the pointer is not a null pointer. If a NULL POINTER is called for and a null pointer is not received by the device driver, it is considered an invalid parameter or data packet value in this section.

The application cannot assume a given timing relationship between when the IOCtls are executed and when data is received or transmitted by the ASYNC hardware.

Data Carrier Detect (DCD) is the same signal as Receiver Line Signal Detect (RLSD).

The device driver services each communications port (COM1, COM2, ...) independently. IOCtls issued to the device driver for a given port have absolutely no effect on any other communications ports that the device driver is servicing.

Following is a summary of Category 1 descriptions:

Function	***Description***
14H	reserved
34H	reserved
41H	Set baud (bit) rate
42H	Set line characteristics (stop, parity, data bits)
44H	Transmit Byte Immediate
45H	Break off
46H	Set modem control signals
47H	Behave as if XOFF received (stop transmit)
48H	Behave as if XON received (start transmit)
49H	reserved
4BH	Break on
53H	Set Device Control Block (DCB) parameters
61H	Return current baud (bit) rate
62H	Return line characteristics
64H	Return COM status
65H	Return transmit data status
66H	Return modem control output signals
67H	Return current modem input signals

68H	Return number of chars in receive queue
69H	Return number of chars in transmit queue
6DH	Return COM error
72H	Return COM event information
73H	Return Device Control Block (DCB) parameters

Category 1 – Function 41H

Purpose
Set Baud Rate

Parameter Packet Format

Field	Length
Bit Rate	WORD

Data Packet Format
None. Packet pointer must be NULL.

Where

Bit Rate

The Bit Rate field is a binary integer representing the actual bit rate that the device driver should use to set the bit rate of the port. The valid values are:

110
150
300
600
1200
2400
4800
9600
19200 (AT hardware not rated for this speed)

An OPEN request packet will not cause the device driver to change the bit rate from its previous value. The initial value is 1200 baud.

Returns
If the call is made with invalid Parameter/Data packet values, a general failure error is reported.

Remarks

If a general failure error is not returned, the device driver will perform the action described in Bit Rate.

Category 1 – Function 42H

Purpose

Set Line Characteristics (stop bits, parity, data bits)

Parameter Packet Format

Field	Length
Data Bits	BYTE
Parity	BYTE
Stop Bits	BYTE

Data Packet Format

None. Packet pointer must be NULL.

Where

Data Bit

Value	***Meaning***
00H-04H	reserved
05H	5 data bits
06H	6 data bits
07H	7 data bits (initial value)
08H	8 data bits
09H-FFH	reserved

Parity

Value	***Meaning***
00H	No parity
01H	Odd parity
02H	Even parity (initial value)
03H	Mark parity (parity bit always 1)
04H	Space parity (parity bit always 0)
05H-FFH	reserved

Stop Bits

Value	***Meaning***
00H	1 stop bit (initial value)
01H	1.5 stop bits (valid with 5 bit word length only)
02H	2 stop bits (not valid with 5 bit word length)
03H-FFH	reserved

Returns

If the call is made with invalid Parameter/Data packet values, a general failure error is reported and the line characteristics are not changed for any parameters that were valid.

Remarks

If a general failure error is not returned, the device driver will set the line characteristics as defined.

An OPEN request packet will not cause the device driver to change the line characteristics from its previous values.

If the word length is less than 8 bits then the appropriate high order bits for received data will be 0 when placed in the receive queue by the device driver and when operated on by the device driver (for example, XON/XOFF checking, null stripping). This only applies to data that is received after the command is operated on by the device driver. Data already in the device driver receive queue is not affected in any way by a change in the word length.

If the word length is less than 8 bits the device driver will not automatically truncate control/transmit data that the application may tell the device driver to operate on or use. No error will be reported by the device driver if transmit or control data given to the device driver has high order bits of non-0 value.

For example, if the device driver is told that the word length is 7 bits (high order bit of all data in receive queue from now on is 0) and the XOFF character is 80H then the device driver will never be able to recognize the XOFF character if automatic transmit flow control is enabled. If the error substitution character is set to 80H by the application with a word length of 7 currently being active, the device driver will still place an 80H in the receive queue. It is the responsibility of the application to maintain consistency between the requested word length for the COM device and the requests that the application makes of the device driver.

Purpose
Transmit Byte Immediate

Parameter Packet Format

Field	Length
Character to be Transmitted	BYTE

Data Packet Format
Packet pointer must be NULL.

Returns
If the call is made with an invalid Data Packet value or if there is already another character waiting to be transmitted immediately due to a previous Category 1 Function 44H request that has not been fulfilled then a general failure error is reported and this request is ignored. A transmit immediate request is considered fulfilled when the character is given to the transmit hardware.

Remarks
If a general failure is not returned, the device driver will immediately transmit the byte contained in the Parameter Packet subject to the following conditions:

1. If there is data currently in the transmit queue being transmitted, or waiting to be transmitted, the character to be transmitted immediately will be placed at the logical front of the device driver transmit queue (not considered in transmit queue) so it is the next character to be given to the transmit hardware. If automatic receive flow control is enabled then a XON or XOFF character may or may not be placed ahead of the character to be transmitted immediately.
2. This request always completes immediately (before the character is actually transmitted) even if the character may not be immediately transmitted for reasons discussed below. If there already is one character waiting to transmit immediately due to a pre-

vious request then a general failure error will be returned and the application must make this request again after there is no character waiting to transmit immediately in the device driver transmit queue. Category 1 Function 64H (Return COM status) can be used to determine whether a character is currently waiting to be transmitted immediately.

3. The device driver will not immediately transmit the character waiting to transmit immediately if the device driver is not transmitting characters due to modem control signal output handshaking (see Set Device Control Block (DCB) Category 1 Function 53H Note 3) or if the device driver is currently transmitting a break.
4. If the device driver is not transmitting characters due to automatic transmit or receive flow control (XON/XOFF) being enabled (with the proper set of conditions having happened, see Category 1 Function 53H - Set Device Control Block (DCB)), or due to being asked to behave as if an XOFF character had been received (Category 1 Function 47H) then the device driver will still transmit a character that is waiting to be transmitted immediately due to this request. WARNING: An application which requests the device driver to transmit a character immediately when automatic transmit or receive flow control is enabled may cause unexpected results to happen to the communications line flow control protocol.
5. This is generally used to manually send XON and XOFF characters.
6. The character waiting to be transmitted immediately is not considered part of the device driver transmit queue and is not flushed due to a flush request. XON/XOFF characters that are automatically transmitted due to automatic receive flow control may or may not be placed ahead of the character waiting to be transmitted immediately. Applications should not have dependencies on this ordering.

Purpose
Set Break Off

Parameter Packet Format
None. Packet pointer must be NULL.

Data Packet Format

Field	Length
COM Error Word (COMERR)	WORD

Where

COM Error Word

The device driver returns this information if a general failure error is not reported. See Category 1 Function 6DH, Return COM Error, for COMERR definition. The COM device error information is not cleared by this action.

Returns
If the call is made with an invalid Parameter Packet value then a general failure error is reported, this function is not performed, and valid information is not returned in the Data Packet.

Remarks
If a general failure error is not returned then the device driver will stop generating a break signal. It is not considered an error if the device driver is not generating a break signal. The device driver will then resume transmitting characters taking into account all the other reasons why it may or may not transmit characters.

Category 1 – Function 46H

Purpose

Set Modem Control Signals

Parameter Packet Format

Field	Length
Modem Control Signals ON Mask	BYTE
Modem Control Signals OFF Mask	BYTE

Data Packet Format

Field	Length
COM Error Word (COMERR)	WORD

Where

Modem Control Signals Value

The device driver will set the modem control signals as defined in this field. Bit 0 is DTR and bit 1 is RTS. If any other bits are set/reset by the masks then a general failure error results. The OFF mask contains a mask of the bits to turn off. The OFF mask has bits of 0 for the bits to turn off. The ON mask contains a mask of the bits to turn on. The ON mask has bits of 1 for the bits to turn on. If the Parameter Packet shows to turn off and on the same bit, the bit will be turned on.

For example:

Mask ON	Mask OFF	Meaning
01H	FFH	Set DTR
00H	FEH	Clear DTR
02H	FFH	Set RTS
00H	FDH	Clear RTS
03H	FDH	Set DTR and RTS
00H	FCH	Clear DTR and RTS

If the DTR control mode input handshaking or the RTS control mode input handshaking or toggling on transmit is set then this request is not allowed to try to change the state of the modem control signal(s) that is (are) being used for input handshaking or toggling on transmit. If the request tries to modify a modem control signal that is being used for input handshaking or toggling on transmit then a general failure will result.

COM Error Word

The device driver returns this information if a general failure error is not returned to the application. See Category 1, Function 6DH, Return COM Error, for COMERR definition. The COM device error information is not cleared by this action.

At device driver initialization, the device driver will turn OFF DTR and RTS for the COM devices that it owns.

An OPEN request packet, when the COM device is not already open (from a previous open without a close) (first level open) will cause DTR and RTS to be set according to the DTR Control Mode and the RTS Control Mode. See Note 1 of Set Device Control Block (DCB) (Category 1 Function 53H).

Note: If the port will not be open any more after processing a close request packet (last level close) DTR and RTS will be turned OFF (by the device driver).

After the transmit hardware has completely transmitted (at the physical RS232 interface) all the data that it has been given to transmit by the device driver and at least 10 additional character times have elapsed.

Category 1 – Function 46H

Returns

If the call is made with invalid Parameter Packet values then a general failure error is reported, the modem control signals are not changed, and the data packet information returned to the application is undefined.

Remarks

None

Purpose
Behave as if XOFF Received (stop transmitting)

Parameter Packet Format
None. Packet pointer must be NULL.

Data Packet Format
None. Packet pointer must be NULL.

Returns
If the call is made with invalid Parameter/Data Packet values, a general failure error is reported and this request is not performed by the device driver.

Remarks
If a general failure error is not returned by the device driver, this function causes data transmission to be halted by preventing the device driver from sending additional data to the transmit hardware.

If automatic transmit flow control is enabled then this request causes the device driver to behave exactly as if it received the XOFF character. Transmission can be resumed (due to being stopped from this request) when an XON is received by the device driver, when a Category 1 Function 48H (Behave as if XON received) request is received, or when the device driver is told to disable automatic transmit flow control and the previous state was that automatic transmit flow control was enabled.

If automatic transmit flow control is disabled then a Category 1 Function 48H (Behave as if XON received) request is required for transmission to be resumed (due to being stopped from this request). If after this request is received, the device driver is told to enable automatic transmit flow control then transmission is still disabled but can be re-enabled (due to being stopped from this request) by any of the scenarios discussed in the automatic transmit flow control being enabled scenario.

Note: There still may be other reasons why transmission may be disabled. (See Return COM Status, Category 1 Function 64H.)

Purpose

Behave as if XON Received (start transmitting)

Parameter Packet Format

None. Packet pointer must be NULL.

Data Packet Format

None. Packet pointer must be NULL.

Returns

If the call is made with invalid Parameter/Data Packet values, a general failure error is reported and this request is not performed by the device driver.

Remarks

If a general failure error is not returned by the device driver, this function allows data transmission to be resumed by the device driver if data transmission is halted due to a Category 1 function 47H (Behave as if XOFF received) request or due to an XOFF character being received while the device driver is in automatic transmit flow control mode.

Note: There still may be other reasons why transmission may be disabled; so transmission may not be resumed. (See Return COM Status, Category 1 Function 64H.)

Category 1 – Function 4BH

Purpose
Set Break On

Parameter Packet Format
None. Packet pointer must be NULL.

Data Packet Format

Field	Length
COM Error Word (COMERR)	WORD

Where

COM Error Word

The device driver returns this information if a general failure error is not reported. See Category 1 Function 6DH, Return COM Error, for COMERR definition. The COM device error information is not cleared by this action.

A CLOSE request packet, when after processing this close request the port will not be open any more (from another open without a close) will cause break to be turned off.

Returns
If the call is made with an invalid Parameter Packet value then a general failure error is reported, this function is not performed and valid information is not returned in the Data Packet.

Remarks
If a general failure error is not returned then the device driver will perform the following action:

The device driver will generate the break signal immediately. It is not considered an error if the device driver is already generating a break signal. The device driver will not wait for the transmit hardware to become empty. Note that more data will not be given to the transmit hardware until the break is turned off The break signal will always be transmitted, regardless of whether the device driver is or is not transmitting characters due to other reasons.

Category 1 – Function 53H

Purpose

Set Device Control Block (DCB)

Parameter Packet Format

Field	Length
Write Timeout	WORD
Read Timeout	WORD
Flags1	BYTE
Flags2	BYTE
Flags3	BYTE
Error Replacement Character	BYTE
Break Replacement Character	BYTE
XON Character	BYTE
XOFF Character	BYTE

Data Packet Format

None. Packet pointer must be NULL.

Where

Write Timeout

specifies the time period used for write timeout processing. The value is in .01 second units (based on 0, where 0 equals .01 seconds). Refer to Note 8: Write Timeout later in this chapter.

Read Timeout

specifies the time period used for read timeout processing. The value is in .01 second units (based on 0, where 0 equals .01 seconds). Refer to Note 9: Read Timeout later in this chapter.

Flags1:

Note	Bit	Meaning
1	0-1	DTR Control Mode
		Bit 1 ***Bit 0***
		0 0 Disable
		0 1 Enable
		1 0 Input handshaking
		1 1 Invalid input resulting in general failure
	2	reserved (set to 0)
3	3	Enable output handshaking using CTS
3	4	Enable output handshaking using DSR
3	5	Enable output handshaking using DCD
4	6	Enable input sensitivity using DSR
	7	reserved (set to 0)

Flags2:

Note	Bit	Meaning
2	0	Enable automatic transmit flow control (XON/XOFF)
2	1	Enable automatic receive flow control (XON/XOFF)
5	2	Enable error replacement character
6	3	Enable null stripping (remove null bytes)
7	4	Enable break replacement character
	5	reserved (set to 0)
1	6-7	RTS Control Mode
		Bit 7 ***Bit 6***
		0 0 Disable
		0 1 Enable
		1 0 Input handshaking
		1 1 Toggling on transmit

Flags3:

Note	Bit	Meaning
8	0	Enable write infinite time out processing
9	1-2	Read timeout processing

Bit 2	Bit 1	
0	0	Invalid input resulting in general failure
0	1	Normal read timeout processing
1	0	Wait for something, read timeout processing
1	1	No wait, read timeout processing

Bit	Meaning
3	reserved (set to 0)
4	reserved (set to 0)
5	reserved (set to 0)
6	reserved (set to 0)
7	reserved (set to 0)

Error Replacement Character
any byte value in the range 00H to FFH. Refer to Note 5: Error Replacement Character later in this chapter.

Break Replacement Character
any byte value in the range 00H to FFH. Refer to Note 7: Break Replacement Character later in this chapter.

XON Character
any byte value in the range 00H to FFH. Refer to Note 2: Automatic Flow Control later in this chapter.

XOFF Character
any byte value in the range 00H to FFH. Refer to Note 2: Automatic Flow Control later in this chapter.

Returns

If the call is made with invalid Parameter/Data Packet values, a general failure error is reported and none of the Device Control Block (DCB) characteristics of the device driver for this COM device are changed.

Remarks

The general Device Control Block (DCB) parameter access functions (53H and 73H) are used for:

- Automatic transmit flow control (start/stop transmit when XON/XOFF character received)
- Automatic receive flow control (transmit XON/XOFF when receive buffer fills/empties)
- Determine XON/XOFF characters
- DTR control mode (enable/disable/input handshaking)
- RTS control mode (enable/disable/input handshaking/toggling on transmit)
- Output handshaking using CTS/DSR/DCD (control signal determines when to transmit)
- Input sensitivity using DSR (reception of data controlled by DSR)
- Error replacement character and processing
- Break replacement character and processing
- Null stripping
- Receive/transmit time out processing

If a general failure error is not returned, the device driver will return valid information in the Data Packet.

The bit fields that are labeled Reserved (returned as 0), will return the current value (0) for these bits so a Set Device Control Block), will maintain the current device driver value for these bits. The bits defined as such will be returned as 0, but applications should not be written to make that assumption. Applications also should not be written to assume that the fourth bit combination will never be returned for DTR Control Mode or Read Timeout processing. Applications should not attempt to manipulate these Reserved bits.

Note: To maintain upward compatibility, the application should do a Return Device Control Block (DCB) information before the Set function is used. This will allow the reserved (set to 0) bits to be set correctly in a future release of the device driver when these bit positions may take on a real meaning. By doing the return first the application can maintain the state of the device driver for a mode that the application is not aware of.

Note 1: Control of DTR and RTS: The device driver allows the caller to automatically control the setting of Data Terminal Ready (DTR) and Request To Send (RTS) in many different ways via the RTS Control Mode and the DTR Control Mode settings of the Set Device Control Block IOCtl. The application can also request manual control over these modem control signals. The ways these signals are controllable are as follows:

Set RTS Control Mode to Toggling on Transmit.

If bits 7,6 of Flags 2 are set to 1,1 then the device driver is in this automatic control mode of RTS. When the device driver is initialized, the RTS Control Mode is Enable; so initially the device driver is not in this automatic control mode of RTS.

Note: This mode of operation of the device driver should only be enabled when the system is attached to devices which will not present data to the system receive hardware when RTS is on.

In this mode, the device driver will:

- Always turn on RTS if a break is being transmitted.
- Once data is in the transmit hardware buffer, the device driver will not turn RTS off until the transmit hardware has emptied its buffers.
- Turn on RTS (if not already on) if there is data in the device driver transmit queue OR if there is an outstanding WRITE request packet and:
  - The device driver is allowed to transmit even if automatic transmit/receive flow control (XON/XOFF) is enabled. Still need to turn on RTS momentarily to transmit a character immediate if not normally allowed to transmit due to automatic transmit/receive flow control (XON/XOFF).

  - The device driver is allowed to transmit because it was not told to behave as if an XOFF had been received (Category 1 Function 47H). The device driver will still need to turn on RTS momentarily to transmit a character immediate if not normally transmitting due to XOFF flow control considerations. The device driver will still need to turn on RTS momentarily to transmit an XON or XOFF due to automatic receive flow control if not normally transmitting due to XOFF flow control considerations.

- Turn off RTS (if not already off) if either of the following conditions are true:
  - No more data in the device driver transmit queue (and no more data in WRITE requests in progress), no queued WRITE requests, and the transmit hardware has physically transmitted (at the physical RS232 interface) all the data that it has been given.
  - The device driver is not allowed to transmit due to transmit/receive flow control (XON/XOFF) being enabled or due to being asked to behave as if an XOFF had been received (Category 1 Function 47H). The device driver still needs to turn on RTS to transmit a character immediate or XON/XOFF due to automatic receive flow control (XON/XOFF). RTS is never turned off until the transmit hardware has physically transmitted (at the physical RS232 interface) all the data that it has been given.
- When this function is enabled, the device driver will control RTS appropriately, as determined by the above description.
- If this function is disabled (by choosing a new RTS Control Mode), then RTS will be controlled appropriately by the new RTS Control Mode that is inherently chosen when this RTS Control Mode is disabled.
- The device driver will NOT examine any other modem control signals before it turns RTS off or on.

An OPEN request packet will not cause the device driver to change the RTS Control Mode that the device driver is in. The device driver will maintain the state of this mode of operation across OPEN request packets.

When the device driver is in the RTS Control Mode toggling on transmit, then the device driver will not allow the application to control RTS by the Set Modem Control Signals (Category 1 Function 46H).

Set DTR and/or RTS Control Mode to Input Handshaking.

Setting bits 1,0 of Flags 1 to 1,0 sets the DTR Control Mode to Input Handshaking. Setting bits 7,6 of Flags 2 to 1,0 sets the RTS control mode to Input Handshaking. When the device driver is initialized, the RTS and DTR Control Mode is Enable; so initially the device driver is not in this automatic control mode of RTS and DTR.

Note: This mode of operation of the device driver should only be set when there is the possibility of a device driver RECEIVE QUEUE overrun and the system is attached to data terminal equipment which will stop transmitting data when the appropriate modem control signals are turned off, due to the cabling and the data terminal equipment characteristics.

Because Input Handshaking mode can be set for either RTS or DTR or both, the DTR and RTS Control Modes are processed independently.

In Input Handshaking mode the device driver will:

- Turn the appropriate modem control signal(s) ON when the device driver receive queue is less than about half full.
- Turn the appropriate modem control signal(s) OFF when the device driver receive queue gets close to full.
- When this mode is first set, the device driver will not monitor the value of the appropriate modem control signal(s) (DTR or RTS) when the queue size is between approximately half full and almost full.
- When this function is enabled, the device driver will determine the correct value of the modem control signal(s) and control them accordingly.
- If this function is disabled (by choosing a new RTS and/or DTR Control Mode), RTS and/or DTR will be controlled appropriately by the new RTS and/or DTR Control Mode that is inher-

ently chosen when this RTS and/or DTR Control Mode is disabled.
- The device driver will NOT examine any other modem control signals before controlling DTR or RTS due to this mode.

An OPEN request packet will not cause the device driver to change the RTS and DTR Control Modes that the device driver is in. The device driver will maintain the state of these modes of operation across OPEN request packets.

When the device driver is in the RTS Control Mode Input Handshaking then the device driver will not allow the application to control RTS via Set Modem Control Signals (Category 1 Function 46H). When the device driver is in the DTR Control Mode Input Handshaking, then the device driver will not allow the application to control DTR via Set Modem Control Signals (Category 1 Function 46H).

Set DTR and/or RTS Control Mode to Enable or Disable.

OPEN processing

- Setting bits 1,0 of Flags 1 to 0,0 sets the DTR Control Mode to Disable.
- Setting bits 1,0 of Flags 1 to 0,1 sets the DTR Control Mode to Enable.
- Setting bits 7,6 of Flags 2 to 0,0 sets the RTS control mode to Disable.
- Setting bits 7,6 of Flags 2 to 0,1 sets the RTS control mode to Enable.

When the device driver is initialized, the RTS and DTR Control Mode is Enable, but the value of the modem control signals is OFF until the port gets an OPEN request packet.

An OPEN request packet will not cause the device driver to change the RTS and DTR Control Modes that the device driver is in. The device driver will maintain the state of these modes of operation across OPEN request packets.

Because Enable or Disable modes can be set for either RTS or DTR or both, the DTR and RTS Control Modes are processed independently. The following discussion covers what happens to RTS. The same discussion also applies to DTR if the DTR Control Mode is set as described in the RTS discussion.

If the RTS Control Mode is Disable, when the device driver receives an OPEN request packet and the device is not already open (from a previous open without a close - First Level Open), the RTS modem control signal will be kept (turned) OFF during the OPEN processing. If the RTS Control Mode is Enable then when the device driver receives an OPEN request packet and the device is not already open (from a previous open without a close - First Level Open), the RTS modem control signal will be turned ON during the OPEN processing.

If the RTS Control Mode is set to Disable and the previous mode was not Disable, the RTS modem control signal is turned OFF. If the RTS Control Mode is set to Disable and the previous mode was also Disable, this IOCtl has no effect on the RTS modem control signal.

If the RTS Control Mode is set to Enable and the previous mode was not Enable, the RTS modem control signal is turned ON. If the RTS Control Mode is set to Enable and the previous mode was also Enable, this IOCtl has no effect on the RTS modem control signal.

The following summarizes the previous discussion:

```
DTR/RTS:
(IH= Input Handshaking)
FROM(1) TO(1)  EFFECT(2)
------- ------- -------
Disable Disable none
Disable Enable  turn ON
Disable  IH     auto(3)

Enable  Disable turn OFF
Enable  Enable  none
Enable   IH     auto(3)

 IH     Disable turn OFF
 IH     Enable  turn ON
 IH      IH     auto(3)

RTS ONLY:
(toggle = toggle on transmit)

Disable toggle  auto(4)
Enable  toggle  auto(4)
 IH     toggle  auto(4)

toggle  Disable turn OFF
toggle  Enable  turn ON
toggle   IH     auto(3)
toggle  toggle  auto(4)
```

(1) - From or To Control Mode.

(2) - Effect on the modem control signal.

(3) - Modem control signal controlled automatically (See the section on Input Handshaking).

(4) - Modem control signal controlled automatically (See the section on Toggling on Transmit).

Because the initial Control Mode of the device driver is Enable for RTS and DTR, both modem control signals will be turned ON when the port is first opened.

If the device driver receives an OPEN request packet and the device is already open, the device driver does not alter the value of the RTS and DTR modem control signals, regardless of the Control Mode.

Category 1 – Function 53H

Application control of DTR and RTS.

The application can explicitly turn DTR or RTS ON or OFF (independently) with the Set Modem Control Signals IOCtl (Category 1 Function 46H).

If the Control Mode of RTS is not Enable or Disable, the application may not control RTS with the Set Modem Control Signals IOCtl because the device driver is controlling the signal automatically (toggling on transmit or input handshaking). If the Control Mode of DTR is not Enable or Disable, the application may not control DTR with the Set Modem Control Signals IOCtl because the device drive is controlling the signal automatically (input handshaking).

CLOSE processing.

If the device driver receives a CLOSE request packet and the COM device will still be open (from another open without a close) then the device driver will not change the values of DTR or RTS.

If the device driver receives a CLOSE request packet, when after processing this close request the port will not be open any more (from another open without a close - Last Level Close) then, at the end of the CLOSE processing, RTS and DTR will be turned OFF by the device driver; after waiting the appropriate amount of time. (See description of CLOSE processing and IOCtl Set Modem Control Signals, Category 1 Function 46H).

Note 2: Automatic Flow Control. XON/XOFF Characters: If bit 0 of Flags 2 is set, the device driver is enabled for automatic transmit flow control. If bit 1 is set, the device driver is enabled for automatic receive flow control.

When the device driver is initialized these bits are reset, so initially the device driver is not enabled for automatic transmit or receive flow control.

An OPEN request packet will not cause the device driver to change the enabling or disabling state of automatic transmit/receive flow control.

An OPEN request packet, when the COM device is not already open (from a previous open without a close - First Level Open), will cause the device driver to believe it has not received an XOFF if automatic transmit flow control is enabled and will cause the device driver to believe it has not transmitted an XOFF if automatic receive flow control is enabled.

Automatic Transmit Flow Control (XON/XOFF)

In the discussion that follows it is stated in places that the device driver will transmit XON or XOFF. There are reasons why the device driver may not be able to transmit an XON or XOFF (transmitting break, invalid output handshaking on modem control signals). It is also stated that the device driver will resume transmitting data. There are also other potential reasons (not related to automatic transmit/receive flow control) why that may not be possible. See Return COM Status IOCtl (Category 1 Function 64H).

When XON and XOFF flow control during transmission is enabled, the device driver will stop sending data to the transmit hardware when an XOFF is received, and resume sending data to the transmit hardware when an XON is received. (Reminder: Transmission may not be possible due to other reasons). The device driver will still transmit characters due to the transmit immediate request IOCtl (Category 1 Function 44H). The device driver will still transmit XON and XOFF due to automatic receive flow control.

When the device driver is in this mode, it will not pass received XON and XOFF characters to the application. Instead, the device driver will act upon receiving those characters and throw them away.

The device driver may transmit additional characters before it recognizes an XOFF character which it has not read but which may be in the receive buffer of the hardware. The extent of this scenario will be minimized, but the combined transmit/receive Advanced BIOS request block will still be used on systems that support Advanced BIOS. If the system is relatively slow in responding to interrupts compared to the current baud rate, receive buffer overruns may not be occurring, but the device driver may be apparently slow in responding to an XOFF character.

If automatic transmit flow control is disabled (after currently enabled) and transmission was not occurring due to an XOFF being received or IOCtl Category 1 Function 47H (behave as if XOFF received) being requested, then transmission will be resumed. (Reminder: transmission may not be resumed for other reasons.)

See States of the RS232 device driver for the effect of OPEN request packets on this mode of the device driver. It is the application's responsibility not to fully close the port in a way that will cause the device driver to illegally transmit characters when the port is re-opened after being fully closed (First Level Open).

Output handshaking using modem control signals is another way that the device driver can be told to stop transmitting. See Note 3.

Automatic Receive Flow Control (XON/XOFF)

When XON and XOFF flow control during receive is enabled, the device driver will transmit an XOFF when its receive queue gets close to full, and an XON when its receive queue is about half full. After the XOFF is sent, the COM device driver will send no characters until it sends an XON due to automatic receive flow control. This is to accommodate those systems that interpret the first character received after an XOFF as an XON, regardless of what the character actually is. The device driver will still transmit characters due to the transmit immediate request (Category 1 Function 44H).

The device driver will not be able to transmit an XOFF or XON if it is transmitting a break. The device driver will not be able to automatically transmit an XOFF or XON if it is enabled for output handshaking, with certain modem control signals, and those modem control signals are not ON. This could cause a deadlock if the device driver wishes to transmit an XON and it cannot. The device driver will remember that it wanted to transmit an XOFF or XON and will still do so when transmit conditions permit; assuming the receive queue conditions still warrant it.

The device driver will not monitor characters being transmitted by WRITE request packets to see if any of them are XON or XOFF. The device driver will also not monitor characters transmitted immediately (Category 1 Function 44H). For example, the device driver will not stop transmitting characters if the application causes the device driver to explicitly transmit an XOFF.

If automatic receive flow control is enabled (after currently disabled), the device driver will immediately check the receive queue level to see if an XOFF needs to be transmitted. An XON is never transmitted immediately due to this function being enabled. The device driver will only automatically transmit an XON character after it has automatically transmitted an XOFF character.

If automatic receive flow control is disabled (after currently enabled), and transmission was not occurring due to an XOFF being automatically transmitted, the device driver will transmit an XON and transmission will be resumed if possible. (Reminder: transmission may not be taking place for other reasons.)

If the device driver previously automatically transmitted an XOFF, and a CLOSE request packet is received, when after processing this close request the port will not be open any more (from another open without a close), the device driver will automatically transmit an XON if possible.

See states of the device driver for the effect of OPEN request packets on this mode of the device driver. It is the application's responsibility not to fully close the port in a way that will cause the device driver to illegally transmit characters or the communications link to be in a deadlock state when the port is re-opened after being fully closed (First Level Open).

Input handshaking using modem control signals is another way that the device driver can tell another device to stop transmitting. See Note 1.

XON and XOFF CHARACTERS

The value of these bytes in the device control block determine the value of the XON and XOFF character that is used for automatic transmit and receive flow control.

When the XON and XOFF characters are referred to in the Category 1 IOCtl section, the reference is to the value of the XON and XOFF character as determined by this IOCtl.

When the device driver is first initialized, the XON character is 11H and the XOFF character is 13H. An OPEN request packet, when the COM device is not already open, (from a previous open without a close - First Level Open), will cause the XON character to be set to 11H and the XOFF character to be set to 13H.

If the XON and XOFF characters are set equal with this IOCtl, the results are UNDEFINED.

Note 3: Output Handshaking Using CTS, DSR, DCD.: Bits 3, 4, and 5 of Flags 1 control output handshaking using CTS, DSR, and DCD respectively. If the bit is set, output handshaking for the appropriate modem control signal is enabled.

Output Handshaking mode can be enabled for any combination of CTS, DSR, or DCD because bit 3, 4, and 5 of Flags 1 are processed independently.

When the device driver is initialized, bits 3 and 4 of Flags1 are set and bit 5 of Flags1 is reset; Therefore, so initially the device driver is enabled for output handshaking using CTS and DSR but disabled for output handshaking using DCD.

Except for attachment to special devices and/or special cables, output handshaking using DCD should not be enabled.

Disabling output handshaking using CTS and/or DSR will cause unexpected results when the system is attached to data terminal devices or data communications devices that toggle CTS and/or DSR in order to control the ability of the system to transmit data.

If the device driver is enabled for this mode of operation, the device driver will be affected in the following manner if the appropriate modem signal(s) are OFF:

- The device driver will be unable to move data from the device driver transmit queue to the transmit hardware.
- The device driver will be unable to transmit a character immediately (Category 1 Function 44H) so the character is remembered by the device driver.
- The device driver will be unable to automatically transmit XONs and XOFFs. The device driver may wish to transmit XONs and XOFFs as a result of automatic receive flow control being enabled.
- The device driver will still generate a break immediately if requested.
- The value of CTS, DSR, and DCD do not affect how the device driver controls RTS and DTR.

An OPEN request packet will not cause the device driver to change the value of bits 3, 4, and 5 of Flags 1. The device driver will maintain the state of this mode of operation across OPEN request packets.

On devices with a transmit holding register and transmit shift register, the transmit holding register will always be given another character to transmit when it empties (even though a character may still be in the transmit shift register), unless the device driver determines that is not allowed to transmit any more.

The device driver will always attempt to detect a change in the modem status signals (CTS, DSR, DCD) before transmitting more data. This requires bypassing the natural priority of the current ASYNC hardware and requires additional complexity in the Advanced

BIOS implementation. This feature prevents a temporary elongation of interrupt latency from not allowing the device driver to recognize a change in a modem control signal. (The modem control signal change happened many character times before the transmit hardware is requesting that another character be given to it).

Note 4: Input Sensitivity Using DSR.: Bit 6 of Flags 1 controls input sensitivity using DSR. If the bit is set, input sensitivity using DSR is enabled.

When the device driver is initialized, bit 6 of Flags 1 is set; so initially the device driver is enabled for input sensitivity using DSR.

Note: Disabling input sensitivity using DSR will cause unexpected results when the system is attached to data terminal devices or data communications devices that toggle DSR when they generate spurious data that they do not wish the system to receive.

If the device driver is enabled for this mode of operation, the device driver will throw away all data input from the receive hardware if DSR is off.

If the device driver processes a change in the DSR modem control signal from ON to OFF or OFF to ON at the same time that it inputs a character from the receive hardware, the device driver will still accept that last character(s). This will prevent a temporary elongation of interrupt latency from causing the device driver to discard a valid character(s). However this could cause the device driver to attempt to process invalid data for one service period of the receive hardware. This requires that the change in the modem control signal gets processed before the device driver attempts to receive data from the receive hardware (See Note 3); or that the received data is saved until a change in modem status (during the same hardware service instance) can be determined.

An OPEN request packet will not cause the device driver to change the value of bit 6 of Flags 1. The device driver will maintain the state of this mode of operation across OPEN request packets.

Note 5: Error Replacement Character: Bit 2 in Flags 2 controls the enabling of error replacement character processing. If the bit is set, the error replacement character processing is enabled.

When the device driver is initialized this bit is reset, so initially the device driver is not enabled for the error replacement character. An OPEN request packet, when the COM device is not already open (from a previous open without a close - First Level Open) will cause this bit to be reset, disabling error replacement character processing.

When the device driver is initialized, the error replacement character is 00H. An OPEN request packet, when the COM device is not already open (from a previous open without a close) will cause the error replacement character to be set back to a 00H.

If error replacement character processing is disabled, the following applies:

- If a parity or framing error occurs and if the character that had the error is available in the receive hardware buffer, it is placed in the device driver receive queue.
- If a hardware or receive queue overrun occurs then nothing special is placed in the receive queue to designate an overrun.

If error replacement character processing is enabled, the following applies:

- If a parity or framing error occurs, the error replacement character is placed in the device driver receive queue. (The character in the receive hardware buffer, if it was available, is not placed in the receive queue.)
- If a hardware buffer overrun occurs, the error replacement character is placed in the device driver receive queue to mark the position that a receive overrun occurred. If valid data is in the receive hardware buffer, it is still placed in the device driver receive queue. The processing of the valid data takes place after the hardware buffer overrun condition is recorded in the device driver receive queue.
- If a device driver receive queue overrun occurs, the last character in the receive queue is replaced with the error replacement character. This allows the application to know the position of

where the error occurred. This error replacement (if enabled) will always take precedence over an error replacement or break replacement event that occurred at the same character time.

Regardless of whether error replacement character processing is enabled, null stripping and checking for XON/XOFF characters will not occur if the character had an error.

This IOCtl can be used to change the error replacement character by changing the byte representing the error replacement character.

Note 6: Null Stripping: Bit 3 in Flags 2 controls the enabling of null stripping processing. If the bit is set, null stripping processing is enabled.

When the device driver is initialized this bit is reset, so initially the device driver is not enabled for null stripping. An OPEN request packet, when the COM device is not already open (from a previous open without a close - First Level Open) will cause this bit to be reset; disabling null stripping.

If the device driver is enabled for null stripping when characters are read in from the receive hardware any (non error or non break) characters with a value of 00H are thrown away, are not checked even if the XON or XOFF character has been set to 00H, and are not placed in the device driver receive queue.

Note: Simultaneously setting the XON or XOFF character to 00H, enabling automatic transmit flow control, and enabling null stripping may cause unexpected results but is not considered an error condition by the device driver error checking logic.

Note 7: Break Replacement Character: Bit 4 in Flags 2 controls the enabling of break replacement character processing. If the bit is set, the break replacement character processing is enabled.

When the device driver is initialized this bit is reset, so initially the device driver is not enabled for the break replacement character. An OPEN request packet, when the COM device is not already open (from a previous open without a close - First Level Open) will cause this bit to be reset, disabling break replacement character processing.

When the device driver is initialized, the break replacement character is 00H. An OPEN request packet, when the COM device is not already open (from a previous open without a close - First Level Open) will cause the break replacement character to be reset back to a 00H.

If break replacement character processing is disabled, the device driver will not place any character in the device driver receive queue when it detects a break condition on the line. A detected break condition has no effect on XON/XOFF detection.

If break replacement character processing is enabled, when the device driver detects a break condition, it will place the break replacement character in the device driver receive queue.

If break replacement character processing is enabled, null stripping and checking for XON/XOFF characters will not operate on the break replacement character.

This IOCtl can be used to change the break replacement character by changing the byte representing the break replacement character.

If a parity or framing error is generated due to the reception of a break, error replacement processing is not done (except for the overrun condition), break replacement processing is done.

Note 8: Write Timeout: Bit 0 in Flags3 controls the characteristics of Write timeout processing. If the bit is 0, Write timeout processing uses the value in the Write Timeout word in the device control block. If the bit is 1, Write timeout processing is infinite time out.

The value in the Write Timeout word is in .01 second units (based on 0 where, 0 = .01 seconds). The device driver is considered doing normal write timeout processing when the Write Timeout word is used for write timeout processing.

During normal write timeout processing, if the device driver does not give ANY data to the transmit hardware (from the transmit queue) within the period of time specified by the Write Timeout word (due to some reason that prevents the device driver from transmitting data), the request will be completed. The accuracy of the time out period MAY be determined by the request packet being blocked in the device driver and how long it takes for the thread to be dispatched once it is made ready by the time out period expiring; OR the accuracy of the time out period MAY be determined by the accuracy of the device driver timer tick processing. If any data had been given to the transmit hardware in that time out period, the specified period of time will be waited for again, to see if any more data had been transmitted.

If the time out period is changed by this IOCtl (or to infinite time out), the new time may take effect immediately or may take effect after the next character is written.

During write infinite time out processing, the request will not complete until all the data from the request has been given to the transmit hardware. The thread of the Write request will not return to the system until the request completes. The device driver will check to see if an IOCtl has changed the write timeout processing characteristics at least every minute. This could occur almost immediately (accuracy MAY be determined by the request packet being blocked and/or by device driver timer ticks). This will insure that the device driver periodically checks to see if write infinite time out processing had been changed to normal write timeout processing.

As discussed above, the write timeout characteristics can be changed in the middle of the processing of a write request and the new time out attributes is guaranteed to eventually take effect.

When the device driver initializes, normal write timeout processing is in effect.

When the device driver receives an OPEN request packet for the port and the port is not already open (from a previous open without a matching close - First Level Open), the value in the write timeout word is set to 1 minute. The current write timeout processing characteristics (normal or infinite) is not affected.

Note 9: Read Timeout: Bits 2, 1 of flags 3, control the Read timeout processing characteristics of the device driver. The three possible types of Read timeout processing are:

- Normal (Bits 2,1= 0,1)
- Wait For Something (Bits 2,1= 1,0)
- NO WAIT (Bits 2,1= 1,1)

The value in the Read Timeout word is in .01 second units (based on 0, where 0 = .01 seconds). The device driver uses the value in the Read timeout word for Normal and Wait For Something Read timeout processing. The accuracy of the time interval MAY be determined by the request being blocked in the device driver and/or by device driver timer ticks.

However, in the following two cases (of data being received), the current interval of time will continue to be waited on without starting to wait from the beginning of the interval again:

1. if input sensitivity using DSR is ENABLED and the value of the DSR modem control signal causes input data to be thrown away. Refer to Note 4: Input Sensitivity using DSR earlier in this chapter.
2. if null stripping is ENABLED and a null character is stripped. Refer to Note 6: Null Stripping earlier in this chapter.

If the device driver is doing Normal Read time out processing, the device driver will wait as long as the value in the Read timeout word says to wait. The request will be completed after that interval of time elapses, if no more data has been received for the request. If any data is received by the device driver (from the receive hardware) for the request (including XON/XOFF characters), the specified period of time will be waited on again (for more data to arrive).

If the device driver is doing NO WAIT read timeout processing, the device driver will not wait for any data to be available in the receive queue. When the device driver begins to try to move data from the receive queue to the request, the request will complete. Whatever data is available in the receive queue at that time is the amount of data that will be moved to the request.

If the device driver is doing Wait For Something read timeout processing, the device driver will process the request initially as if it had no wait time out processing. If no data was available at the time the request would have completed due to NO WAIT processing, the request is not completed. Instead, the request waits for some data to be available before completing the request. However, the device driver does enter Normal read timeout processing for this request. Therefore, if no data is available after the Normal timeout processing interval then the request will be completed anyway. The request will never wait any longer then it would have due to Normal read timeout processing.

The read timeout processing characteristics that apply to a given read request is not determined until the device driver begins processing that request. Once the device driver begins processing that request, a change to the read timeout processing characteristics of the device driver between Wait For Something and Normal time out processing may or may not take effect for the current read request being processed. If the time out period is changed by this IOCtl, the new time out period may take effect immediately, or it may take effect after the next character is received from the receive hardware.

When the device driver initializes, normal read timeout processing is in effect.

When the device driver receives an OPEN request packet for the port and the port is not already open (from a previous open without a matching close - First Level Open), the value in the write timeout word is set to 1 minute and normal read timeout processing characteristics is put into effect.

Purpose

Return Baud (bit) Rate

Parameter Packet Format

None. Packet pointer must be NULL.

Data Packet Format

Field	Length
Bit Rate	WORD

Where

Bit Rate
The binary integer representing the actual baud (bit) rate of the COM device in bits per second.

Returns

If the call is made with an invalid Parameter Packet value then a general failure error is reported and valid information is not returned in the Data Packet.

Remarks

If a general failure error is not returned then the device driver will return the current baud (bit) rate of the COM device.

Category 1 – Function 62H

Purpose
Return Line Characteristics (stop bits, parity, data bits, break)

Parameter Packet Format
None. Packet Pointer must be NULL.

Data Packet Format

Field	Length
Data Bits	BYTE
Parity	BYTE
Stop Bits	BYTE
Transmitting Break	BYTE

Where

Data Bits
See set function 42H (Set Line Characteristics)

Parity
See set function 42H (Set Line Characteristics)

Stop Bits
See set function 42H (Set Line Characteristics)

Transmitting Break
0 means not currently transmitting break. 1 means currently transmitting break.

Returns
If the call is made with an invalid Parameter Packet value then a general failure error is reported and valid information is not returned in the Data Packet.

Remarks

If a general failure is not returned, the device driver will return the line characteristics as defined.

Category 1 – Function 64H

Purpose
Return COM Status

Parameter Packet Format
None. Packet pointer must be NULL.

Data Packet Format

Field	Length
COM Status Byte	BYTE

Where

COM Status Byte

1	The condition is true
0	The condition is false

Note	Bit	Meaning
3	0	Tx waiting for CTS to be turned ON
3	1	Tx waiting for DSR to be turned ON
3	2	Tx waiting for DCD to be turned ON
1	3	Tx waiting because XOFF received
2	4	Tx waiting because XOFF transmitted
5	5	Tx waiting because break being transmitted
6	6	Character waiting to transmit immediately
4	7	Receive waiting for DSR to be turned ON

Tx (transmit) status indicates why we may not be transmitting; regardless of whether there is data to transmit. However the device driver must be enabled for the given condition (for example, enabled for output handshaking for the modem control signal in question) for the status to reflect that the device driver would be waiting for the given condition to transmit.

For example, 00000001 means the device driver will put receive characters in the device driver receive queue, the device driver is not waiting to transmit a character immediately (Category 1 Function 44H), and we will not transmit characters from the device driver transmit queue because we are using CTS for output handshaking and CTS does not have the proper value.

Note:

1. This occurs because the proper conditions when the device driver is enabled for automatic transmit flow control (XON/XOFF), or the device driver is not enabled for automatic transmit flow control (XON/XOFF) and the device driver is told to behave as if an XOFF had been received (Category 1 Function 47H).

   Characters will still be transmitted immediately (Category 1 Function 44H) and the device driver can still automatically transmit XONs and XOFFs because of automatic receive flow control (XON/XOFF) when the device driver is in this state.
2. This is because of the proper conditions when the device driver is enabled for automatic receive flow control.

   Characters will still be transmitted immediately (Category 1 Function 44H) and the device driver can still automatically transmit XONs because of the automatic receive flow control when the device driver is in this state.
3. See Set Device Control Block Note 3 (Category 1 Function 53H).
4. See Set Device Control Block Note 4 (Category 1 Function 53H).
5. See Set break on (Category 1 Function 4BH).
6. See Transmit Byte Immediate (Category 1 Function 44H).

Returns

If the call is made with an invalid Parameter Packet value a general failure error is reported, and valid information is not returned in the Data Packet.

Remarks

If a general failure error is not returned the device driver will return the COM device current status.

Category 1 – Function 65H

Purpose

Return Transmit Data Status

Parameter Packet Format

None. Packet Pointer must be NULL.

Data Packet Format

Field	Length
Transmit Status	BYTE

Where

Transmit Status

Is returned as bit significant values. If the bit is 1, the condition is true. The bit is 0 if the condition is false. The number at the beginning of the description is the bit position number. The bit positions go from least to most significant.

0 - WRITE request packets in progress or queued.
1 - Data in the device driver transmit queue.
2 - The transmit hardware is currently transmitting data.
3 - Character waiting to be transmitted immediately.
4 - Waiting to automatically transmit an XON.
5 - Waiting to automatically transmit an XOFF.
6 - undefined
7 - undefined

Returns

If the call is made with an invalid Parameter Packet value a general failure error is reported and valid information is not returned in the Data Packet.

Remarks

If a general failure error is not returned the device driver will return the current transmit status of the COM device.

Category 1 – Function 66H

Purpose

Return Modem Control Output Signals

Parameter Packet Format

None. Packet pointer must be NULL.

Data Packet Format

Field	Length
Modem Control Output Signals	BYTE

Where

Modem Control Output Signals

A bit value of 1 means the condition is ON. A bit value of 0 means the condition is OFF.

Bit	*Meaning*
0	Data terminal ready (DTR)
1	Request to send (RTS)
2 to 7	Undefined

Returns

If the call is made with an invalid Parameter Packet value a general failure error is reported and valid information is not returned in the Data Packet.

Remarks

If a general failure error is not returned, the device driver will return the current modem control output signals of the COM device.

Purpose
Return Current Modem Control Input Signals

Parameter Packet Format

Field	Length
Modem Control Input Signals	BYTE

Data Packet Format
None. Packet pointer must be NULL.

Where

Modem Control Input Signals
A bit value of 1 means the condition is ON. A bit value of 0 means that the condition is OFF.

Bit	*Meaning*
0 to 3	Undefined
4	Clear To Send (CTS)
5	Data Set Ready (DSR)
6	Ring Indicator (RI)
7	Data Carrier Detect (DCD)

Returns
If the call is made with an invalid Parameter Packet value a general failure error is reported and valid information is not returned in the Data Packet.

Remarks
If a general failure error is not returned, the device driver will return the current modem control input signals of the COM device.

Category 1 – Function 68H

Purpose

Return Number of Characters in the Receive Queue

Parameter Packet Format

None. Packet pointer must be NULL.

Data Packet Format

Field	Length
Number of Characters Queued	WORD
Size of Receive Queue	WORD

Where

Number of Characters Queued

Binary integer with the number of received characters in the device driver receive queue. The device driver receive queue is a memory buffer in between the memory pointed to by the READ request packet and the receive hardware for this COM device. The application may not assume that there are no unsatisfied READ requests if there are characters in the device driver receive queue. The behavior of data movement between the READ request and the receive queue may change from release to release of the device driver. Applications should not be written to have a dependency on this information.

Size of Receive Queue

Binary integer with the size of the device driver receive queue. Applications should be written to be independent of the receive queue being of a fixed size. The information in this field allows the application to get the size of the receive queue. The current size of the receive queue is approximately 1K bytes but is subject to change.

Using this information, the application could be written to avoid device driver receive queue overruns. This could be done by using an application to application block protocol with the system that the application is communicating with.

Returns

If the call is made with an invalid Parameter Packet value, a general failure error is reported and valid information is not returned in the Data Packet.

Remarks

If a general failure error is not returned, the device driver will return the information.

Category 1 – Function 69H

Purpose

Return Number of Characters in Transmit Queue

Parameter Packet Format

None. Packet pointer must be NULL.

Data Packet Format

Field	Length
Number of Characters Queued	WORD
Size of Transmit Queue	WORD

Where

Number of Characters Queued

Binary integer with the number of characters ready to be transmitted in the device driver transmit queue. The device driver transmit queue is a memory buffer between the memory pointed to by the WRITE request packet and the transmit hardware for this COM device. If the transmit queue is empty, the application may not assume that all WRITE requests have completed or that no WRITE requests are outstanding. The behavior of data movement between the WRITE request and the transmit queue may change from release to release of the device driver. Applications should not be written to be dependent on this information.

Size of Transmit Queue

Binary integer with the size of the device driver transmit queue. Applications should be written to be independent of the transmit queue being of a fixed size. The information in this field allows the application to get the size of the transmit queue. The size of the transmit queue is 128 bytes. Applications should not be written to have a dependency on this value as it is subject to change.

Returns

If the call is made with an invalid Parameter Packet value, a general failure error is reported and valid information is not returned in the Data Packet.

Remarks

If a general failure error is not returned, the device driver will return the information.

Category 1 – Function 6DH

Purpose

Return COM Error (retrieve and then clear the COM device error information)

Parameter Packet Format

None. Packet pointer must be NULL.

Data Packet Format

Field	Length
COM Error Word (COMERR)	WORD

Where

COM Error

The appropriate bits in the COM Error Word are set by the device driver when the events described below occur. The COM error word is not cleared unless this function is performed by the device driver or an OPEN request packet is received by the device driver and the COM device is not already open (from a previous open without a close first level open). See Note 6 of Set Device Control Block (Category 1 Function 53H).

Bit	***Meaning***
0	Receive queue overrun. No room in the device driver receive queue to put a character read in from the receive hardware.
1	Receive hardware overrun. A character was not read from the hardware before the next character arrived causing a character to be lost.
2	The hardware detected a parity error.
3	The hardware detected a framing error.
4 to 15	Undefined

Returns

If the call is made with an invalid Parameter Packet value, a general failure error is reported, valid information is not returned in the Data Packet, and the COM error word is not cleared.

Remarks

If a general failure error is not returned, the device driver will return and clear the COM device error information.

Category 1 – Function 72H

Purpose

Return COM Event Information (retrieve and then clear the COM device event word)

Parameter Packet Format

None. Packet pointer must be NULL.

Data Packet Format

Field	Length
COM Event Word	WORD

Where

COM Event Word

The appropriate bits in the COM Event Word are set by the device driver when the following events occur:

Note: The COM Event Word is not cleared unless this function is performed by the device driver or an OPEN request packet is received by the device driver and the COM device is not already open (from a previous open without a close) (first level open).

Bit	*Meaning*
0	Set when any character is read from the COM device receive hardware and placed in the receive queue.
1	Undefined
2	Set when the last character in the device driver transmit queue is sent to the COM device transmit hardware. This does not mean that there is no data to send in any outstanding WRITE requests.
3	Set when the Clear to Send (CTS) signal changes state.
4	Set when the Data Set Ready (DSR) signal changes state.
5	Set when the Data Carrier Detect (DCD) signal changes state.
6	Set when a break is detected.

7	Set when a parity, framing or overrun error occurs (an overrun can be a receive hardware overrun or a receive queue overrun).
8	Set when trailing edge of Ring Indicator is detected.
9 to 15	Undefined

Returns

If the call is made with an invalid Parameter Packet value, a general failure error is reported, valid information is not returned in the Data Packet, and the event word is not cleared.

Remarks

If a general failure is not returned, the device driver will return the current value of the event word and then clear it.

Category 1 – Function 73H

Purpose
Return Device Control Block (DCB) Information

Parameter Packet Format
None. Packet pointer must be NULL.

Data Packet Format

Field	Length
Write Timeout	WORD
Read Timeout	WORD
Flags1	BYTE
Flags2	BYTE
Flags3	BYTE
Error Replacement Character	BYTE
Break Replacement Character	BYTE
XON Character	BYTE
XOFF Character	BYTE

Where

Write Timeout
specifies the time period used for write timeout processing. The value is in .01 second units (based on zero, where zero equals .01 seconds). Refer to Note 8: Write Timeout earlier in this chapter.

Read Timeout
specifies the time period used for read timeout processing. The value is in .01 second units (based on zero, where zero equals .01 seconds). Refer to Note 9: Read Timeout earlier in this chapter.

Flags1

Bit	***Meaning***		
0-1	DTR Control Mode		
	Bit 1	***Bit 0***	
	0	0	Disable
	0	1	Enable
	1	0	Input handshaking
2	Reserved (returned as zero)		
3	Enable output handshaking using CTS		
4	Enable output handshaking using DSR		
5	Reserved (returned as zero)		
6	Enable input sensitivity using DSR		
7	Reserved (returned as zero)		

Flags2

Bit	***Meaning***		
0	Enable automatic transmit flow control (XON/XOFF)		
1	Enable automatic receive flow control (XON/XOFF)		
2	Enable error replacement character		
3	Enable null stripping (remove null bytes)		
4	Enable break replacement character		
5	Reserved (returned as zero)		
6-7	RTS Control Mode		
	Bit 7	***Bit 6***	
	0	0	Disable
	0	1	Enable
	1	0	Input handshaking
	1	1	Toggling on transmit

Category 1 – Function 73H

Flags3

Bit	*Meaning*
0	Enable write infinite timeout processing
1-2	Read timeout processing

	Bit 2	***Bit 1***	
	0	1	Normal read timeout processing
	1	0	Wait for something, read timeout processing
	1	1	No wait, read timeout processing
3	Reserved (returned as zero)		
4	Reserved (returned as zero)		
5	Reserved (returned as zero)		
6	Reserved (returned as zero)		
7	Reserved (returned as zero)		

See function 53H, Set Device Control Block (DCB) for field definitions.

Error Replacement Character
any byte value in the range 00H to FFH. Refer to Note 5: Error Replacement Character earlier in this chapter.

Break Replacement Character
any byte value in the range 00H to FFH. Refer to Note 7: Break Replacement Character earlier in this chapter.

XON Character
any byte value in the range 00H to FFH. Refer to Note 2: Automatic Flow Control earlier in this chapter.

XOFF Character
any byte value in the range 00H to FFH. Refer to Note 2: Automatic Flow Control earlier in this chapter.

Returns

If the call is made with an invalid Parameter Packet value, a general failure error is reported and valid information is not returned in the Data Packet.

Category 1 – Function 73H

Remarks

The general Device Control Block (DCB) parameter access functions (53H and 73H) are used for:

- Automatic transmit flow control (start/stop transmit when XON/XOFF character received)
- Automatic receive flow control (transmit XON/XOFF when receive buffer fills/empties)
- Determine XON/XOFF characters
- DTR control mode (enable/disable/input handshaking)
- RTS control mode (enable/disable/input handshaking/toggling on transmit)
- Output handshaking using CTS/DSR/DCD (control signal determines when to transmit)
- Input sensitivity using DSR (reception of data controlled by DSR)
- Error replacement character and processing
- Break replacement character and processing
- Null stripping
- Receive/transmit timeout processing

If a general failure error is not returned, the device driver will return valid information in the Data Packet.

Note: To maintain upward compatibility, it is the responsibility of the application to do a Return Device Control Block (DCB) Information BEFORE doing a Set Device Control Block (DCB). The appropriate information in the returned control block should be modified by the application and then the Set function can be done.

Category 3 Pointer Draw Control IOCtl Commands

Following is a summary of Category 3 descriptions:

Function	***Description***
72H	Get pointer draw address

Category 3 – Function 72H

Purpose
Get pointer draw address

Parameter Packet Format
None

Data Packet Format

Field	Length
Return Code	WORD
Pointer Draw Routine Entry Point (Selector:Offset)	DWORD
Pointer Draw Routine Data Segment Selector	WORD

Returns
Information in data packet defined above.

Remark
The call is used by the mouse subsystem to obtain the entry point address of the pointer draw routine. The pointer draw routine is contained within the pointer draw device driver. It is called by the mouse device driver to update the pointer image on the screen. The far address returned by function code 72H is passed by the mouse subsystem to the mouse device driver through an IOCtl interface (reference the mouse control IOCtl commands, category 07H). The mouse device driver saves the far address passed and uses it when calling the pointer draw routine.

This function is supported by the pointer draw device driver.

Category 4 Keyboard Control IOCtl Commands

Following is a summary of Category 4 descriptions:

Function	*Description*
50H	Set code page
51H	Set input mode (default ASCII)
52H	Set interim character flags
53H	Set shift state
54H	Set typematic rate and delay
55H	Notify of change of foreground session
56H	Set session manager Hot Key
57H	Set KCB
58H	Set code page ID
5BH	Reserved
5CH	Set NLS & custom code page
71H	Get input mode
72H	Get interim character flags
73H	Get shift state
74H	Read character data record(s)
75H	Peek character data record
76H	Get session manager Hot Key
77H	Get keyboard type
78H	Get code page ID
79H	Translate scan code to ASCII

Category 4 – Function 50H

Purpose

Set Code Page

Parameter Packet Format

Field	Length
Pointer to Code Page	DWORD

Data Packet Format

None

Where

Code Page Format

has the following format where there are 127 copies of the KeyDef rec below (includes 1 for each possible scan code that may be returned from the keyboard). Not all entries are used; unused entries are zero. The entries are in scan code order, based on the remapped scan codes returned by the keyboard controller when it is in the DOS execution environment. The DOS execution environment translates keyboard scan codes to scan codes based on the position of the keys as they are on the standard PC keyboard (plus additional keys on the Enhanced Keyboard). The DOS execution environment also converts key "break" codes to the equivalent scan code with the high order bit turned on (that is, adds 128 to the code).

```
XlateTable:
  XHeader   : XHeader
  KeyDef1   : KeyDef
  KeyDef2   : KeyDef
  KeyDef3   : KeyDef
     .          .
     .          .
     .          .
  KeyDef127 : KeyDef
  AccentTbl : AccentTable
End XlateTable
XHeader:
  XTableID          : Word        [Code Page ID               ]
  XTableFlags1      : Rec[Word Width]
                    : The following three bits determine which
                    : shift key or key combination affects CHAR3
                    : of each KeyDef.
     ShiftAlt       : Bit 0 [Use Shift-Alt instead of Ctrl-Alt]
     AltGrafL       : Bit 1 [Use left Alt key as Alt-Graphics]
     AltGrafR       : Bit 2 [Use right Alt key as Alt-Graphics]
     ShiftLock      : Bit 3 [Treat Caps Lock As Shift Lock    ]
     DefaultTable   : Bit 4 [Default table for the Lang.]
     ShiftToggle    : Bit 5 [Toggle or Latch ShiftLock]
                    :       When 1 toggle else latch
     AccentPass     : Bit 6 [Pass accent and non-accent key through]
                    :       When 1 pass on accent keys and beep,
                    :       else beep only.
                    : The following four bits determine which
                    : shift key or key combination causes Char5
                    : to be used in each KeyDef.
     CapsShift      : Bit 7 [Caps-Shift uses CHAR5]
     Reserved       : Bit 8-10
     Reserved       : Bit 11 - 15
  EndRec XtableFlags1
  XTableFlags2      : Rec[Word Width]
     Reserved       : Bit 0 - 15
  EndRec XtableFlags2
  KbdType           : Word      [Keyboard type, see below  ]
  KbdSubType        : Word      [Reserved.                 ]
  XtableLen         : Word      [Length of Table           ]
  EntryCount        : Word      [Number of KeyDef entries  ]
```

```
    EntryWidth        : Word      [Width of KeyDef entries   ]
    Country           : Word      [Language ID               ]
    TableTypeID       : Word      [Identifies the table type ]
    Reserved          : 10 Words  [Reserved.                 ]
  End XHeader
  KeyDef = Rec                    [127 copies of this record.]
    XlateOp = Rec [word field]    [Translate operation specifier.]
      AccentFlags : 7 Bits        [See notes 1 and 8, below.]
      KeyType     : 9 bits        [Note 2, below.]
    Char1 : Byte                  [Use depends on KeyType, below.]
    Char2 : Byte                  [Use depends on KeyType, below.]
    Char3 : Byte                  [Use depends on KeyType, below.]
    Char4 : Byte                  [Use depends on KeyType, below.]
    Char5 : Byte                  [Use depends on KeyType, below.]
  EndRec KeyDef
  AccentTable = Rec               [Table of accent key definitions.]
    AccentEntry1 : AccentEntry
    AccentEntry2 : AccentEntry
        .          .
        .          .
        .          .
    AccentEntry7 : AccentEntry
  EndRec AccentTable


  AccentEntry = Rec     [Accent entry definition. See notes 1 and 9.]
    NonAccent : 2 Bytes [Char/scan code when not used as accent]
    CtlAccent : 2 Bytes [Char/scan code when used with CTL.     ]
    AltAccent : 2 Bytes [Char/scan code when used with ALT.     ]
    Map1   : 2 Bytes   [From char to char for translation. ]
    Map2   : 2 Bytes     "    "        "             "
                         "    "        "             "
        .          .     "    "        "             "
        .          .     "    "        "             "
        .          .
    Map20 : 2 Bytes      "    "        "             "
  EndRec AccentEntry

  TableTypeID
            1st byte     2nd byte
            type         sub-type
            00X          Reserved
  OS/2      01X          00X
```

Notes about the Code Page

1. The AccentFlags field of the KeyDef record has seven flags that are individually set if a corresponding entry in the accent table applies to this scan code. If the key pressed immediately before the current one was an accent key, and the bit for that accent is set in the AccentFlags field for the current key, the corresponding AccentTable entry is searched for the replacement character value to use. If no replacement is found the "not-an-accent" beep is sounded and the accent character and current character are passed as two separate characters. Also see note 8.
2. The KeyType field of the KeyDef record currently has the following values defined. The remaining values up to 1FH are undefined. In the following table the effect of each type of shift is defined. Except where otherwise noted, when no shifts are active, Char1 is the translated character. References to undefined are clarified in note 3 below. Note that either the ALT, ALTC/S/G or both may be present on a keyboard based on the AltGrafL and AltGrafR bits in the XTableFlags1 flagword in the table header.
   - 01H) AlphaKey - Alphabetical character key:

```
SHIFT   - Uses Char2 (If CAPSLOCK, uses Char1).
CAPSLOCK- Uses Char2 (If SHIFT, uses Char1).
CTL     - Set standard "control" code for this key's
          Char1 value (see note 4 below).
ALT     - Standard "extended" code (see note 7).

ALTC/S/G - Uses Char 3 if it is not zero
```

   - 02H) SpecKey - Special non-alphabetical character key, no CAPSLOCK or Alt:

```
SHIFT   - Uses Char2.
CAPSLOCK- No effect, only depends on SHIFT or CTL.
CTL     - See note 4 below.
ALT     - Marked undefined.

ALTC/S/G - Uses Char 3 if it is not zero
```

   - 03H) SpecKeyC - Special non-alpha character key with CAPSLOCK See note 15.

```
SHIFT    - Uses Char2 (If CAPSLOCK, uses Char1).
CAPSLOCK- Uses Char2 (If SHIFT, uses Char1).
CTL      - See note 4 below.
ALT      - Use Char4 if not zero.

ALTC/S/G - Uses Char 3 if it is not zero
```

- 04H) SpecKeyA - Special non-alpha character key, with ALT (no CAPSLOCK):

```
SHIFT    - Uses Char2
CAPSLOCK- No effect, only depends on SHIFT, CTL or ALT.
CTL      - See note 5 and note 9 below.
ALT      - See note 7 below.

ALTC/S/G - Uses Char 3 if it is not zero
```

- 05H SpecKeyCA - Special non-alpha character key, with CAPSLOCK & Alt:

```
SHIFT    - Uses Char2 (If CAPSLOCK, uses Char1).
CAPSLOCK- Uses Char2 (If SHIFT, uses Char1).
CTL      - See note 4 below.
ALT      - See note 7 below.

ALTC/S/G - Uses Char 3 if it is not zero
```

- 06H) FuncKey - Function keys (Char1 = “n” in “Fn”, Char2 ignored, sets “extended” codes 58+Char1 if no shift or if F11 or F12, uses 139 and 140).

```
SHIFT    - Sets "extended" codes 83+Char1;
           F11 and F12 use 141 and 142 respectively
CAPSLOCK- No effect on function keys.
CTL      - Sets "extended" codes 93+Char1;
           F11 and F12 use 143 and 144 respectively
ALT      - Sets "extended" codes 103+Char1;
           F11 and F12 use 145 and 146 respectively.

ALTC/S/G - Uses Char 3 if it is not zero
```

- 07H) PadKey - Keypad keys (see note 5 for definition of Char1, and note that non-shifted use of these keys is fixed to the "extended" codes):

```
SHIFT   - Uses Char2 (Unless NUMLOCK, then see note 5).
CAPSLOCK- No effect on pad keys (NUMLOCK does, note 5).
CTL     - Sets "extended" codes (see note 5).
ALT     - Used to "build" a character (see note 5).

ALTC/S/G - Uses Char 3 if it is not zero
```

- 08H) SpecCtlKey - "Action" keys that do special things with Ctrl down:

```
SHIFT   - No effect on these keys.
CAPSLOCK- No effect on these keys.
CTL     - Uses Char2.
ALT     - Marked undefined.

ALTC/S/G - Uses Char 3 if it is not zero
```

- 09H) PrtSc - Print Screen key (sets Char1 normally):

```
SHIFT   - Signal the Print Screen function.
CAPSLOCK- No effect on this key.
CTL     - Sets extended code and signals
          the Print Echo function.
ALT     - Marked undefined.

ALTC/S/G - Uses Char 3 if it is not zero
```

- 0AH) SysReq - System Request key; treated like a shift key. (See note 6 below).
- 0BH) AccentKey - Keys that affect the "next" key pressed (also known as dead keys). Char1 is an index into the AccentTbl field of the XlateTable, selecting the AccentEntry that corresponds to this key. Char2 and Char3 do the same for the shifted Accent character. See note 15.

```
SHIFT    - Use Char2 to index to applicable AccentEntry.
CAPSLOCK- No effect on this key.
CTL      - Use CtlAccent character from AccentEntry
           (see note 8).
ALT      - Use AltAccent character from AccentEntry
           (see note8).

ALTC/S/G - Use Char3 to index to applicable AccentEntry.
```

Note: Key types 0CH through 13H set Char1 & Char2 to mask values as defined in note 6 below.

- 0CH) ShiftKeys - SHIFT or Ctrl key, sets/clears flags. Char1 holds the bits in the lower byte of the shift status word to set when the key is down and clear when the key is released. Char2 does the same thing for the upper byte of the shift status word, unless the "secondary" key prefix (hex E0) is seen immediately prior to this key, in which case Char3 is used in place of Char2.
- 0DH) ToggleKey - General toggle key (like Caps Lock). Char1 holds the bits in the lower byte of the shift status word to toggle on the first make of the key after it is pressed. Char2 holds the bits in the upper byte of the shift status word to set when the key is down and clear when the key is released unless the "secondary" key prefix (hex E0) is seen immediately prior to this key, in which case Char3 is used in place of Char2.
- 0EH) ALTKey - ALT key. Treated just like ShiftKeys above, but has its own key type because when seen, the accumulator used for ALT-Padkey entry is zeroed to prepare such entry. See note 5 for more information about ALT-PadKey entry. Sometimes this key is treated as "ALTC/S/G" key if one of the AltGraf bits is on in XTableFlags1.
- 0FH) NumLock - NUMLOCK key. Behaves like ToggleKey normally, but the KBDDD will set a pause screen indication when this key is seen along with the Ctrl key depressed. The pause is cleared on the following keystroke, if that stroke is a character generating key.

- 10H) Caps Lock - The Caps Lock key. This key is treated exactly like a type 0DH toggle key. It has a separate entry here so that if it can be processed like a shift lock key when that flag is set in the XTableFlags1 word in the header. When treated as a ShiftLock, the Caps Lock flag in the shift status word is set ON on any make of this key, and only cleared when the left or right shift key is depressed. Char2 and Char3 are processed the same as ToggleKey.

- 11H) ScrollLock - SCROLL LOCK key. Behaves like ToggleKey normally, but has a separate entry here so that when used with "Ctrl-" it can be recognized as "Ctrl-Break".

- 12H) XShiftKey - Extended Shift Key (for Country Support). See note 9 for more information.

- 13H) XToggleKey - Extended Toggle Key (for Country Support). See note 9 for more information.

- 14H) SpecKeyCS - Special key 1 for foreign keyboard processing. See note 15 for more information.

```
SHIFT     - Use Char2
CAPSLOCK  - Use Char4
CTL       - See note 4 below.
ALT       - No effect on this key

ALTC/S/G  - Use Char3

CAPS+SHFT - Use Char5
```

- 15H) SpecKeyAS - Special key 2 for foreign keyboard processing. See note 15 for more information.

```
SHIFT       - Use Char2
CAPSLOCK    - No effect on this key
CTL         - See note 4 below.
ALT         - Use Char4. See note 14 below

ALTC/S/G    - Use Char3. See note 14 below
```

- 16-19H) Reserved

- 20-1FFH) Reserved

3. Undefined Character Code: Any key combination that doesn't fall into any of the defined categories (e.g., the Ctrl key pressed along with a key that has no defined control mapping) will be mapped to the value 0 and the keytype will be set in the KeyPacket record indicating undefined translation. The KeyPacket record passed to the monitors (if any are installed) will contain the original scan code in the ScanCode field and the 0 in the Character field for this key. Note that no chardata recs with an undefined character code will be placed in the keyboard input buffer.
4. Ctrl Key Notes: There are six possible situations for when a key is pressed along with only the Ctrl shift key. They are:
   a. The key pressed is an AlphaKey character. In this case the Ctrl plus Char1 combination defines one of the standard defined control codes. They are (all numbers are decimal):

Ctrl-	Mapping	Code Name	Ctrl-	Mapping	Code Name
a	1	SOH	n	14	SO
b	2	STX	o	15	SI
c	3	ETX	p	16	DLE
d	4	EOT	q	17	DC1
e	5	ENQ	r	18	DC2
f	6	ACK	s	19	DC3
g	7	BEL	t	20	DC4
h	8	BS	u	21	NAK
i	9	HT	v	22	SYN
j	10	LF	w	23	ETB
k	11	VT	x	24	CAN
l	12	FF	y	25	EM
m	13	CR	z	26	SUB

   Note that any key defined as AlphaKey will use the Char1 code value minus 96 (ASCII code for "a") plus 1 to set the mapping shown above. So any scan code defined as AlphaKey must assign to Char1 one of the allowed lower case letters.
   b. The key pressed is a non-alpha character (like "["), but is not an "action" key (like Enter, Backspace, or an arrow key). This is a SpecKey[C][A] in the list of key types above. In this case (with one exception) the mapping is based on the scan code of the key. Though the key may be re-labeled, the Ctrl+Char combination is always mapped based on the scan

code of the key using the following table (all numbers are decimal):

	Scan Code	US Kbd Legend	Mapped Value	Name of New Code	
	3	2 @	0	Null	
	7	6 ^	30	RS	
*	12	- _	31	US	(see note below)
	26	[{	27	Esc	
	27	] }	29	GS	
	43	\ |	28	FS	

Note: The mapping for the hyphen character ("-") is the one exception. The scan code for it is ignored, only the ASCII code for hyphen (decimal 45) is looked for (in Char1) when mapping the Ctrl + "-" combination. This is because there may be more than one occurrence of the "-" key on the keyboard.

c. The key pressed is an "action" key like Enter, backspace, or an arrow key. These keys generate special values when used in conjunction with the Ctrl key. Those actions are defined in other notes where they apply.

d. The key pressed is a function key (F1 - F12).

e. The key pressed is an accent key. See note 8 for details.

f. The key is not defined in conjunction with Ctrl. In that case, the key will treated as undefined, as described in note 3.

5. Pad Key Notes: The pad keys have several uses depending on various shift states. Some of them are based on their position on the keyboard. Because keyboard layouts change, here are the hard coded assumed positions of the keypad keys, with the "offset" value that must be coded into Char1 of the table. Any remapping must use the Char1 values defined below for the keys that correspond to the pad keys given by the Legend or Char2 values shown:

US Kbd Legend	Scan Code	Char1 REQUIRED	Char2 US Kbd
Home 7	71	Binary 0	ASCII 7
Up 8	72	" 1	" 8
PgUp 9	73	" 2	" 9
-	74	" 3	" -
Left 4	75	" 4	" 4
5	76	" 5	" 5
Right 6	77	" 6	" 6
+	78	" 7	" +
End 1	79	" 8	" 1
Down 2	80	" 9	" 2
PgDn 3	81	" 10	" 3
Ins 0	82	" 11	" 0
Del .	83	" 12	" .

Note that when NumLock is OFF, or if Shift is active & NumLock ON, the code returned is the "extended" code. The code returned corresponds to the Legends above (Home, PgUp, etc). When NumLock is ON, or if SHIFT is active & NumLock is OFF, the code returned is Char2. Note that the " + " and "-" keys will return Char2 under ALL shift combinations except Alt (see below).

When the Alt key is used with the PadKeys, the absolute value of the pressed key (looked up using the required Char1 value) is added to the accumulated value of any of the previous numeric keys pressed without releasing the Alt key. Before adding the new number to the accumulated value, that accumulation is multiplied by ten, with overflow beyond 255 ignored. When Alt is released, the accumulation becomes a Character code, and is passed along with a scan code of zero. Note that if any key other than the 10 numeric keys is hit, the accumulated value is reset to zero.

When AltGraphics is used with the PadKeys the Char3 value is returned if it is non-zero and if an AltGraf bit is set in XTableFlags1; otherwise it is treated the same as the Alt key.

6. State Key Notes: Each state key entry has Char1, Char2, and Char3 defined as follows:

Char1 is a mask to set the appropriate bit in the low byte of the keyboard Shift Flags when the state key is pressed. When the state key is a toggle key, the set bit will be toggled each additional time the key is pressed. When the state key is not a toggle key, the set bit will be cleared when the key is released.

Char2 is a mask to set the appropriate bit in the high byte of the Keyboard Shift Flags when the key is pressed.

Char3 is used in place of Char2 when the secondary key prefix is seen immediately prior to this key.

The masks are (numbers are in hex):

Key	Char1	Char2	Char3
Right Shift	01	00	00
Left Shift	02	00	00
Ctrl Shift	04	01	04
Alt Shift	08	02	08
Scroll Lock	10	10	10
Num Lock	20	20	20
Caps Lock	40	40	40
SysReq	00	80	80

Note that the INS key is not treated as a state key, but as a Pad key. Also note that SysReq is included here as it is treated as a shift key.

7. Alt Character Notes: Most of the keys defined in a category that allows the Alt key (AlphaKey, SpecKeyA, SpecKeyCA) return a value called an Extended Character. This value is a character code of 00H or E0H with a second byte (using the ScanCode field of the CharData record) defining the extended code. In most cases this value is the scan code of the key. Since the legend on these keys may be remapped on a foreign language keyboard, the Alt based extended code is hard to define in a general sense. The following rules are used:
   - AlphaKey: The extended code is derived from Char1 (the "lower case" character) as it was originally mapped on the PC keyboard. The original scan code value itself is the extended code that a character will return. These keys can be moved and will still return their original Alt extended codes.
   - SpecKeyA and SpecKeyCA: This category is used for all keys that are not an alphabetical character or an "action" code (like Enter or Backspace, the only exception being the tab key which IS treated as a character). On foreign keyboards these

keys may be moved around and/or have new values assigned to them (such as special punctuation symbols). So the Alt mappings must be based on the real scan code. The effect of this is that keys defined by the SpecKey__ classification will only have an Alt mapping if it is in one of the positions defined below. In that case the Alt extended code is as shown in the table.

Scan Code	US Kbd Legend	ALT Value	Scan Code	US Kbd Legend	ALT Value
2	1 !	120	8	7 &	126
3	2 @	121	9	8 *	127
4	3 #	122	10	9 (	128
5	4 $	123	11	0)	129
6	5 %	124	12	- _	130
7	6 ^	125	13	= +	131

- FuncKey: Defined in note 2.

When AltGraphics is used the Char3 value is returned if it is non-zero and if an AltGraf bit is set in XTableFlags1; otherwise it is treated the same as the Alt key.

8. Accent Key Notes: When an accent key is pressed along with Ctrl or Alt it is treated as a regular key. The character it is translated to is the one in the CtlAccent or AltAccent field of the AccentEntry pointed to by the Char5 value of the KeyDef. If the key being defined will have no defined value with Ctrl or Alt, it should have zeroes in the field of the undefined combination.

   When an accent key is pressed by itself (or with Right or Left Shift or AltGraphics) it will not be translated immediately. The Char1 (or Char2 when Left or Right Shift is used or when the AltGraphics is used) index in the KeyDef record will be used with the next key received to check if the next key has an accent mapping. If that next key has no mapping for this accent (i.e., if it has no bit set in its AccentFlags for this accent, or if that next key is not found in this accent's AccentEntry) then the character value in the NonAccent field of the AccentEntry is used as the character to display, followed by the translation of that next key (i.e., both characters are passed on) after the not-an-accent beep is sounded.

Note that if a key doesn't change when a left or right shift key is held down it should use the same value for Char1 and Char2 so that the accent will apply in both the shifted and non-shifted cases. If the accent value is undefined when used with a shift key or AltGraphics the value in Char2 or Char3 should be zero.

Any accent key that doesn't have an Alt or Ctrl mapping should put zeros in the AltAccent and CtlAccent fields of its AccentEntry. If the value in the table is between 1 and 7 then the key is considered an accent key and further accent key processing is indicated. Reference note 1 for further information.

9. Extended State Key Notes: For special Country support, the Keyboard Device Driver maintains another byte of shift status. Key types 12H and 13H are provided for manipulation of that byte. The other fields of the KeyDef are:
   - Char1: A mask where the bits that are on are those bits that define the field being used for the Char2 value. Only the bits in the NLS shift status byte that correspond to the bits in this byte will be altered by the Char2 value.
   - Char2: For KeyType 12H (Extended Shift), the value to OR into the byte when the make code is seen and who's inverted value is ANDed when the break code is seen. For KeyType 13H (Extended Toggle), the value XORed into the byte on each make code seen (break code ignored).
   - Char3: Use in place of the Char2 when the "secondary" key prefix (hex E0) is seen immediately prior to this key.

   Examples of usage are: Char1/2 can define single shift status bits to set/clear/toggle. Char2 can be a set of coded bits (delineated by Char1) that will be set to a numeric value when the key is hit and cleared to zero when released (or on the next hit if toggle). The whole byte can be set to Char2 when Char1 has all bits on.
10. Space Key Note: The key treated as the space character should have a flag set in its AccentFlags field for each possible accent (i.e., for each defined AccentEntry in the AccentTable). And each AccentEntry should have the SPACE character defined as one of its accented characters, with the translation being to the same value as the accent character itself. The reason for this is that, by definition, an Accent Key followed by the space character maps to the accent character alone. If the table is not set up as just

described, a not-an-accent beep will be sounded whenever the accent key followed by a space is pressed.

Note that the space key is defined as a SpecKeyA (type 4) because its use in conjunction with the Alt key is allowed. In that case it will still return the ASCII space character. It will also return the ASCII space character when used with the Ctrl key.

This works correctly except in the case of the diaresis accent (double-dot) in code page 437. Here, the space is treated as an invalid character and the beep result occurs, with the diaresis represented by double quote. Characters are displayed depending upon the language in effect when the invalid diaresis is encountered. For some languages the character substituted is the double-quote; for others, the character used is the F9h character.

11. KbdType will identify the hardware specific keyboard this table is for. The values and allowable types are the same as specified in the Get Keyboard Type, IOCtl 77H.
12. The DefaultTable flag in XtableFlags1 is used by the KEYB utility in loading code pages when changing from one language to another. It identifies the default code page to KEYB should KEYB not find one or both CODEPAGE= defined code pages.
13. The language IDs used are as follows:

Keyboard Layout Code	***Country***
US	UNITED STATES
UK	UNITED KINGDOM
GR	GERMANY
FR	FRANCE
IT	ITALY
SP	SPAIN
DK	DENMARK
NL	NETHERLANDS
SU	FINLAND
NO	NORWAY
PO	PORTUGAL
SV	SWEDEN
SF	SWISS-FRENCH
SG	SWISS-GERMAN

CF	CANADIAN-FRENCH
BE	BELGIUM
LA	LATIN-AMERICAN SPANISH

14. Keytype 15: When ALT or ALT+SHFT key is pressed both the XlatedChar and XlatedScan in the CharData record will have the same value.
15. If the Charx value is in the range of 1 - 7 then Charx identifies an accent key; otherwise Charx is treated as a valid ASCII character. This does not apply to CTL-Charx sequences.
16. If either the ALTGRF, ALT SHIFT, or ALT CTL are pressed and Char3 is zero, the ALT key will be used to translate to a valid result.
17. Size: The code page described here is of the following dimensions:

```
Xlate Header                 =   40
127 KeyDefs @ 7 Bytes        =  889
7 AccentEntries @ 46 Bytes   =  322
                                ---
                               1251 Bytes
```

Returns

None

Remarks

This request changes the device driver resident code page for the system. This IOCtl updates the ZERO entry of the code page control block.

Category 4 – Function 51H

Purpose
Set Input Mode

Parameter Packet Format

Field	Length
Mode	BYTE

Data Packet Format
None

Where

Mode
is a 1-byte field containing:

Bit	*Meaning*
1xxxxxx1	Shift Report
0xxxxxx0	ASCII mode
1xxxxxxx	BINARY mode

Returns
None

Remarks
This request is used to pass the current input mode to the keyboard device driver. The keyboard device driver will maintain the mode separately for each session. The caller can interrogate the mode using function code 71H. The mode is also returned on every READ CHARACTER DATA RECORD(S) call, function code 74H, and PEEK CHARACTER DATA RECORD call, function code 75H. The default input mode is ASCII. The device driver uses mode when processing CTL functions and reporting the shift state when shift report is set on.

Purpose
Set Interim Character Flags

Parameter Packet Format

Field	Length
Flag	BYTE

Data Packet Format
None

Where

Flag

is a 17-byte field containing flag bits. A bit set = to 1 indicates the state listed below:

Bit	*Meaning*
7	Interim console flag on
6	Reserved = 0
5	Program requested on-the-spot conversion
4	Reserved = 0
3	Reserved = 0
2	Reserved = 0
1	Reserved = 0
0	Reserved = 0

Returns
None

Remarks
This request is used to set the interim character flags maintained by the keyboard device driver. The keyboard device driver will maintain the flags separately for each session. The keyboard device driver passes the interim character flags with each character data record to keyboard monitors.

Category 4 – Function 53H

Purpose

Set Shift State

Parameter Packet Format

Field	Length
Shift State	WORD
NLS	BYTE

Data Packet Format

None

Where

Shift State

is a word field containing shift states.

Word High Byte

Bit	*Meaning*
15	SysReq Key down
14	Caps Lock Key down
13	NumLock Key down
12	ScrollLock Key down
11	Right Alt Key Down
10	Right Ctrl Key Down
9	Left Alt Key Down
8	Left Ctrl Key Down

Word Low Byte

Bit	***Meaning***
7	Insert On
6	Caps Lock On
5	NumLock On
4	ScrollLock On
3	Either Alt Key Down
2	Either Ctrl Key Down
1	Left Shift Key Down
0	Right Shift Key Down

NLS
is a byte field containing NLS shift status.

Returns

None

Remarks

This request is used to set the current shift state for the keyboard. The keyboard device driver maintains the shift state separately for each logical keyboard. Note that this call will override the shift state set by previous shift keystrokes. Also the shift state set by this function code will be overridden by any subsequent shift keystrokes. The shift state is inserted into the character data record that is built for each incoming keystroke.

Category 4 — Function 54H

Purpose

Set Typematic Rate and Delay

Parameter Packet Format

Field	Length
Delay	WORD
Rate	WORD

Data Packet Format

None

Where

Delay
specifies the typematic delay in milliseconds. A value greater than the maximum value defaults to the maximum value.

Rate
specifies the typematic rate in characters per second. A value greater than the maximum value defaults to the maximum value.

Returns

None

Remarks

This request is used to set the keyboard typematic rate and delay to the values specified in the request.

Purpose
Notify of Change of Foreground Session

Parameter Packet Format

Field	Length
Session Number	WORD
Terminate Flag	WORD

Data Packet Format
None

Where

Session Number
is a one word field containing the new foreground session. The session number must fall within the range 0 through 15.

Terminate Flag
is a one word field indicating whether the session is being terminated. A -1 value indicates termination otherwise non-termination is indicated.

Returns
None

Remarks
This request is used to tell the keyboard device driver a new foreground session has been made active. The keyboard device driver will set the shift state of the keyboard to the state that was current when the new session was last active. The keyboard device driver will begin using the keyboard input buffer (KIB) and keystroke monitor chain associated with the new session. This command is restricted and may only be used by the first process that makes the call.

Category 4 — Function 56H

Purpose
Set Session Manager Hot Key

Parameter Packet Format

Field	Length
State	WORD
Make Code	BYTE
Break Code	BYTE
Hot Key ID	WORD

Data Packet Format
None

Where

State

High Byte

Bit	*Meaning*
15	SysReq
14	Caps Lock Key down
13	NumLock Key down
12	ScrollLock Key down
11	Right Alt Key Down
10	Right Ctrl Key Down
9	Left Alt Key Down
8	Left Ctrl Key Down

Low Byte

Bit	*Meaning*
7	Reserved = 0
6	Reserved = 0
5	Reserved = 0
4	Reserved = 0
3	Reserved = 0
2	Reserved = 0
1	Left Shift Key Down
0	Right Shift Key Down

Make Code
is the scan code of the hot key make

Break Code
is the scan code of the hot key break

Hot Key ID
is the hot key ID (Value is set by the caller).

Returns
None

Remarks
This request is used by the session manager to set a list of keyboard hot keys the keyboard device driver will begin looking for. The new hot key is global, that is, it applies to all sessions. Up to 16 hot keys can be defined by the session manager for handling by the keyboard device driver. This IOCtl call will only be successful if done by the process which initially invoked IOCtl 55H (Set Foreground Session). The combination of the shift flags in the first word and the scan codes in the second allow the session manager to set hot key combinations such as Alt+Esc. The hot key is triggered on detection of the scan code for the hot key break.

Note: If a DOS execution environment application has claimed hardware interrupt 9 or interrupt 50, the hot key will be triggered on detection of the break scan code for the required shift key.

A hot key can be redefined by calling this function with the same hot key ID.

Purpose

Set KCB

Parameter Packet Format

Field	Length
KCB Handle	WORD

Data Packet Format

None

Where

KCB Handle
is the handle identifying the logical keyboard's KCB.

Returns

None

Remarks

This request binds the specified logical keyboard (KCB) to the physical keyboard for this session.

Category 4 – Function 58H

Purpose
Set Code Page ID

Parameter Packet Format

Field	Length
Code Page Pointer	DWORD
Code Page ID	WORD
Set to Zero	WORD

Data Packet Format
None

Where

Code Page Pointer
is the selector:offset pointing to the code page.

Code Page ID
is one word containing the current code page ID.

Returns
None

Remarks
Sets the code page used by the current KCB to the code page identified by the input parameter. This IOCtl is callable from the DOS execution environment.

Purpose
Set NLS and Custom Code Page

Parameter Packet Format

Field	Length
Code Page Pointer	DWORD
Code Page Number	WORD
Code Page to Load	WORD
Hot Key ID	WORD

Data Packet Format
None

Where

Code Page Pointer
is the selector:offset pointing to the code page.

Code Page Number
is one word identifying the code page number.

Code Page to Load
is one word identifying the number of the code page to load, 1 or 2. A -1 indicates a custom code page for which the segment containing the custom code page will be locked. This option is not valid for the DOS execution environment.

Hot Key ID
is the hot key ID (Value is set by the caller).

Returns
None

Category 4 – Function 5CH

Remarks

This request is used to install one of two possible code pages into the device driver. This IOCtl will update the number one or two entry of the code page control block; entry zero is the device driver resident code page. Note that this IOCtl is similar to IOCtl 50H, Set Code Page, except that different entries in the code page control block are updated. This IOCtl is callable from the DOS execution environment.

Purpose
Create Keyboard creates a new logical keyboard.

Parameter Packet Format

Field	Length
KCB ID	WORD

Data Packet Format
None

Where

KCB ID
is one word containing a unique value used to identify the new logical keyboard. A zero indicates the default keyboard.

Returns
None

Remarks
None

Category 4 – Function 5EH

Purpose

Destroy Keyboard destroys an existing logical keyboard.

Parameter Packet Format

Field	Length
KCB ID	WORD

Data Packet Format

None

Where

KCB ID

is one word containing a unique value used to identify the new logical keyboard. A zero indicates the default keyboard.

Returns

None

Remarks

None

Purpose
Get Input Mode

Parameter Packet Format
None

Data Packet Format

Field	Length
Mode	BYTE

Where

Mode
is a 1-byte field containing one of the following values:

Bit	*Meaning*
1xxxxxx1	Shift Report
0xxxxxx0	ASCII mode
1xxxxxxx	BINARY mode

Returns
None

Remarks
This request is used to obtain the input mode of the session of the currently active process. The input mode can be set with function code 51H. The input mode is meaningful for Ctrl-C, Ctrl-P, Ctrl-S, Ctrl-Break, Ctrl-ScrollLock, and Ctrl-PrtSc processing only.

Category 4 – Function 72H

Purpose
Get Interim Character Flags

Parameter Packet Format
None

Data Packet Format

Field	Length
Flags	BYTE

Where

Flags

is a 1-byte field containing flag bits. A bit set = to 1 indicates the state listed below:

Bit	*Meaning*
7	Interim console flag on
6	Reserved = 0
5	Program requested on-the-spot conversion
4	Reserved = 0
3	Reserved = 0
2	Reserved = 0
1	Reserved = 0
0	Reserved = 0

Returns
None

Remarks
This request is used to obtain the interim character flags maintained by the keyboard device driver.

Purpose
Get Shift State

Parameter Packet Format
None

Data Packet Format
None

Returns
None

Remarks
This request is used to obtain the shift state of the session of the currently active process. The shift state is set by incoming key strokes and by function code 53H calls.

Refer to function 53H, Set Shift State for the data structure.

Category 4 – Function 74H

Purpose
Read Character Data Record(s)

Parameter Packet Format

Field	Length
Transfer Count	WORD

Data Packet Format
See "KbdCharIn – Read Character, Scan Code" on page 3-2 for the CharData data structure.

Where

Transfer Count
is a one word field containing the record transfer count. The sign bit of this word is set to request one of the following actions:

0	Wait for the requested number of key strokes to become available. The device driver will block the requestor until all requested character data records are available and have been transferred to the caller.
1	Do not wait for the requested number of key strokes to become available. In this case, all characters currently available will be transferred, up to the requested transfer count.

Returns
None

Remarks

This request is used to obtain one or more character data records from the keyboard input buffer (KIB) for the session of the currently active process. Note that if shift report is on then the CharData record may not contain a character but a shift state change in the shift status field.

Category 4 – Function 75H

Purpose
Peek Character Data Record

Parameter Packet Format

Field	Length
Status	WORD

Data Packet Format
See "KbdCharIn – Read Character, Scan Code" on page 3-2 for the CharData data structure.

Where

Status
is a one word field which contains either 0 to indicate no key stroke is available or 1 to indicate that a character data record is being returned. The sign bit is set to either 0 if the current input mode is ASCII or 1 if the input mode is BINARY.

Returns
None

Remarks
This request is used to obtain one character data record from the head of the keyboard input buffer (KIB) of the session for the currently active process. The character data record is not removed from the KIB. Note that if shift report is on then the CharData record may not contain a character but a shift state change in the shift status field.

Purpose

Get Session Manager Hot Key

Parameter Packet Format

Field	Length
Type	WORD

Data Packet Format

None

Where

Type

is a one word subtype indicating the type of information to return:

0 Return the maximum number of hot keys the keyboard device driver can support.

1 Return the number of hot keys currently defined in the system and return the key information for each.

Returned Data Buffer

If parameter list on entry was 1, one or more hot key data structures as described under IOCtl function code 56H (Set Session Manager Hot Key).

Returns

None

Remarks

This request is used to obtain the scan code the keyboard device driver is using as the session manager hot key.

Category 4 – Function 76H

This function should first be called with parameter list subtype = 0 to determine the maximum number of hot keys the device driver can support. The value returned should be used to determine the required size of the data buffer on a subsequent call to return the hot key data structures (parameter list subtype = 1).

Purpose
Get Keyboard Type

Parameter Packet Format
None

Data Packet Format

Field	Length
Keyboard Type	WORD
Reserved = 0	DWORD

Where

Keyboard Type
is a 1-word field containing:

High Byte: Reserved = 0

Low Byte:

Value	*Meaning*
00H	Personal Computer AT keyboard
01H	Enhanced Keyboard
02H to FFH	Reserved = 0

Returns
None

Remarks
This request returns the keyboard type.

Category 4 – Function 78H

Purpose
Get Code Page ID

Parameter Packet Format

Field	Length
Code Page ID	WORD

Data Packet Format
None

Where

Code Page ID
is one word containing the current code page ID. A 0 indicates that PC US 437 is being used; One word reserved and set to 0.

Returns
None

Remarks
This request returns the code page in use by the current KCB. A -1 is returned if a custom code page is installed. This IOCtl is callable from the DOS execution environment.

Purpose

Translate Scan Code to ASCII

Parameter Packet Format

Field	Length
Code Page ID	WORD

Data Packet Format

Field	Length
CharData Record	10 BYTES
KbdDDFlags	WORD
Xlate Flags	WORD
Xlate State1	WORD
Xlate State2	WORD

Where

Code Page ID
is one word containing the current code page ID.

CharData Record
See "KbdCharIn – Read Character, Scan Code" on page 3-2 (10 bytes).

KbdDDFlags
as defined in the Device Monitor packets in OS/2 Technical Reference, Volume 1 Chapter 2

Xlate Flags

High Byte

Value	*Meaning*
8-15	Reserved = 0

Low Byte

Value	*Meaning*
1-7	Reserved = 0
0	Translation complete

Xlate State1 and Xlate State2
identifies the state of translation across successive calls. Initially these words should be 0. They should be reset to 0 when the caller wants a new start to translation. Note that it may take several calls to this IOCtl to complete a character so these fields should not be touched unless a fresh start to translation is desired. These fields are set to 0 at the completion of translation.

Returns
None

Remarks
This request translates a scan code in a CharData record to an ASCII character. Optionally a code page may be specified to use for translation otherwise the code page of the active KCB will be used.

Category 5 Printer Control IOCtl Commands

Following is a summary of Category 5 descriptions:

Function	***Description***
42H	Set frame control (CPL, LPI)
44H	Set infinite retry
45H	Reserved
46H	Initialize printer
48H	Activate font
62H	Get frame control
64H	Get infinite retry
66H	Get printer status
69H	Query active font
6AH	Verify font

Category 5 – Function 42H

Purpose

Set Frame Control

Parameter Packet Format

Field	Length
Command Information	BYTE

Data Packet Format

Field	Length
Characters per Line	BYTE
Lines per Inch	BYTE

Where

Command Information
is reserved and must be set to 0

Characters Per Line
Valid numbers are 80 and 132.

Lines Per Inch
Valid numbers are 6 and 8.

Returns

None

Remarks

None

Purpose

Set Infinite Retry

Parameter Packet Format

Field	Length
Command Information	BYTE

Data Packet Format

Field	Length
Data	BYTE

Where

Command Information
is reserved and must be set to 0

Data
The data is defined as:

0 = Disable Infinite retry
1 = Enable Infinite retry

Returns

None

Remarks

None

Category 5 – Function 46H

Purpose
Initialize Printer

Parameter Packet Format

Field	Length
Command Information	BYTE

Data Packet Format

Field	Length
Set To 0	BYTE

The following fields are defined:

Command Information
is reserved and must be set to 0.

Returns
None

Remarks
None

Purpose

Activate Font

Parameter Packet Format

Field	Length
Command Information	BYTE

ledi process.

Field	Length
Code Page	WORD
Font ID	WORD

Where

Command Information
is reserved and must be set to 0.

Code Page
is the value of the code page to make the currently active code page.

0000H	If the Code Page value and Font ID are specified as 0 (0), set printer to hardware default code page and font.
0001H-FFFFH	Valid code page numbers.

Font ID
is the ID value of the Font to make currently active.

0000H	If the Code Page value and Font ID are specified as 0 (0), set printer to hardware default code page and font. If font Id is 0 and the code page is a valid non-0, then any font is acceptable.

0001H-FFFFH	Valid font ID numbers, font types defined by the font file definitions for download fonts. For cartridge fonts, font IDs are the numbers on the cartridge label and are also entered in the DEVINFO statement for the printer.

Returns

IF AX = 0 NO error
ELSE AX = Error Code

FE02	Code page is not available
FE03	No code page function because spooler not started
FE04	Font ID is not available(verify)
FE09	Error caused by switcher error not by input parameters
FE0A	Error caused by invalid printer name as input
FE0D	Got code page req when CP switcher not initialized
FE0F	PID table full cannot activate another entry
FE13	DASD error reading font file control sequence section
FE15	DASD error reading font file font definition block
FE17	DASD error while writing to temporary spool file
FE18	Disk full error while writing to temporary spool file
FE19	Spool file handle was bad

Remarks

None

Purpose
Return Frame Control

Parameter Packet Format

Field	Length
Command Information	BYTE

Data Packet Format

Field	Length
Characters per Line	BYTE
Lines per Inch	BYTE

Where

Command Information
is reserved and must be set to 0.

Characters Per Line
On return, field is set to 80 or 132.

Lines Per Inch
On return, field is set to 6 or 8.

Returns
None

Remarks
None

Category 5 – Function 64H

Purpose
Return Infinite Retry

Parameter Packet Format

Field	Length
Command Information	BYTE

Data Packet Format

Field	Length
Data	BYTE

Where

Command Information
is reserved and must be set to 0.

Data
On return, Data byte is set:

0 = Infinite retry is disabled.
1 = Infinite retry is enabled.

Returns
None

Remarks
None

Purpose

Return Printer Status

Parameter Packet Format

Field	Length
Command Information	BYTE

Data Packet Format

Field	Length
Data	BYTE

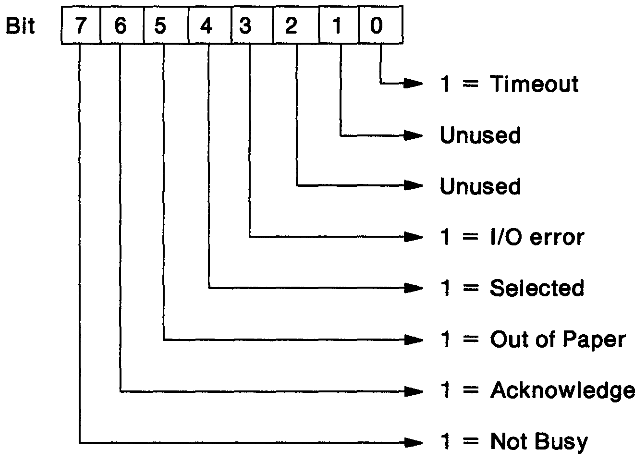
Where

Command Information
is reserved and must be set to 0.

Category 5 – Function 66H

Data

On return, Data byte is set:



Returns

None

Remarks

None

Purpose

Query Active Font

Parameter Packet Format

Field	Length
Command Information	BYTE

Data Packet Format

Field	Length
Code Page	WORD
Font Id	WORD

Where

Command Information
is reserved and must be set to 0.

Code Page
On return, is set to currently active code page.

0000H	if the Code Page value and Font ID are returned as 0 (0), the printer is set to the hardware default code page and font.
0001H-FFFFH	Valid code page numbers.

Font ID
On return, is the ID value of the Font which is currently active.

0000H	if the Code Page value and Font ID are specified as 0 (0), the printer is set to the hardware default code page and font. If font id is 0 and code page is non-0, no error will be returned if any font id is available for the specified code page.

0001H-FFFFH	Valid font id numbers, font types defined by the font file definitions for download fonts. For cartridge fonts, font IDs are the numbers on the cartridge label and are also entered in the DEVINFO statement for the printer.

Returns

```
IF AX = 0
    then NO Error
ELSE
    AX = Error Code
```

FE03	No code page function because spooler not started
FE09	Error caused by switcher error not by input parameters
FE0A	Error caused by invalid printer name as input
FE0D	Got code page req when CP switcher not initialized
FE10	Received request for process ID not in PID table

Remarks

None

Purpose

Verify that a particular code page and font is available for the specified printer

Parameter Packet Format

Field	Length
Command Information	BYTE

Data Packet Format

Field	Length
Code Page	WORD
Font Id	WORD

Where

Command Information
is reserved and must be set to 0.

Code Page
The Code Page number to validate.

Values may be 0 to 65535.

Font ID
The Font ID value to validate.

values may be 0 to 65535. The font ID is contained in the font file for download fonts. For cartridge fonts, font IDs are the numbers on the cartridge label and are also entered in the DEVINFO statement for the printer.

Note: A value of 0 (0) for both the Code Page number and Font Id indicates the default hardware code page and font; this combination is always valid.

Category 5 – Function 6AH

Returns

```
IF AX = 0
      then NO Error
ELSE
      AX = Error Code
```

FE02	Code page is not available
FE03	No code page function because spooler not started
FE04	Font id is not available(verify)
FE0A	Error caused by invalid printer name as input
FE0D	Got code page req when CP switcher not initialized

Remarks

None

Category 7 Mouse Control IOCtl Commands

Following is a summary of Category 7 descriptions:

Function	***Description***
50H	Allows Pointer Drawing after Screen Switch
51H	Notifies of Display Mode Change
52H	Notifies of Impending Session Switch
53H	Reassigns the Current Mouse Scaling Factors
54H	Assigns a New Mouse Event Mask
56H	Sets the Pointer Shape
57H	Frees the Mouse to Draw the Pointer anywhere on the Screen (unmark collision area)
58H	Restricts the Mouse from Pointer Drawing in Specified Area(s) of the Screen (mark collision area)
59H	Specifies/Replaces the Pointer Position
5AH	Specifies the Pointer Draw Device Driver Address (OS/2 mode only)
5BH	Specifies the Pointer Draw Device Driver Address (DOS mode only)
5CH	Specifies a Subset of the Current Mouse Device Driver Status Flags
60H	Indicates the Number of Mouse Buttons Supported by the Device Driver
61H	Indicates the Mouse Setting for the Number of Mickeys/Centimeter
62H	Indicates the Current Pointer Device Driver Status Flags
63H	Reads the Mouse Event Queue
64H	Indicates the Current Event Queue Status
65H	Indicates the Current Mouse Event Mask
66H	Indicates the Current Mouse Scaling Factors
67H	Indicates the Current Pointer Screen Position
68H	Indicates the Current Pointer Shape

Category 7 – Function 50H

Purpose
Allows Pointer Drawing after Session Switch

Parameter Packet Format
None

Data Packet Format
None

Returns
None

Remarks
This function has no input or output parameter values.

This function notifies the mouse device driver that a session switch has been completed and the pointer may now be drawn.

Purpose

Notifies of Display Mode Change

Parameter Packet Format

Field	Length
Length	WORD
Type	BYTE
Color	BYTE
Text Column	WORD
Text Row	WORD
Graphics Column	WORD
Graphics Row	WORD

Data Packet Format

None

Where

Length
is an input parameter to VioSetMode. Length specifies the length of the data structure in bytes including Length itself. The minimum structure size required is 3 bytes, and the maximum structure size required is 12 bytes. Any value specified for Length other than 3 must be an even number. If a structure of length less than the maximum is specified, OS/2 will use default values for the remaining fields.

Type
is a bit mask that contain specifications for the mode being set. The definitions of the bits follow:

```
xxxxxxxb b = 0 monochrome compatible mode
         b = 1 other

xxxxxxbx b = 0 text mode
         b = 1 graphics mode
xxxxxbxx b = 0 enable color burst
         b = 1 disable color burst
```

Color

defines the number of colors as a power of 2. This is equivalent to the number of color bits which define the color. For example,

```
Color = 1 specifies 2 colors
Color = 2 specifies 4 colors
Color = 4 specifies 16 colors
Color = 8 specifies 256 colors

Color = 0 should be specified for
                monochrome modes 7, 7+, and F.
```

Text Column

are the number of text columns.

Text Row

are the number of text rows. 25 rows are supported for the color graphics adapter. Supported for the enhanced graphics adapter are rows 25 and 43. are Supported for the VGA adapter and the IBM Personal System/2 Display Adapter are rows 25 and 50.

Graphics Column

is the number of pel columns.

Graphics Row

is the number of pel rows.

Returns

None

Remarks

When the Video Subsystem or registered Video Subsystem sets/resets the display mode, they must synchronize the mouse device driver pointer update routines by providing this notification record to the mouse device driver prior to switching display modes.

The parameter packet is a far pointer to a Mode Data Definition record.

This call returns no parameter values.

Category 7 – Function 52H

Purpose

Notifies of Impending Session Switch

Parameter Packet Format

Field	Length
Session Number	WORD
Switch Notification Type	WORD

Data Packet Format

None

Where

Session Number

is the session number for notification action. This value must be in the range of 0 < = session number < = MaxNumberOfScreenGroups. The Global InfoSeg defines the valid range session ID's.

Switch Notification Type

is the switch notification type. The values for this parameter follow:

Value	*Meaning*
-1	= The specified session number is terminating.
> = 0	= The specified session number is being switched to.

Returns

None

Remarks

This function sets a system pointer draw enable/disable flag which does not allow pointer drawing until the flag is cleared via a 50H function.

Purpose
Reassigns the Current Mouse Scaling Factors

Parameter Packet Format

Field	Length
Row Data	WORD
Column Data	WORD

Data Packet Format
None

Where

Row Data
Row coordinate scaling factor.

Column Data
Column coordinate scaling factor.

The scaling factor values are positive integers in the range of:

$0 < value <= (32K - 1)$

Returns
None

Remarks
This function requires two 1-word caller-specified parameters.

Scaling factors are ratio values that determine how much relative movement is necessary before the mouse device driver will report a mouse event.

The ratios specify the number of mickeys per 8 pixels. The default ratio values are:

Vertical/Row ration - 16 mickeys per 8 pixels
Horizontal/Row ratio - 8 mickeys per 8 pixels.

Purpose

Assigns a New Mouse Event Mask

Parameter Packet Format

Field	Length
Event Mask	WORD

Data Packet Format

None

Where

Event Mask

Bit	*Meaning*
7-15	Reserved = 0
6	Set if button 3 is down
5	Set if motion with button 3 down
4	Set if button 2 is down
3	Set if motion with button 2 down
2	Set if button 1 is down
1	Set if motion with button 1 down
0	Set if all mouse motion, no buttons

Returns

None

Remarks

This function requires a caller-specified one word parameter containing the new mask for enabled/disabled device events.

Category 7 – Function 56H

Purpose
Sets the Pointer Shape

Parameter Packet Format

Field	Length
Buffer length	WORD
Columns	WORD
Rows	WORD
Column Hot Spot	WORD
Row Hot Spot	WORD

Data Packet Format
The format is dependent on the mode of the display. Refer to the Remarks section in this call.

Where

Buffer length
is the length of pointer image buffer.

Columns
is the width in columns of pointer image.

Rows
is the height in rows of pointer image.

Column Hot Spot
is the column offset within pointer image to hotspot.

Row Hot Spot
is the row offset within pointer image to hotspot.

Returns
None

Remarks

This function requires six caller specified parameters:

Mono & Text

```
Buffer length = (height in characters) *
                 (width in characters) * 2 * 2
              = 1 * 1 * 2 * 2
```

Note: For text mode height and width must be 1, so length is always 4.

Graphics

```
Buffer length = (height in pels) *
                 (width in pels) * (bits per pel) * 2 / 8
```

Note: Width (width in pels) must be a multiple of 8.

Modes 4 & 5 (320 x 200)

```
Buffer length = (height) * (width) * 2 * 2 / 8
```

Mode 6 (640 x 200)

```
Buffer length = (height) * (width) * 1 * 2 / 8
```

Note: Length calculations produce byte boundary buffer sizes.

All of the pointer definition record fields and the pointer shape buffer are validated using the session's mode table values. The parameter values must be the same orientation as the current session display mode:

- graphics = pixel values
- text = character values

The data packet is a far pointer to an area in application storage containing the pointer image buffer.

The pointer image buffer format is dependent on the mode of the display. For currently supported modes the buffer always consists of the "and pointer image data" followed by the "XOR pointer image data" The buffer always describes only one display plane.

The parameter packet is a far pointer to an input pointer definition record.

Category 7 – Function 57H

Purpose
Frees the Mouse to Draw the Pointer anywhere on the Screen

Parameter Packet Format
None

Data Packet Format
None

Returns
None

Remarks
This function checks the pointer position, frees it if necessary, and allows it to draw anywhere on the screen.

Purpose

Restricts the Mouse from Pointer Drawing in Specified Area(s) of the Screen

Parameter Packet Format

Field	Length
Left Row Position	WORD
Left Column Position	WORD
Right Row Position	WORD
Right Column Position	WORD

Data Packet Format

None

Returns

None

Remarks

This function requires one caller specified parameter. This parameter is an address pointing to a 8-byte structured buffer. This buffer defines the collision area that will be protected from being overwritten by system pointer draw operations.

Values must be specified in either character or pixel values, depending on the current mode setting of the display.

The data packet is a far pointer to an area in application storage where a collision area definition record will be read by the mouse device driver.

If the entire screen is specified, this function disables pointer drawing for the session.

Category 7 – Function 59H

Purpose
Specifies/Replaces the Pointer Position

Parameter Packet Format

Field	Length
Row Position	WORD
Column Position	WORD

Data Packet Format
None

Row Position
The new row coordinate pointer screen position.

Column Position
The new column coordinate pointer screen position.

The coordinate values are display mode dependent. Pixel values must be used if the display is in graphics mode. Character position values must be used if the display is in text mode.

Returns

None

Remarks
This function does not override functions 57H and 58H.

If the pointer is directed into a restricted area, it remains invisible until moved out of the area or until the area is freed of restrictions.

The parameter packet is a far pointer to a structure in application storage where the mouse device driver will read coordinate positions.

Purpose

Specifies the Pointer Draw Device Driver Address (OS/2 mode only)

Parameter Packet Format

Field	Length
Pointer Entry	DWORD
Pointer DS	DWORD

Data Packet Format

None

Where

Pointer Entry

Contains two 1-word fields whose format is as follows:

Word 0 = Pointer Draw Rtn Device Driver's Entry Point Offset

Word 1 = Pointer Draw DD Entry Point Selector

Pointer DS

Contains two 1-word fields whose format is as follows:

Word 0 = Reserved = 0

Word 1 = Pointer Draw Rtn Device Driver's Data Segment selector

Returns

None

Remarks

This parameter packet is a far pointer to a structure in application storage where the mouse device driver will read the selector : offset of the entry point of the session's pointer draw routine. The pointer image draw routine is an installed pseudo character device driver. The mouse router/handler must:

- OPEN the pointer draw device driver.
- Query the pointer draw device driver for the address of its entry point.
- Pass the resulting address of the pointer draw entry point to the mouse device driver using the IOCtl described above.

The mouse device driver issues a far call to the pointer draw device driver when ever a mouse interrupt occurs that requires action concerning the pointer image.

In addition, the mouse device driver may call the pointer draw routine as a result of some action on the part of the application, such as:

- MouDrawPtr
- MouRemovePtr
- MouSetPtrPos
- MouSetPtrShape
- MouGetPtrShape

This function is applicable in the OS/2 mode only.

Purpose

Specifies the Pointer Draw Device Driver Address (DOS mode only)

Parameter Packet Format

Field	Length
Pointer Entry	DWORD
Pointer DS	DWORD

Data Packet Format

None

Where

Pointer Entry

Contains two 1-word fields whose format is:

Word 0 =	Pointer Draw Rtn Device Driver's Entry Point Offset
Word 1 =	Pointer Draw Rtn Device Driver's Entry Point Selector

Pointer DS

Entry contains two 1-word fields whose format is:

Word 0 =	Reserved = 0
Word 1 =	Pointer Draw Rtn Device Driver's Data Segment Selector

Returns

None

Category 7 – Function 5BH

Remarks

This IOCtl is for the DOS execution environment only.

This is the same structure passed by the OS/2 IOCtl.

This IOCtl is issued by the Shell/Session Manager at the end of SysInit.

The call passes to the mouse device driver the address of the entry point of a pointer draw routine for DOS execution environment display support.

The data is passed as (selector : offset) pairs. The DOS execution environment portion of the device driver uses the VirtToPhys and PhysToVirt OS/2 calls to convert this address to the (segment : offset) real address for use in the DOS execution environment.

The parameter packet is a far pointer to a structure in application storage where the mouse device driver will read the selector : offset of the entry point of the DOS execution environment session's pointer draw routine.

Purpose

Specifies a Subset of the Current Mouse Device Driver Status Flags

Parameter Packet Format

Field	Length
Status Mask	WORD

Data Packet Format

None

Where

Status Mask

is defined with the bit level definitions as follows:

High Byte

Bit	***Meaning***
7-2	Reserved = 0
1	Set if mouse data returned in mickeys, not display units
0	Set if the interrupt level pointer draw routine is not called

Low Byte

Bit	***Meaning***
7-0	Reserved = 0

A set bit is a value of 1.

Returns

None

Category 7 – Function 5CH

Remarks

This function is the complement to 62H.

The parameter packet is a far pointer to an application area where the Mouse Device Driver will read a one-word input device status mask. Only the high byte of this one-word device status mask may be set.

Purpose

Indicates Number of Buttons Supported by the Device Driver

Parameter Packet Format

None

Data Packet Format

Field	Length
Number Supported	WORD

Where

Number Supported

the data packet is a far pointer to word in application storage where the mouse device driver will write a one-word return value. Return values will be in the range of:

1 = one-button support
2 = two-button support
3 = three-button support

Returns

None

Remarks

This function requires a caller-specified address designating where the device driver can write a one-word return value.

Category 7 – Function 61H

Purpose

Indicates Mouse Setting for the Number of Mickeys/Centimeter

Parameter Packet Format

None

Data Packet Format

Field	Length
Mickeys/Centimeter	WORD

Where

Mickeys/Centimeter

the data packet is a far pointer to a word in application storage where the mouse device driver will write a return value. Return values will be in the range of:

0 < number of mickeys/centimeter <= (32K - 1)

Returns

None

Remarks

None

Purpose

Indicates the Current Pointer Device Driver Status Flags

Parameter Packet Format

None

Data Packet Format

Field	Length
Status Flags	WORD

Where

Status Flags

High Byte

Bit	*Meaning*
7-2	Reserved = 0
1	Set if mouse data returned in mickeys
0	Set if the interrupt level pointer draw routine is not called

Low Byte

Bit	*Meaning*
7-4	Reserved = 0
3	Set if pointer draw routine disabled by unsupported mode
2	Set if flush in progress
1	Set if block read in progress
0	Set if event queue busy with I/O

A set bit is a value of 1.

Returns

None

Category 7 – Function 62H

Remarks

The data packet is a far pointer to a word in application storage where the mouse device driver will write a one-word return value. Return values have the following meaning:

This function is the complement to 5CH.

Purpose

Reads the Mouse Event Queue

Parameter Packet Format

Field	Length
Read Type	WORD

Data Packet Format

Field	Length
Event Mask	WORD
Time	DWORD
Row Position	WORD
Column Position	WORD

Where

Read Type

is only used to determine the type of action to be taken if no event queue data is available. The values may be:

0 = Block the process (Wait) until event data is available

1 = Return a NULL record (No Wait) for the request.

The data packet is a far pointer to an event queue element record structure in application storage to be written into.

Event Mask

(See function 65H)

Time

is Event time stamp in milliseconds

Row Position
is Pointer row coordinate

Column Position
is Pointer column coordinate

Returns
None

Remarks
This function requires two caller-specified parameters:

1. An address where the Mouse Device Driver will write the event queue's FIFO element 10-byte record contents.
2. A one-word value indicating the type of read operation to be performed. The read type is only used to determine the type of action to be taken if no event queue data is available.

Purpose

Indicates the Current Event Queue Status

Parameter Packet Format

None

Data Packet Format

Field	Length
Element Number	WORD
Queue Number	WORD

Where

Element Number

is where the mouse device driver will write the current number of event queue elements. The return value is a one-word value in the range of:

0 < = value < = MaxNumQueueElements.

Queue Number

is where the mouse device driver will write a one-word return value for the MaxNumQueueElements.

Returns

None

Remarks

This function returns both the current number of queued elements in the event queue and the maximum number of elements allowed in an event queue.

Category 7 – Function 65H

Purpose

Indicates the Current Mouse Event Mask

Parameter Packet Format

None

Data Packet Format

Field	Length
Event Mask	WORD

Where

Event Mask

has the following bit level definitions:

Bit	*Meaning*
7-15	Reserved = 0
6	Set if button 3 is down
5	Set if motion with button 3 down
4	Set if button 2 is down
3	Set if motion with button 2 down
2	Set if button 1 is down
1	Set if motion with button 1 down
0	Set if all mouse motion, no buttons

Returns

None

Remarks

This function requires a caller specified address designating where the mouse device driver will write a one-word return value. The return values could be any valid combination of enabled/disabled event flags.

Purpose
Indicates the Current Mouse Scaling Factors

Parameter Packet Format
None

Data Packet Format

Field	Length
Row Data	WORD
Column Data	WORD

Where

Row data
is the row coordinate scaling factor.

Column data
is the column coordinate scaling factor.

The scaling factor values are positive integers in the range of:

$0 < value <= (32K - 1)$

Returns
None

Remarks
This call does not require input parameters. This function requires one caller specified address. The mouse device driver will place a one-word return value at each of the given addresses.

- The first address will receive the row coordinate scaling factor
- The second address will receive the column coordinate scaling factor.

The scaling factor values are positive integers in the range of:

$0 < \text{value} <= (32K - 1)$

The data packet is a far pointer to a two-word structure in application storage where the mouse device driver will write two one-word return values.

Purpose

Indicates the Current Pointer Screen Position

Parameter Packet Format

None

Data Packet Format

Field	Length
Row Position	WORD
Column Position	WORD

Where

Row Position

Row coordinate pointer screen position.

Column Position

Column coordinate pointer screen position.

The coordinate values are display mode dependent. Pixel values are returned if the display is in graphics mode. Character position values are returned if the display is in text mode.

Returns

None

Remarks

This call does not require input parameters. The data packet is a far pointer to a structure in application storage where the mouse device driver will write coordinate positions.

Category 7 – Function 68H

Purpose

Indicates the Current Pointer Shape

Parameter Packet Format

Field	Length
Buffer Length	WORD
Columns	WORD
Rows	WORD
Column Hot Spot	WORD
Row Hot Spot	WORD

Data Packet Format

Field	Length
FAR Pointer	DWORD

Where

Buffer Length

Length of pointer image buffer

Exit Error: if the input Buffer Length value is smaller than the required storage to perform the data copy then the Buffer Length field will be returned with the minimum required pointer shape buffer length. An error code is also returned for an invalid parameter

Exit Normal: if the input Buffer Length value is greater than or equal to the amount of storage required for the pointer shape image then the current pointer information is returned in the pointer data record and the pointer shape image data is copied into the user specified Data address.

Columns
Width in columns of pointer image

Rows
Height in rows of pointer image

Column Hot Spot
Column offset within pointer image to hotspot

Row Hot Spot
Row offset within pointer image to hotspot

Returns

On input the only pointer definition record field used by the mouse device driver is the Buffer Length field. The Buffer Length value must specify the length of the user provided shape buffer, pointed to by the Data parameter address.

Remarks

This buffer is described by the pointer definition record and for normal conditions consists of the Screen AND and Pointer XOR masks.

category 8 Logical Disk Control IOCtl Commands

Following is a summary of Category 8 descriptions:

Function	*Description*
00H	Lock drive
01H	Unlock drive
02H	Redetermine media
03H	Set logical map
20H	Block removable
21H	Get logical map
22H	Reserved
43H	Set device parameters
44H	Write track
45H	Format and verify track
5FH	Reserved
63H	Get device parameters
64H	Read track
65H	Verify track

Purpose

Lock Drive

The operation of locking a drive is used to exclude I/O by another process on the volume in the drive. The Lock Drive call will succeed only if there is one file handle open on the volume in the drive; that is, the file handle on which this DosDevIOCtl call is issued. This is necessary since the desired result is to exclude all other I/O to this volume until the Unlock Drive call is issued.

Parameter Packet Format

Field	Length
Command Information	BYTE

Data Packet Format

Field	Length
Set To 0	BYTE

Where

Command Information
is reserved and must be set to 0.

Returns

None

Remarks

None

Category 8 – Function 01H

Purpose
Unlock Drive

Parameter Packet Format

Field	Length
Command Information	BYTE

Data Packet Format

Field	Length
Set To 0	BYTE

Where

Command Information
is reserved and must be set to 0.

Returns
None

Remarks
The locked volume represented by the file handle on which this DosDevIOCtl call is issued must be in the drive.

Purpose
Redetermine Media -- This function causes OS/2 to re-generate the internal ID for the volume currently in the drive, and associate this ID with the specified handle.

Parameter Packet Format

Field	Length
Command Information	BYTE

Data Packet Format

Field	Length
Set To 0	BYTE

Where

Command Information
is reserved and must be set to 0.

Returns
None

Remarks
The caller must have the disk or diskette locked when calling this function. Otherwise, the call will fail with the error ERROR_LOCK_VIOLATION.

The caller can have only one file open to refer to the disk or diskette. If other processes have the volume open, or the calling process has opened the volume multiple times, the call will fail returning ERROR_DRIVE_BUSY.

Category 8 – Function 03H

Purpose
Set Logical Map

Parameter Packet Format

Field	**Length**
Command Information	BYTE

Data Packet Format

Field	**Length**
Logical Drive Number	BYTE

Where

Command Information
is reserved and must be set to 0.

Logical Drive Number
on entry logical drive number (1=A, 2=B, etc) is specified.

on return, this byte specifies the logical drive currently mapped to the drive that the specified file handle is opened on. A 0 is returned if there is only one logical drive mapped onto this physical drive.

Returns
None

Remarks
None.

Purpose

Block Removable

Parameter Packet Format

Field	Length
Command Information	BYTE

Data Packet Format

Field	Length
Data	BYTE

Where

Command Information
is reserved and must be set to 0.

Data
on return, the data byte is set accordingly:

0 = removable media
1 = nonremovable media

Returns

None

Remarks

None

Category 8 – Function 21H

Purpose
Get Logical Map

Parameter Packet Format

Field	Length
Command Information	BYTE

Data Packet Format

Field	Length
Logical Drive Number	BYTE

Where

Command Information
is reserved and must be set to 0.

Logical Drive Number
on entry Logical Drive Number (1 = A, and others).

on return this byte is filled with the logical drive is currently mapped to the drive the specified handle is opened on. A 0 is returned if there is only one logical drive mapped onto this physical drive.

Returns
None

Remarks
None

Purpose

Set Device Parameters

Parameter Packet Format

Field	Length
Command Information	BYTE

Data Packet Format

Field	Length
Extended BPB for devices	31 BYTES
Number of cylinders	WORD
Device type	BYTE
Device attributes	WORD

Where

Command Information

the two low bits of the command byte are used to indicate one of three possible actions:

Bit Values	*Description*
00	Revert to building the BPB off the medium for all subsequent Build BPB calls. This is used to reset the device parameters back to their original state.
01	Change the default BPB for the physical device. This is not used for the medium and should be used with caution.

10 Change the BPB for the medium to the specified BPB and return the new BPB as the BPB for the medium for all subsequent Build BPB calls. This is used for the initial set device parameters of the medium.

All other bits are reserved and must be set to 0.

Extended BPB
The extended BPB has the following format:

Field	**Length**
Bytes Per Sector	WORD
Sectors Per Cluster	BYTE
Reserved Sectors	WORD
Number of FATs	BYTE
Root Dir Entries	WORD
Total Sectors	WORD
Media Descriptor	BYTE
Sectors Per FAT	WORD
Sectors Per Track	WORD
Number Of Heads	WORD
Hidden Sectors	DWORD
Large Total Sectors	DWORD
Reserved	6 BYTES

Number of cylinders
indicates the number of cylinders defined for the physical device.

Device type
field describes the physical layout of the device specified. It takes one of the following values:

Value	*Meaning*
0	48 TPI low density diskette drive
1	96 TPI high density diskette drive
2	Small (3 1/2 inch) (720KB) drive
3	8 Inch Single Density floppy drive
4	8 Inch Double Density floppy drive
5	Fixed disk
6	Tape drive
7	Other (other type of device)

Device attributes

The device attributes is a bit field that returns various flag information about the specified drive:

Bit	*Description*
0	Removable media flag. If set, the media can be changed
1	Changeline flag. If set, the media cannot be removed.

All other bits are reserved and must be set to 0.

Returns

None

Remarks

None

Category 8 – Functions 44H, 64H, 65H

Purpose

Write Track, Read Track, Verify Track.

These commands have the same parameter packet.

Parameter Packet Format

Field	Length
Command Information	BYTE
Head	WORD
Cylinder	WORD
First Sector	WORD
Number of sectors	WORD
Track Layout field	BYTES

Data Packet Format

The data packet is a buffer. For the write call it contains the data to be written. For a read call the buffer must be large enough to hold requested data. For the verify call the data packet is not used.

Field	Length
Buffer	BYTES

Where

Command Information
is a bit field as follows:

Bit	***Description***
0	Clear - Track layout contains non-consecutive sectors or does not start with sector 1 Set - Track layout starts with sector 1 and contains only consecutive sectors
Reserved	All other bits are reserved and must be set to 0.

Head
is the physical head on the drive to perform the operation.

Cylinder
is the cylinder for the read/write/verify.

First Sector
is the logical sector number within the track layout table to start the I/O operation.

Note that the sector numbers are based from 0. (For example, the third sector is numbered 2.)

Number of Sectors
is the number of sectors to read/write/verify (up to the maximum specified in the track table - the IOCtl subfunctions will not step heads/tracks).

Track layout field
is as follows:

Field	**Length**
Sector number for sector 1	WORD
Sector size for sector 1	WORD
Sector number for sector 2	WORD
Sector size for sector 2	WORD
Sector number for sector 3	WORD
Sector size for sector 3	WORD
...	...
Sector number for sector *n*	WORD
Sector size for sector *n*	WORD

Returns

None

Remarks

This call performs the operation on the device that is specified in this request. The track table passed in on the call is used to determine the sector number which is passed on to the disk controller for the operation. In cases where the sectors are oddly numbered or are non-consecutive, we break this request into N single sector operations and read/write/verify one sector at a time. Note also that the device driver will NOT correctly read a non-512 byte sector if the read operation would generate a DMA violation error. Application writers must be careful to make sure that this error does NOT occur.

The sector table that is specified is used to provide information that is used during the READ/WRITE/VERIFY track operations.

Purpose

Format and Verify a Track on a Drive

Parameter Packet Format

Field	Length
Command Information	BYTE
Head	WORD
Cylinder	WORD
Reserved	WORD
Number of sectors	WORD
Format Track Table	BYTES

Data Packet Format

Field	Length
Set To 0	BYTE

Where

The following fields are defined:

Command Information
is a bit field as follows:

Bit	*Description*
0	Clear - Track layout contains non-consecutive sectors or does not start with sector 1
	Set - Track layout starts with sector 1 and contains only consecutive sectors
	All other bits are reserved and must be set to 0.

Head
is the physical head on the drive to perform the operation.

Cylinder
is the cylinder for the operation.

Number of Sectors
is the number of sectors on the track being formatted.

Format Track Table
is the format track table contains four byte tuples. Each tuple is in the form (c,h,r,n) with c = cylinder number, h = head number, r = sector id, and n = bytes per sector.

n	***bytes / sector***
0	128
1	256
2	512
3	1024

Returns

None

Remarks

This routine formats and verifies the track specified according to the information passed in the Track Layout field. The track layout is passed to the controller and the controller performs the formatting. Note that some controllers do NOT support formatting tracks with varying sector sizes, so in general the application writer must take care to be sure that the sector sizes specified in the Track Layout table are all the same.

There is a 4-tuple for each sector in the track to be formatted. Both the head and cylinder numbers must be consistent within the tuple and with the corresponding parameter packet field.

Purpose
Get Device Parameters

Parameter Packet Format

Field	Length
Command Information	BYTE

Data Packet Format

Field	Length
Extended BPB for device	31 BYTES
Number of cylinders	WORD
Device type	BYTE
Device attributes	WORD

Where
The following fields are defined:

Command Information
is a bit field as follows:

For bit 0:

Value	***Description***
0	Return the recommended BPB for the drive. The recommended BPB for the drive is the BPB for the physical device.
1	Return the BPB for the media currently in the drive

Extended BPB for device
The device driver maintains two BPB's for each drive, one is the current BPB (that corresponds to the media in the drive), and the other is a recommended BPB that is based on the type of media that corresponds to the physical device (for a high density drive,

that is a BPB for a 96 tracks-per-inch (TPI) floppy, for a low density drive it is the BPB for a 48 TPI floppy, etc). The low bit of the command information field indicates which BPB the application would like to see.

Number of cylinders
The number of cylinders indicates the number of cylinders defined for the physical device.

Device type
The device type field describes the physical layout of the device specified. It takes one of the following values:

Value	***Meaning***
0	48 TPI low density diskette drive
1	96 TPI high density diskette drive
2	Small (3 1/2 inch) (720KB) diskette drive
3	8 Inch Single Density diskette drive
4	8 Inch Double Density diskette drive
5	Fixed disk
6	Tape drive
7	Other (other type of device)

Device attributes
The device attributes is a bit field that returns various flag information about the specified drive:

Bit	***Description***
0	Removable media flag If set, the media can be changed
1	Changeline flag If set, the media cannot be removed

Returns
None

Remarks
All other bits are reserved and must be set to 0.

Category 9 Physical Disk Control IOCtl Commands

Category 9 is a category which is used to access physical partitionable fixed disks.

The handle, used for Category 9 commands is the handle returned by the DosPhysicalDisk (function 2) API function call. This handle is used to tell the system which physical disk the IOCtl command is for.

The physical disk control commands relate to the entire partitionable fixed disk. Direct track and sector I/O begin at the beginning of the physical drive. Function 63H, get physical device parameters, describes the entire physical device.

Following is a summary of Category 9 descriptions:

Function	*Description*
00H	Lock physical drive
01H	Unlock physical drive
44H	Physical write track
63H	Get physical device parameters
64H	Physical read track
65H	Physical verify track

Category 9 – Function 00H

Purpose
Lock Physical Drive

Parameter Packet Format

Field	Length
Command Information	BYTE

Data Packet Format

Field	Length
Set To 0	BYTE

Where

Command Information
is reserved and must be set to 0.

Returns
None

Remarks
All the logical units on the physical drive are affected as well.

Purpose
Unlock Physical Drive

Parameter Packet Format

Field	Length
Command Information	BYTE

Data Packet Format

Field	Length
Set To 0	BYTE

Where

Command Information
is reserved and must be set to 0.

Returns
None

Remarks
All the logical units on the physical drive are affected as well.

Category 9 – Functions 44H, 64H, 65H

Purpose

Physical Write Track, Physical Read Track, Physical Verify Track

Parameter Packet Format

These commands have the same parameter packet.

Field	Length
Command Information	BYTE
Head	WORD
Cylinder	WORD
First Sector	WORD
Number of sectors	WORD
Track Layout Table	BYTES

Data Packet Format

The data packet is a buffer. For the WRITE call it contains the data to be written. For a READ call the buffer must be large enough to hold requested data. For the VERIFY call the data packet is not used.

Field	Length
Buffer	BYTES

Where

Command Information
is a bit field as follows:

Bit	*Description*
0	0 (Clear) - Track layout contains non-consecutive sectors or does not start with sector 1

1 (Set) - Track layout start with sector 1 and contains only consecutive sectors

All other bits are reserved and must be 0.

Head
is the physical head on the drive to perform the operation.

Cylinder
is the cylinder for the read/write/verify.

First Sector
is the logical sector number within the track layout table to start the I/O operation.

Note that the sector numbers are based from 0. (For example, the third sector is numbered 2.)

Number of Sectors
The number of sectors to read/write/verify (up to the maximum specified in the track table - the IOCtl subfunctions will NOT step heads/tracks).

Track Layout Table
The track layout table is as follows:

Field	**Length**
Sector number for sector 1	WORD
Sector size for sector 1	WORD
Sector number for sector 2	WORD
Sector size for sector 2	WORD
Sector number for sector 3	WORD
Sector size for sector 3	WORD
...	...
Sector number for sector N	WORD

Field	Length
Sector size for sector N	WORD

Returns
None

Remarks
This call will perform the operation on the physical drive that is specified in this request. It works like the similar Category 8 command except that the I/O is done offset from the beginning of the physical drive instead of from the beginning of the extended volume associated with the unit number (category 8).

The track table passed in on the call is used to determine the sector number which is passed on to the disk controller for the operation. In cases where the sectors are oddly numbered or are non-consecutive the request is broken into N single sector operations, and read/written/verified one sector at a time. Note also that the device driver will not correctly read a non 512 byte sector if the read operation would generate a DMA violation error. Application writers must be careful to make sure that this error will not occur.

The sector table that is specified is used to provide information that is used during the READ/WRITE/VERIFY track operations.

Purpose

Get Physical Device Parameters

Parameter Packet Format

Field	Length
Command Information	BYTE

Data Packet Format

Field	Length
Reserved	WORD
Number of Cylinders	WORD
Number of Heads	WORD
Number of Sectors per Track	WORD
Reserved	WORD
Reserved	WORD
Reserved	WORD
Reserved	WORD

Where

Command Information
is reserved and must be set to 0.

Number of Cylinders
is where the number of cylinders on the physical drive are returned.

Number of Heads
is where the number of heads on the physical drive are returned.

Number of Sectors per Track
is where the number of sectors per track on the physical drive are returned.

Returns

None

Remarks

The data values returned apply to the entire physical disk.

Category 10 Character Device Monitor IOCtl Commands

Following is a summary of Category 10 descriptions:

Function	***Description***
40H	Register a Monitor

Category 10 — Function 40H

Purpose

Register a Monitor

Parameter Packet Format

Field	Length
Command Information	BYTE

Data Packet Format

Field	Length
Placement flag	WORD
Index	WORD
Address of input buffer	DWORD
Offset of output buffer	WORD

Where

Command Information
is reserved and must be set to 0.

Placement flag
is the parameter described in DosMonReg. Values can be 0, 1, or 2.

Index
is a device driver dependent field.

Address of input buffer
is the monitor input buffer initialized by the monitor dispatcher and used by DosMonRead.

Offset of output buffer
is the monitor output buffer initialized by the monitor dispatcher and used by DosMonWrite.

Note: Refer to DosMonRead, DosMonReg and DosMonWrite in Chapter 2, for additional information concerning the device monitor.

Returns
None

Remarks
These fields are used by the device drivers to formulate the MonRegister calls (DevHlp).

Category 11 General Device Control IOCtl Commands

Following is a summary of Category 11 descriptions:

Function	***Description***
01H	Flush input buffer
02H	Flush output buffer
60H	Query monitor support

Purpose
Flush Input Buffer.

Parameter Packet Format

Field	Length
Command Information	BYTE

Data Packet Format

Field	Length
Set To 0	BYTE

Where

Command Information
is reserved and must be set to 0.

Returns
None

Remarks
None

Category 11 – Function 02H

Purpose
Flush output buffer.

Parameter Packet Format

Field	Length
Command Information	BYTE

Data Packet Format

Field	Length
Set To 0	BYTE

Where

Command Information
is reserved and must be set to 0.

Returns
None

Remarks
None

Purpose
Query monitor support.

Parameter Packet Format

Field	Length
Command Information	BYTE

Data Packet Format

Field	Length
Set To 0	BYTE

Where

Command Information
is reserved and must be set to 0.

Returns
None

Remarks
This request is used to query a device driver for monitor support.

The device driver should return the system error, Monitors-Not-Supported, if it does not support character monitors. If monitors are supported, then it should return No-Error (00H).

Appendix A. IBM OS/2 Return Codes

Number/Return Code/Definition

0	NO_ERROR	no error occurred
1	ERROR_INVALID_FUNCTION	invalid function number
2	ERROR_FILE_NOT_FOUND	file not found
3	ERROR_PATH_NOT_FOUND	path not found
4	ERROR_TOO_MANY_OPEN_FILES	too many open files (no handles left)
5	ERROR_ACCESS_DENIED	access denied
6	ERROR_INVALID_HANDLE	invalid handle
7	ERROR_ARENA_TRASHED	memory control blocks destroyed
8	ERROR_NOT_ENOUGH_MEMORY	insufficient memory
9	ERROR_INVALID_BLOCK	invalid memory block address
10	ERROR_BAD_ENVIRONMENT	invalid environment
11	ERROR_BAD_FORMAT	invalid format
12	ERROR_INVALID_ACCESS	invalid access code
13	ERROR_INVALID_DATA	invalid data
14		Reserved
15	ERROR_INVALID_DRIVE	invalid drive was specified
16	ERROR_CURRENT_DIRECTORY	attempt to remove current directory
17	ERROR_NOT_SAME_DEVICE	not same device
18	ERROR_NO_MORE_FILES	no more files

19	ERROR_WRITE_PROTECT attempt to write on write protected diskette
20	ERROR_BAD_UNIT unknown unit
21	ERROR_NOT_READY drive not ready
22	ERROR_BAD_COMMAND unknown command
23	ERROR_CRC data error (CRC)
24	ERROR_BAD_LENGTH bad request structure length
25	ERROR_SEEK seek error
26	ERROR_NOT_DOS_DISK unknown media type
27	ERROR_SECTOR_NOT_FOUND sector not found
28	ERROR_OUT_OF_PAPER printer out of paper
29	ERROR_WRITE_FAULT write fault
30	ERROR_READ_FAULT read fault
31	ERROR_GEN_FAILURE general failure
32	ERROR_SHARING_VIOLATION sharing violation
33	ERROR_LOCK_VIOLATION lock violation
34	ERROR_WRONG_DISK invalid disk change
35	ERROR_FCB_UNAVAILABLE FCB unavailable
36	ERROR_SHARING_BUFFER_EXCEEDED sharing buffer overflow
37-49	Reserved
50	ERROR_NOT_SUPPORTED network request not supported
65	ERROR_NETWORK_ACCESS_DENIED access denied
73-79	Reserved

80	ERROR_FILE_EXISTS file exists
81	ERROR_DUP_FCB Reserved
82	ERROR_CANNOT_MAKE cannot make directory entry
83	ERROR_FAIL_I24 fail on INT 24
84	ERROR_OUT_OF_STRUCTURES too many redirections
85	ERROR_ALREADY_ASSIGNED duplicate redirection
86	ERROR_INVALID_PASSWORD invalid password
87	ERROR_INVALID_PARAMETER invalid parameter
88	ERROR_NET_WRITE_FAULT network device fault
89	ERROR_NO_PROC_SLOTS no process slots available
90	ERROR_NOT_FROZEN system error
91	ERROR_TSTOVFL timer service table overflow
92	ERROR_TSTDUP timer service table duplicate
93	ERROR_NO_ITEMS no items to work on
95	ERROR_INTERRUPT interrupted system call
100	ERROR_TOO_MANY_SEMAPHORES hit user/open semaphore limit exceeded
101	ERROR_EXCL_SEM_ALREADY_OWNED exclusive semaphore already owned
102	ERROR_SEM_IS_SET SemClose found semaphore set
103	ERROR_TOO_MANY_SEM_REQUESTS too many exclusive semaphore requests
104	ERROR_INVALID_AT_INTERRUPT_TIME operation invalid at interrupt time
105	ERROR_SEM_OWNER_DIED previous semaphore owner terminated without freeing semaphore

106	ERROR_SEM_USER_LIMIT semaphore limit exceeded
107	ERROR_DISK_CHANGE insert drive B disk into drive A
108	ERROR_DRIVE_LOCKED drive locked by another process
109	ERROR_BROKEN_PIPE write on pipe with no reader
110	ERROR_OPEN_FAILED open/create failed due to explicit fail command
111	ERROR_BUFFER_OVERFLOW buffer passed to system call too small to hold return data
112	ERROR_DISK_FULL not enough space on the disk
113	ERROR_NO_MORE_SEARCH_HANDLES cannot allocate another search structure and handle
114	ERROR_INVALID_TARGET_HANDLE target handle in DosDupHandle invalid
115	ERROR_PROTECTION_VIOLATION bad user virtual address
116	ERROR_VIOKBD_REQUEST error on display write or keyboard read
117	ERROR_INVALID_CATEGORY category for DevIOCTL not defined
118	ERROR_INVALID_VERIFY_SWITCH invalid value passed for verify flag
119	ERROR_BAD_DRIVER_LEVEL Level four driver not found
120	ERROR_CALL_NOT_IMPLEMENTED invalid function called
121	ERROR_SEM_TIMEOUT time out occurred from semaphore API function
122	ERROR_INSUFFICIENT_BUFFER data buffer too small
123	ERROR_INVALID_NAME illegal character or malformed file system name
124	ERROR_INVALID_LEVEL unimplemented level for information retrieval or setting
125	ERROR_NO_VOLUME_LABEL no volume label found with DosQFsInfo command
126	ERROR_MOD_NOT_FOUND module handle not found with getprocaddr,getmodhandle

127 ERROR_PROC_NOT_FOUND
procedure address not found with getprocaddr

128 ERROR_WAIT_NO_CHILDREN
CWait finds no children

129 ERROR_CHILD_NOT_COMPLETE
CWait children not terminated

130 ERROR_DIRECT_ACCESS_HANDLE
handle operation invalid for direct disk access handles

131 ERROR_NEGATIVE_SEEK
attempted seek to negative offset

132 ERROR_SEEK_ON_DEVICE
application tried to seek on device or pipe

133 ERROR_IS_JOIN_TARGET
drive has previously joined drives

134 ERROR_IS_JOINED
drive is already joined

135 ERROR_IS_SUBSTED
drive is already substituted

136 ERROR_NOT_JOINED
cannot delete drive that is not joined

137 ERROR_NOT_SUBSTED
cannot delete drive that is not substituted

138 ERROR_JOIN_TO_JOIN
cannot join to a joined drive

139 ERROR_SUBST_TO_SUBST
cannot substitute to a substituted drive

140 ERROR_JOIN_TO_SUBST
cannot join to a substituted drive

141 ERROR_SUBST_TO_JOIN
cannot substitute to a joined drive

142 ERROR_BUSY_DRIVE
specified drive is busy

143 ERROR_SAME_DRIVE
cannot join or substitute a drive to a directory on the same drive

144 ERROR_DIR_NOT_ROOT
directory must be a subdirectory of the root

145 ERROR_DIR_NOT_EMPTY
directory must be empty to use join command

146 ERROR_IS_SUBST_PATH
path specified is being used in a substitute

147 ERROR_IS_JOIN_PATH
path specified is being used in join

148	ERROR_PATH_BUSY path specified is being used by another process
149	ERROR_IS_SUBST_TARGET cannot join or substitute drive having directory that is target of a previous substitute
150	ERROR_SYSTEM_TRACE system trace error
151	ERROR_INVALID_EVENT_COUNT DosMuxSemWait errors
152	ERROR_TOO_MANY_MUXWAITERS system limit of 100 entries was reached
153	ERROR_INVALID_LIST_FORMAT invalid list format
154	ERROR_LABEL_TOO_LONG volume label too big
155	ERROR_TOO_MANY_TCBS cannot create another TCB
156	ERROR_SIGNAL_REFUSED Signal refused
157	ERROR_DISCARDED segment is discarded
158	ERROR_NOT_LOCKED segment was not locked
159	ERROR_BAD_THREADID_ADDR bad thread id address
160	ERROR_BAD_ARGUMENTS bad environment pointer
161	ERROR_BAD_PATHNAME bad pathname passed to exec
162	ERROR_SIGNAL_PENDING signal already pending
163	ERROR_UNCERTAIN_MEDIA ERROR_I24 mapping
164	ERROR_MAX_THRDS_REACHED No more proc slots
165	ERROR_MONITORS_NOT_SUPPORTED ERROR_I24 mapping
180	ERROR_INVALID_SEGMENT_NUMBER invalid segment number
181	ERROR_INVALID_CALLGATE invalid call gate
182	ERROR_INVALID_ORDINAL invalid ordinal

183 ERROR_ALREADY_EXISTS
shared segment already exists

184 ERROR_NO_CHILD_PROCESS
no child process to wait for

185 ERROR_CHILD_ALIVE_NOWAIT
NoWait specified & child alive

186 ERROR_INVALID_FLAG_NUMBER
invalid flag number

187 ERROR_SEM_NOT_FOUND
semaphore does not exist

188 ERROR_INVALID_STARTING_CODESEG
invalid starting code segment, incorrect END (label)directive

189 ERROR_INVALID_STACKSEG
invalid stack segment

190 ERROR_INVALID_MODULETYPE
invalid module type - Dynamic link library file cannot be used as an application. Application cannot be used as a dynamic link library.

191 ERROR_INVALID_EXE_SIGNATURE
invalid EXE signature - File is DOS mode program or improper program.

192 ERROR_EXE_MARKED_INVALID
EXE marked invalid - LINK detected errors when application created.

193 ERROR_BAD_EXE_FORMAT
bad EXE format - File is DOS mode program or improper program.

194 ERROR_ITERATED_DATA_EXCEEDS_64k
iterated data exceeds 64K - More than 64K of data in one of the segments of the file.

195 ERROR_INVALID_MINALLOCSIZE
invalid minimum allocation size - Size is specified to be less than the size of the segment data in the file.

196 ERROR_DYNLINK_FROM_INVALID_RING
dynamic link from invalid ring - Ring 2 routine cannot link to dynalink libraries.

197 ERROR_IOPL_NOT_ENABLED
IOPL not enabled - IOPL set to NO in CONFIG.SYS.

198 ERROR_INVALID_SEGDPL
invalid segment descriptor privilege level - can only have privilege levels of 2 and 3

199 ERROR_AUTODATASEG_EXCEEDS_64k
automatic data segment exceeds 64K

200 ERROR_RING2SEG_MUST_BE_MOVABLE
ring 2 segment must be movable

201 ERROR_RELOC_CHAIN_XEEDS_SEGLIM
relocation chain exceeds segment limit

202 ERROR_INFLOOP_IN_RELOC_CHAIN
infinite loop in relocation chain segment

203 ERROR_ENVVAR_NOT_FOUND
environment variable not found

204 ERROR_NOT_CURRENT_CTRY
not current country

205 ERROR_NO_SIGNAL_SENT
no signal sent - No process in the command subtree has a signal handler.

206 ERROR_FILENAME_EXCED_RANGE
filename/extension greater than 8.3

207 ERROR_RING2_STACK_IN_USE
ring 2 stack in use

208 ERROR_META_EXPANSION_TOO_LONG
meta expansion is too long

209 ERROR_INVALID_SIGNAL_NUMBER
invalid signal number

210 ERROR_THREAD_1_INACTIVE
inactive thread

211 ERROR_INFO_NOT_AVAILABLE
filesystem information not available for this file

212 ERROR_LOCKED
locked error

213 ERROR_BAD_DYNALINK
attempted to execute non family API in DOS mode

214 ERROR_TOO_MANY_MODULES
too many modules

215 ERROR_NESTING_NOT_ALLOWED
nesting not allowed

303 ERROR_INVALID_PROCID
invalid process ID

304 ERROR_INVALID_PDELTA
invalid priority delta

305 ERROR_NOT_DESCENDANT
not descendant

306 ERROR_NOT_SESSION_MANAGER
requestor not session manager

307 ERROR_INVALID_PCLASS
invalid P class

308 ERROR_INVALID_SCOPE
invalid scope

309 ERROR_INVALID_THREADID
invalid thread id

310 ERROR_DOSSUB_SHRINK
cannot shrink segment - DosSubSet

311 ERROR_DOSSUB_NOMEM
no memory to satisfy request - DosSubAlloc

312 ERROR_DOSSUB_OVERLAP
overlap of specified block with an allocated memory - DosSubFree

313 ERROR_DOSSUB_:BADSIZE
bad size parameter - DosSubAlloc or DosSubFree

314 ERROR_DOSSUB_BADFLAG
bad flag parameter - DosSubSet

315 ERROR_DOSSUB_BADSELECTOR
invalid segment selector

316 ERROR_MR_MSG_TOO_LONG
message too long for buffer

317 ERROR_MR_MID_NOT_FOUND
message ID number not found

318 ERROR_MR_UN_ACC_MSGF
unable to access message file

319 ERROR_MR_INV_MSFG_FORMAT
invalid message file format

320 ERROR_MR_INV_IVCOUNT
invalid insertion variable count

321 ERROR_MR_UN_PERFORM
unable to perform function

322 ERROR_TS_WAKEUP
unable to wake up

323 ERROR_TS_SEMHANDLE
invalid system semaphore

324 ERROR_TS_NOTIMER
no timers available

326 ERROR_TS_HANDLE
invalid timer handle

327 ERROR_TS_DATETIME
date or time invalid

328 ERROR_SYS_INTERNAL
internal system error

329 ERROR_QUE_CURRENT_NAME
current queue name does not exist

330 ERROR_QUE_PROC_NOT_OWNED
current process does not own queue

331 ERROR_QUE_PROC_OWNED
current process owns queue

332 ERROR_QUE_DUPLICATE
duplicate queue name

333 ERROR_QUE_ELEMENT_NOT_EXIST
queue element does not exist

334 ERROR_QUE_NO_MEMORY
inadequate queue memory

335 ERROR_QUE_INVALID_NAME
invalid queue name

336 ERROR_QUE_INVALID_PRIORITY
invalid queue priority parameter

337 ERROR_QUE_INVALID_HANDLE
invalid queue handle

338 ERROR_QUE_LINK_NOT_FOUND
queue link not found

339 ERROR_QUE_MEMORY_ERROR
queue memory error

340 ERROR_QUE_PREV_AT_END
previous queue element was at end of queue

341 ERROR_QUE_PROC_NO_ACCESS
process does not have access to queues

342 ERROR_QUE_EMPTY
queue is empty

343 ERROR_QUE_NAME_NOT_EXIST
queue name does not exist

344 ERROR_QUE_NOT_INITIALIZED
queues not initialized

345 ERROR_QUE_UNABLE_TO_ACCESS
unable to access queues

346 ERROR_QUE_UNABLE_TO_ADD
unable to add new queue

347 ERROR_QUE_UNABLE_TO_INIT
unable to initialize queues

349 ERROR_VIO_INVALID_MASK
invalid function replaced

350 ERROR_VIO_PTR
invalid pointer to parameter

355 ERROR_VIO_MODE
unsupported screen mode

356	ERROR_VIO_WIDTH invalid cursor width value
358	ERROR_VIO_ROW invalid row value
359	ERROR_VIO_COL invalid column value
366	ERROR_VIO_WAIT_FLAG invalid wait flag setting
367	ERROR_VIO_UNLOCK screen not previously locked
369	ERROR_SMG_INVALID_SESSION_ID invalid session ID
370	ERROR_SMG_NO_SESSIONS no sessions available
371	ERROR_SMG_SESSION_NOT_FOUND session not found
372	ERROR_SMG_SET_TITLE title sent by shell or parent cannot be changed
373	ERROR_KBD_PARAMETER invalid parameter to Kbd
375	ERROR_KBD_INVALID_IOWAIT invalid I/O wait specified
376	ERROR_KBD_INVALID_LENGTH invalid length for keyboard
377	ERROR_KBD_INVALID_ECHO_MASK invalid echo mode mask
378	ERROR_KBD_INVALID_INPUT_MASK invalid input mode mask
379	ERROR_MON_INVALID_PARMS invalid parameters to DosMon
380	ERROR_MON_INVALID_DEVNAME invalid device name string
381	ERROR_MON_INVALID_HANDLE invalid device handle
382	ERROR_MON_BUFFER_TOO_SMALL buffer too small
383	ERROR_MON_BUFFER_EMPTY buffer is empty
384	ERROR_MON_DATA_TOO_LARGE data record too large
386	ERROR_MOUSE_INV_HANDLE Mouse device closed (invalid device handle)

389 ERROR_MOUSE_DISPLAY_PARMS
parameters invalid for display mode

391 ERROR_MOUSE_INV_ENTRY_PT
entry point not valid

392 ERROR_MOUSE_INV_MASK
function mask invalid

394 NO_ERROR_MOUSE_P TR_DRAWN
pointer drawn

395 ERROR_INVALID_FREQUENCY
invalid frequency for beep

396 ERROR_NLS_NO_COUNTRY_FILE
can't find country.sys file

397 ERROR_NLS_OPEN_FAILED
can't open country.sys file

398 ERROR_NO_COUNTRY_OR_CODEPAGE
country code not found

399 ERROR_NLS_TABLE_TRUNCATED
table returned information truncated, buffer too small

400 ERROR_NLS_BAD_TYPE
selected type does not exist

401 ERROR_NLS_TYPE_NOT_FOUND
selected type not in file

402 ERROR_VIO_SMG_ONLY
valid from session manager only

403 ERROR_VIO_INVALID_ASCIIZ
invalid ASCIIZ length

404 ERROR_VIO_DEREGISTER
VioDeRegister not allowed

405 ERROR_VIO_NO_POPUP
popup not allocated

406 ERROR_VIO_EXISTING_POPUP
popup on screen (no wait)

407 ERROR_KBD_SMG_ONLY
valid from session manager only

408 ERROR_KBD_INVALID_ASCIIZ
invalid ASCIIZ length

409 ERROR_KBD_INVALID_MASK
invalid replacement mask

410 ERROR_KBD_REGISTER
KbdRegister not allowed

411 ERROR_KBD_DEREGISTER
KbdDeRegister not allowed

412 ERROR_MOUSE_SMG_ONLY
valid from session manager only

413 ERROR_MOUSE_INVALID_ASCIIZ
invalid ASCIIZ length

414 ERROR_MOUSE_INVALID_MASK
invalid replacement mask

415 ERROR_MOUSE_REGISTER
Mouse register not allowed

416 ERROR_MOUSE_DEREGISTER
Mouse deregister not allowed

417 ERROR_SMG_BAD_ACTION
invalid action specified

418 ERROR_SMG_INVALID_CALL
INIT called more than once

419 ERROR_SCS_SG_NOT_FOUND
new screen group number

420 ERROR_SCS_NOT_SHELL
caller is not shell

421 ERROR_VIO_INVALID_PARMS
invalid parms passed

422 ERROR_VIO_FUNCTION_OWNED
save/restore already owned

423 ERROR_VIO_RETURN
non-destruct return (undo)

425 ERROR_SCS_NOT_SESSION_MGR
caller not session manager

426 ERROR_VIO_REGISTER
Vio register not allowed

427 ERROR_VIO_NO_MODE_THREAD
no mode restore thread in SG

428 ERROR_VIO_NO_SAVE_RESTORE_THD
no save/rest thread in SG

429 ERROR_VIO_IN_BG
function invalid in background

430 ERROR_VIO_ILLEGAL_DURING_POPUP
function not allowed during popup

431 ERROR_SMG_NOT_BASESHELL
caller is not the base shell

432 ERROR_SMG_BAD_STATUSREQ
invalid status requested

433 ERROR_QUE_INVALID_WAIT
nowait parameter out of bounds

434 ERROR_VIO_LOCK
error returned from scrlock
435 ERROR_MOUSE_INVALID_IOWAIT
invalid parameters for IOWait
436 ERROR_VIO_INVALID_HANDLE
invalid vio handle
438 ERROR_VIO_INVALID_LENGTH
invalid vio length
439 ERROR_KBD_INVALID_HANDLE
invalid kbd handle
440 ERROR_KBD_NO_MORE_HANDLE
ran out of handles
441 ERROR_KBD_CANNOT_CREATE_KCB
unable to create kcb
442 ERROR_KBD_CODEPAGE_LOAD_INCOMPL
unsuccessful codepage load
443 ERROR_KBD_INVALID_CODEPAGE_ID
invalid codepage id
444 ERROR_KBD_NO_CODEPAGE_SUPPORT
no codepage support
445 ERROR_KBD_FOCUS_REQUIRED
keyboard focus required
446 ERROR_KBD_FOCUS_ALREADY_ACTIVE
calling thread has an outstanding focus
447 ERROR_KBD_KEYBOARD_BUSY
keyboard busy
448 ERROR_KBD_INVALID_CODEPAGE
invalid codepage
449 ERROR_KBD_UNABLE_TO_FOCUS
focus attempt failed
450 ERROR_SMG_SESSION_NON_SELECT
session is not selectable
451 ERROR_SMG_SESSION_NOT_FOREGRND
parent/child session not foreground
452 ERROR_SMG_SESSION_NOT_PARENT
not parent of requested child
453 ERROR_SMG_INVALID_START_MODE
invalid session start mode
454 ERROR_SMG_INVALID_RELATED_OPT
invalid session start related option
455 ERROR_SMG_INVALID_BOND_OPTION
invalid session bond option

456	ERROR_SMG_INVALID_SELECT_OPT invalid session select option
457	ERROR_SMG_START_IN_BACKGROUND session started in background
458	ERROR_SMG_INVALID_STOP_OPTION invalid session stop option
459	ERROR_SMG_BAD_RESERVE reserved parameters not zero
460	ERROR_SMG_PROCESS_NOT_PARENT session parent process already exists
461	ERROR_SMG_INVALID_DATA_LENGTH invalid data length
462	ERROR_SMG_NOT_BOUND parent not bound
463	ERROR_SMG_RETRY_SUB_ALLOC retry request block allocation
464	ERROR_KBD_DETACHED this call disallowed for detached pid
465	ERROR_VIO_DETACHED this call disallowed for detached pid
466	ERROR_MOU_DETACHED this call disallowed for detached pid
467	ERROR_VIO_FONT no font available to support mode
468	ERROR_VIO_USER_FONT user font active
469	ERROR_VIO_BAD_CP invalid code page specified
470	ERROR_VIO_NO_CP system displays don't support code page
471	ERROR_VIO_NA_CP current display does not support code page
472	ERROR_INVALID_CODE_PAGE invalid code page
473	ERROR_CPLIST_TOO_SMALL code page list is too small
474	ERROR_CP_NOT_MOVED code page not moved
475	ERROR_MODE_SWITCH_INIT mode switch init error
476	ERROR_CODE_PAGE_NOT_FOUND code page not found

477 ERROR_UNEXPECTED_SLOT_RETURNED
internal error

478 ERROR_SMG_INVALID_TRACE_OPTION
invalid start session trace indicator

479 ERROR_VIO_INTERNAL_RESOURCE
vio internal resource error

480 ERROR_VIO_SHELL_INIT
vio shell init error

481 ERROR_SMG_NO_HARD_ERRORS
no session manager hard errors

482 ERROR_CP_SWITCH_INCOMPLETE
DosSetCp unable to set kbd/vio cp

483 ERROR_VIO_TRANSPARENT_POPUP
error during vio popup

484 ERROR_CRITSEC_OVERFLOW
critical section overflow

485 ERROR_CRITSEC_UNDERFLOW
critical section underflow

486 ERROR_VIO_BAD_RESERVE
reserved parameter is not zero

487 ERROR_INVALID_ADDRESS
bad physical address

488 ERROR_ZERO_SELECTORS_REQUESTED
at least one selector must be requested

489 ERROR_NOT_ENOUGH_SELECTORS_AVA
not enough GDT selectors to satisfy request

490 ERROR_INVALID_SELECTOR
not a GDT selector

Index