Chapter 4
NIOS Client DOS APIs

Global Variables

UINT8 DosSwitcherActive

#include <tasksw.h>

Global variable set to a non-zero value when a DOS task switcher is active in the system. Possible values are:

0	No Task Switcher Active
1	Task Switcher Is Active

UINT8 DosWinDebFlag

#include <dosvmm.h>

Global variable set to a non-zero value if MS Windows is active and a debugger compatible with MS Windows is loaded.

UINT8 DosWinDebKernelFlag

#include <dosvmm.h>

Global variable set to a non-zero value if Windows is active and the debug version of the Windows VMM is loaded.

UINT8 DosWinStandardMode

#include <tasksw.h>

Global variable set to a value of 1 if standard-mode Windows is active. The value is zero otherwise.

UINT8 DosWinFlag

#include <dosvmm.h>

Global variable set to a nonzero value when enhanced-mode MS Windows is active. Possible values are:

```
DOSWINFLAG_DOS_ONLY       equ  0  ; Only DOS is active
DOSWINFLAG_WIN_31x        equ  1  ; Windows v3.1x is active
DOSWINFLAG_WIN_4x      equ  2  ; Windows v4.x is active
```

UINT32 DosVmIdToVmCbTable[MAX_NUM_VM]

#include <dosvmm.h>

Global table that can be used to translate a VM ID into its associated VM Control Block value.

Since an NLM should not save a VM's control block address long term, it must instead save the VM's Id. This table assists in translating the VM Id into the VM's control block.

0xFFFFFFFF is returned for entries that are *not* in use.

DosAllocV86Callback

Description

DosAllocV86Callback allocates to the caller a V86 callback address. This seg:off address can be used by V86 code to make requests to Ring-0 modules.

Syntax

```
#include <dosvmm.h>

UINT32
DosAllocV86Callback
    (modHandle      moduleHandle,
     UINT32         referenceData,
     void           (*handler)(void));
```

Parameters

moduleHandle	Caller's module handle.
referenceData	Value passed to callback handler when the V86 callback is invoked.
handler	Pointer to Ring-0 procedure that is invoked when the allocated V86 callback is called. **The handler must execute (simulate) a V86 retf instruction before returning from its handler.**

Entry and exit conditions for this handler are:

```
On entry:
    ebx     -> VM CB
    ebp     -> CRS
    edx     reference data
    CLD has been executed
    Interrupts are disabled in DOS only
    Interrupts are enabled if MS Windows is active

On return:
    CLD preserved
    All registers can be destroyed
    Interrupts in any state
```

Returns

!0	V86 Seg:Off address
0	All V86 callback resources are in use

Remarks

The value passed in *referenceData* can be anything the caller desires, or can be ignored if not needed.

See Also

DosFreeV86Callback
DosExecuteFarRet
DosFastExecuteFarRet

DosAMapFlat

Description

DosAMapFlat converts a *selector:offset* to its flat linear address.

Assumes

eax High word = Selector
Low word = Offset

Returns

eax Linear address
All other registers are preserved

Remarks

The selector can be either an LDT or a GDT. Passed-in LDT selectors must exist in the active LDT.

See Also

DosCMapFlat

DosBeginNestExec

Description

This function allocates a new *ClientRegStruc* from the current stack and initializes it for nested execution.

Assumes

Interrupt state undefined

Returns

ebp Pointer to CRS to use during nested execution
eax Destroyed
All other registers preserved
Interrupts same as entered

Remarks

This function is callable at interrupt time if MS Windows is NOT running. V86 code is NOT callable at interrupt time under Windows.

This service is designed to be called from assembly code. Use **DosBeginNestExecWithCrs** if calling from 'C' code.

NLMs that wish to call code in V86 mode usually use the **DosExecuteV86FarCall** or **DosExecuteV86Int** services. These services require the NLM to enter a nested execution block prior to calling them.

The caller must not make any assumptions about where the returned CRS is actually located.

Note: The caller must explicitly set the interrupt enable flag to the desired value in the new CrsEFlags field prior to invoking V86 code, since the new CrsEFlags are inherited from the active CrsEFlags at the time this function is invoked.

See Also

DosEndNestExec
DosBeginNestExecWithCrs
DosEndNestExecWithCrs

DosBeginNestExecWithCrs

Description

This function initializes the passed-in CRS and prepares the system for nested V86 execution.

Syntax

```
#include <dosvmm.h>

CRS
*DosBeginNestExecWithCrs(
    CRS     *saveCrsBuf);
```

Parameters

saveCrsBug	Pointer to a CRS structure used to preserve the current CRS information. The caller should not interpret any information in this structure after invoking this service.

Returns

Pointer to CRS to use during nested execution. All register values except CrsSS,CrsSP are undefined in this structure.

Remarks

This function is callable at interrupt time if MS Windows isn't running.

The returned CRS pointer may be different than the passed in CRS structure; therefore the caller must use the returned value when setting up the registers for V86 execution.

If the caller needs the original CRS register values during the nested execution block, it should preserve the active CRS register values into a work buffer prior to invoking this service.

See Also

DosEndNestExecWithCrs
DosBeginNestExec
DosEndNestExec

DosBeginReentrantExec

Description

Used in special cases where an NLM needs to invoke a service that normally isn't callable at hardware interrupt time.

Syntax

```
#include <dosvmm.h>

UNIT32
DosBeginReentrantExec(
        void);
```

Parameters

None

Returns

Previous reentrancy level.

Remarks

Although this service will allow a foreground only function to be called, it may causes corruption or other problems since the routine being called may be reentered.

See Also

DosEndReentrantExec

DosBeginUseDos

Description

This function hooks Control-C, Control-Break, and Interrupt 24h vectors in the current VM with handlers that effectively cause these interrupts to be ignored.

Syntax

```
#include <dosvmm.h>

void
DosBeginUseDos(
    UINT32      *savedVectInfo);
```

Parameters

savedVectInfo	Pointer to an array of three UINT32's that will be used to save the current INT 1Bh, INT 23h, and INT 24h vector information. The caller must pass this same buffer to DosEndUseDos when finished with the execution block.

Returns

Nothing
Interrupts same as entered

Remarks

DosBeginUseDos is usually used to bracket code that will be calling DOS using **DosExecuteV86Int** for which the caller wants to protect itself from abort conditions. Users of the DosCall, DosCallC, and/or DosCallUseCurrSDA services do not need to use this function.

See Also

DosEndUseDos

DosCancelDosAvailEvent

Description

Cancels a "DOS available event" that was previously scheduled using *DosScheduleDosAvailEvent*.

Syntax

```
#include <dosvmm.h>

UINT32
DosCancelDosAvailEvent(
    FEB      *eventBlock);
```

Parameters

eventBlock	Pointer to a FEBStruc that was passed to *DosScheduleDosAvailEvent*

Returns

0	Event was cancelled successfully
!0	Event is NOT currently scheduled

Remarks

See Also

DosCall

Description

DosCall executes the specified DOS function.

Assumes

ebp Points to CRS with regs set appropriate for DOS function
Interrupts in any state

Returns

Carry flag equal to CRS carry flag
ebp Points to CRS
All registers preserved
Interrupts same as entered

Remarks

Generally it is easier to use this function than executing the Int 21h using the **DosExecuteV86Int** function, since this function takes care of many of the issues related to calling DOS from an NLM, such as setting up NIOS's PSP, failing critical errors, and disabling Control-Break.

This function cannot be used if DOS is busy, unless the caller has preserved DOS's data areas prior to invoking this function. Because DOS is never busy inside an NLM's initialization and unload function, this function can be used without restriction in these two cases.

The caller must set up a nested execution **ClientRegStruc** prior to calling this function. Also, the caller must set the CRS register values appropriately for the DOS call that will be invoked.

This function yields control.

See Also

DosCallC
DosCallUseCurrSDA

DosCallC

Description

"C" wrapper for the DosCall service.

Syntax

```
#include <dosvmm.h>

UINT32
DosCallC(
      void);
```

Returns

0	Carry flag is 0 (DOS function successful)
!0	Carry flag is 1 (DOS function failed)

Interrupts same as entered

Remarks

Executes the specified DOS function. Generally it is easier to use this function than executing the Int 21h using the DosExecuteV86Int function, since this function takes care of many of the issues related to calling DOS from an NLM, such as setting up NIOS's PSP, failing critical errors, and disabling Control-Break.

This function cannot be used if DOS is busy unless the caller has preserved DOS's data areas prior to invoking this function. Because DOS is never busy inside of an NLM's initialization and unload functions, this function, in the context of a **DosScheduleDosAvailEvent**, can be used without restrictions in these two cases.

The caller must set up a nested execution *ClientRegStruc* prior to calling this function. Also, the caller must set the CRS register values appropriate for the DOS call that will be invoked.

This function yields control.

See Also

DosCall
DosCallUseCurrSDA
DosCallWithDTA

DosCallUseCurrSDA

Description

Executes the specified DOS function.

Syntax

```
#include <dosvmm.h>

UINT32
DosCallUseCurrSDA(
      void);
```

Returns

0	Carry flag is 0 (DOS function successful)
!0	Carry flag is 1 (DOS function failed)

Interrupts same as entered

Remarks

This function is the same as DosCallC except that the current DOS Swappable Data Area (SDA) information is used instead of swapping NIOS's SDA information in.

Generally it is easier to use this function than to execute the Int 21h using the DosExecuteV86Int function, since this function takes care of many of the issues related to calling DOS from an NLM, such as failing critical errors, and disabling Control-Break.

This function cannot be used if DOS is busy unless the caller has preserved DOS's data areas prior to invoking this function. DOS is never busy inside of an NLM's initialization and unload functions; therefore this function can be used without restrictions in these two cases.

The caller must set up a nested execution ClientRegStruc prior to calling this function. In addition the caller must set the CRS register values appropriate for the DOS call that will be invoked.

This function yields control.

See Also

DosCall
DosCallC

DosCallWhenV86IntReturns

Description

This service can be used in an NLM's V86 interrupt handler to obtain control on the back end of a current V86 interrupt.

Assumes

edx Reference data
Interrupts in any state
esi Points to callback handler. Called as follows:

On entry:

```
ebx     -> VM CB
edx     Reference data
ebp     -> CRS
Interrupts are disabled if DOS only case
Interrupts are enabled if MS Windows is active
CLD has been executed
```

On return:

```
CLD preserved
Interrupt state undefined
All registers can be destroyed
```

Returns

Z flag cleared
Interrupt state preserved.
All registers preserved.
Nothing.

Remarks

This function is callable at interrupt time if MS Windows isn't running.

An NLM that uses this service must first call this service, then return from its V86 interrupt handler signalling that the interrupt was NOT consumed. This service is designed so that the NLM's interrupt handler can simply jump to this service and this service will return back from the handler with the Z flag cleared.

This service places a callback address on the current V86 stack such that when the V86 interrupt handling code iret's out of the interrupt, NLM handlers that have used this service will receive control. This occurs in a LIFO manner, thus preserving the ordering that should occur when multiple NLMs hook the back end of the same V86 interrupt.

When the handler is invoked, the current CrsCS, CrsIP, and CrsFlags will hold the current iret information. If the handler needs to modify the return flags it should do so by modifying the CrsFlags field.

See Also

DosCallWithDTA

Description

Makes DOS functions calls that use the Disk Transfer Area.

Syntax

```
#include<dosvmm.h>

UINT32
DosCallWithDTA(
        UINT8      *nlmDTA);
```

Parameters

nlmDTA	Pointer to 128-byte DTA buffer that will be used as the active DTA prior to invoking the DOS function and will receive a copy of the DTA information after the DOS function completes.

Returns

Zero	Carry flag is zero (DOS function successful).
Non-zero	Carry flag is one (DOS function failed)

Interrupts same as at entry.

Remarks

Examples of DOS functions calls that use the DTA are: DOS find first and DOS find next.

Executes the specified DOS function. Generally it is easier to use this function than executing the Int 21h using the **DosExecuteV86Int** function since this function takes care of many of the issues related to calling DOS from an NLM, such as setting up Nios's PSP, failing critical errors, and disabling Control-Break, etc.

This function cannot be used if DOS is busy unless the caller has preserved DOS's data areas prior to invoking this function. DOS is never busy inside of an NLM's initialization and unload functions, and in the context of a **DosScheduleDosA**vailEvent therefore this function can be used without restrictions in these cases.

The caller must setup a nested execution *ClientRegStruc* prior to calling this function. In addition the caller must set the CRS register values appropriate for the DOS call that will be invoked.

This function yields control.

See Also	DosCall DosCallC

DosClose

Description

Closes the specified file.

Syntax

```
#include <dosvmm.h>

UINT32
DosClose(
    modHandle       module,
    UINT32          fileHandle);
```

Parameters

module	Caller's module handle
fileHandle	Handle of file to close

Returns

0	Close was successful
0xFFFFFFFF	Invalid file handle

Remarks

DOS must be in a callable state. Generally this function can be used inside of an NLM's initialization function as well as during an event scheduled using the **DosScheduleDosAvailEvent** service.

See Also

DosCreate
DosOpen
DosDelete
DosRead
DosWrite
DosSeek
DosRename
DosGetFileSize
DosDoesFileExist
DosSearchForFile

DosCMapFlat

Description

DosCMapFlat converts a *selector:offset* to its flat linear address.

Syntax

```
#include <dosvmm.h>

void
*DosCMapFlat(
    UINT32      SelOff);
```

Parameters

SelOff	High word has selector Low word has offset

Returns

Linear address

Remarks

The selector can be either an LDT or a GDT. Passed-in LDT selectors must exist in the active LDT.

See Also

DosAMapFlat

DosConvGetInfo

Description

Returns the size of the largest block of conventional memory that can be currently allocated using DosConvMemAlloc.

Syntax

```
#include <dosvmm.h>

UINT32
DosConvGetInfo(
      void);
```

Returns

Number of bytes in largest free conventional memory block

Remarks

See Also

DosConvMemAlloc

DosConvMemAlloc

Description

Attemps to allocate a block of memory from conventional memory that is globally accessible in all VMs.

Syntax

```
#include <dosvmm.h>

UINT32
DosConvMemAlloc(
    modHandle     module,
    UINT32        options,
    UINT32        size,
    UINT32        *lowAddress,
    void          **linAddress );
```

Parameters

module	Caller's module handle.
options	Reserved for future use, must be 0.
size	Number of bytes to allocate. This value is rounded up to the next highest multiple of 16 (paragraph) value.
lowAddress	Set on return to the allocated memory below 1 megabyte address. This value is always on a paragraph boundary. For example, if the allocated memroy was at D000:0000 (seg:off) then *lowAddress would be set to 000D0000h. This address should not be used by an NLM to read from and/or write to the conventional memory block since there are certain times in a Windows environment where this linear address is not valid, instead always use the *linAddress value to access the memory from protected mode.
linAddress	Set on return to a linear address which can be used to read from and/or write to the allocated conventional memory block.

Returns

zero	Allocation successful.
non-zero	Allocation FAILED.

Remarks

In a DOS-only configuration, **DosConvMemAlloc** attempts the following strategies:

(1) Allocate best fit, UMB.

(2) Allocate DOS memory block using first fit. This allocates the block at the lowest possible address in the 640K address space.

There are situations where conventional memory cannot be allocated. In the DOS-only case, this service will fail if inadequate free UMB memory is available and/or inadequate conventional memory below 640K is available.

In a Windows environment, this function allocates conventional memory from the NIOS global V86 memory region. By default, NIOS allocates 8K for this pool, but that can be changed by specifying a different value for the **GLOBAL V86 WIN PAGES** parameter in the NetWare configuration database.

Note that this pool of memory is only available during the Windows session and becomes invalid once Windows is exited. If, at Windows' exit, conventional memory is still allocated from this pool, NIOS will auto free it before exiting back to DOS.

In this case, an NLM should schedule a DOS available event by using **DosScheduleDosAvailEvent** during the Windows "WIN SYS CRIT EXIT" event. When your DOS available event handler is invoked, use **DosConvMemAlloc** to allocate a new block of memory.

Conventional memory allocated prior to Windows loading is available inside of Windows and after Windows is exited.

In the MS Windows environment, **DosConvMemAlloc** fails if all of the NIOS-reserved global V86 memory region is already allocated, or there is insufficient memory in this pool to satisfy the request.

This function yields.

See Also

DosConvMemFree
DosConvGetInfo

DosConvMemFree

Description

DosConvMemFree frees a block of conventional memory that was previously allocated using the **DosConvMemAlloc** service. This function yields.

Syntax

```
#include <dosvmm.h>

UINT32
DosConvMemFree(
    modHandle       module,
    UINT32          lowAddress);
```

Parameters

module	Caller's module handle
lowAddress	Below 1 megabyte address returned from **DosConvMemAlloc**.

Returns

0	Invalid *lowAddress* parameter
!0	Memory was freed

Remarks

See Also

DosConvMemAlloc

DosCreate

Description

Creates the specified file.

Syntax

```
#include <dosvmm.h>

UINT32
DosCreate (
    modHandle    module,
    UINT8        *filePath,
    UINT32       createAttributes);
```

Parameters

module	Caller's module handle
filePath	ASCIIZ string containing the full or partial path of the file to be created
createAttributes	Attribute flags for the file: CREATE_NORMAL CREATE_HIDDEN CREATE_SYSTEM CREATE_HIDDEN_SYSTEM

Returns

file handle	If create was successful
0xFFFFFFFF	Invalid path, root directory full, existing file is read-only, or insufficient access rights

Remarks

If the file already exists, **NiosCreate** will truncate the file to zero length.

DOS must be in a callable state. Generally this function can be inside of an NLM's initialization function as well as during an event scheduled using the **DosScheduleDosAvailEvent** service.

See Also	DosClose DosOpen DosDelete DosRead DosWrite DosSeek DosRename DosGetFileSize DosDoesFileExist DosSearchForFile

DosDelete

Description

Deletes the specified file.

Syntax

```
#include <dosvmm.h>

UINT32
DosDelete(
    UINT8        *filePath);
```

Parameters

filePath	ASCIIZ string full or partial path of the file to be deleted

Returns

0	Delete was successful
0xFFFFFFFF	Invalid path, file does not exist, file is read-only, or insufficient access rights

Remarks

DOS must be in a callable state. Generally this function can be inside of an NLM's initialization function as well as during an event scheduled using the **DosScheduleDosAvailEvent** service.

See Also

DosClose
DosOpen
DosCreate
DosRead
DosWrite
DosSeek
DosRename
DosGetFileSize
DosDoesFileExist
DosSearchForFile

DosDeRegisterUserCmd

Description

Deregisters a previously installed custom DOS command processor command.

Syntax

```
#include <cmdcom.h>

UINT32
DosDeRegisterUserCmd(
    modHandle               moduleHandle,
    struct UserCmdStruc     *userCmdInfo);
```

Parameters

moduleHandle	Caller's module handle
userCmdInfo	Pointer to UserCmd structure

Returns

0	Command successfully deregistered
0xFFFFFFFF	Specified command was not registered

Remarks

See Also

CMDCOM.H
CMDCOM.INC
DosRegisterUserCmd
DosEnumerateUserCmds

DosDeRegisterV86Int2F

Description

DosDeRegisterV86Int2F removes a previously registered Interrupt 2Fh handler.

Syntax

```
#include <dosvmm.h>

UINT32
DosDeRegisterV86Int2F
     (modHandle               moduleHandle,
     struct Int2FInfoStruc    *int2FInfo);
```

Parameters

moduleHandle	Caller's module handle
int2FInfo	Pointer to *Int2FInfoStruc* that was used to originally register the handler

Returns

0	Handler was deregistered successfully
0xFFFFFFFF	Handler was not registered

Remarks

See Also

DosRegisterV86Int2F

DosDoesFileExist

Description

Determines if the specified [path\] file exists.

Syntax

```
#include <dosvmm.h>

UINT8
DosDoesFileExist(
    UINT8 *filename);
```

Parameters

filename	Specifies the file to look for. Cannot contain wildcards.

Returns

0	File does NOT exist
!0	File does exist

Remarks

DOS must be in a callable state. Generally this function can be inside of an NLM's initialization function as well as during an event scheduled using the **DosScheduleDosAvailEvent** service.

See Also

DosClose
DosOpen
DosCreate
DosGetFileSize
DosRead
DosDelete
DosSeek
DosWrite
DosSearchForFile

DosEndNestExec

Description

DosEndNestExec is callable at interrupt time if MS Windows isn't running. This function ends a previously started nested execution. The passed-in CRS is deallocated and the previous CRS is restored.

Assumes

ebp Points to CRS
Interrupt state undefined

Returns

ebp destroyed
All other registers preserved
Interrupts same as entered

Remarks

The ESP value coming into this routine must have the same value as when the DosBeginNestExec function returned.

Note: Do NOT access the nested execution CRS registers after this function has been called.

See Also

DosBeginNestExec
DosBeginNestExecWithCrs
DosEndNestExecWithCrs

DosEndNestExecWithCrs

Description

This function ends a previously started nested execution entered using the **DosBeginNestExecWithCrs** service. Ownership of the CRS structure provided to start the nested block is returned to the NLM when this service is invoked.

Syntax

```
#include <dosvmm.h>

CRS
*DosEndNestExecWithCrs(
     CRS      *saveCrsBuf);
```

Parameters

saveCrsBuf	Pointer to CRS structure that was used during the call to DosBeginNestExec to preserve the current CRS

Returns

Pointer to previous CRS

Remarks

This function is callable at interrupt time if MS Windows isn't running. V86 code is NOT callable at interrupt time under MS Windows.

Note: Do NOT access the nested execution CRS registers after this function has been called.

See Also

DosBeginNestExecWithCrs
DosBeginNestExec
DosEndNestExec

DosEndReentrantExec

Description

Ends a reentrant execution block.

Syntax

```
#include <dosvmm.h>

void
DosEndReentrantExec(
        UINT32 prevCount);
```

Parameters

prevCount	Value for restoring the internal interrupt nesting level obtained from a previous call.

Returns

Nothing

Remarks

The internal interrupt nesting level is restored to *prevCount* which should be the value obtained from a previous call to **DosBeginReentrantExec**.

See Also

DosBeginReentrantExec

DosEndUseDos

Description

This function restores the Control-C, Control-Break, and Interrupt 24h vectors in the current VM with the previous vectors that were preserved using **DosBeginUseDos**.

Syntax

```
#include <dosvmm.h>

void
DosEndUseDos(
    UINT32      *savedVectInfo);
```

Parameters

savedVectInfo	Pointer to an array of 3 UINT32's that holds the previously saved INT 1Bh, INT 23h, and INT 24h vectors from a call to **DosBeginUseDos**

Returns

Nothing
Interrupts same as entered

Remarks

See Also

DosBeginUseDos

DosEnumerateUserCmds

Description

Allows the caller to determine which DOS custom commands have been registered with NIOS.

Syntax

```
#include <cmdcom.h>

UINT32
DosEnumerateUserCmds(
    struct UserCmdStruc       *userCmdInfo);
```

Parameters

userCmdInfo	Pointer to a UserCmdStruc structure that will be filled with next registered command structure

Returns

0	Next command successfully found
0xFFFFFFFF	No more registered commands

Remarks

Note that the UCText buffer must be provided by the user of this function and will be a copy of the actual command text.

The passed-in UserCmdStruc structure's UCText field must point to a 10-byte buffer which will receive a copy of command's text.

To start the search, UCOwner must be set to NULL. Subsequent calls to this function use the previously returned text in the UCText buffer and the previous value in the UCOwner field to locate the next registered command.

See Also

CMDCOM.H
CMDCOM.INC
DosRegisterUserCmd
DosDeRegisterUserCmd

DosExecuteFarRet

Description

Use **DosExecuteFarRet** to execute (simulate) a V86 retf instruction.

Assumes

ebp Points to CRS

Returns

ebp.CrsSP	increased by four
ebp.CrsCS	set to seg value on stack
ebp.CrsIP	set to off value on stack
eax, edx	destroyed

All other registers preserved

Remarks

An in-line version of this function is available using the macro **DosFastExecuteFarRet.**

See Also

DosExecuteIRet

Description

DosExecuteIRet executes (simulates) a V86 iret instruction.

Assumes

ebp Points to CRS

Returns

ebp.CrsSP	increased by six
ebp.CrsCS	set to seg value on stack
ebp.CrsIP	set to off value on stack
ebp.CrsEFlags	set to flags on stack
eax, edx	destroyed

All other registers preserved

Remarks

NLMs should not perform a simulated V86 IRET manually. Always use this function for proper operation under MS Windows.

See Also

DosExecutePop

Description	Use **DosExecutePop** to execute (simulate) a V86 pop instruction.
Assumes	*ebp* Points to CRS
Returns	ax Value popped from stack ebp.CrsSP increased by two edx destroyed All other registers preserved
Remarks	An in-line version of this function is available using the macro **DosFastExecutePop.**
See Also	

DosExecutePush

Description

Use **DosExecutePush** to execute a V86 push instruction.

Assumes

ax	Value to push
ebp	Points to CRS

Returns

ebp.CrsSP	decreased by two
crs.Stack	has ax value

All registers preserved

Remarks

An in-line version of this function is available using the macro **DosFastExecutePush.**

See Also

DosExecuteV86FarCall

Description

DosExecuteV86FarCall switches to real mode and executes a far call to a specified procedure.

Assumes

eax seg:off of V86 routine to call
ebp Points to CRS
Interrupt state undefined

Returns

ebp Pointer to CRS
All other pm registers destroyed
Interrupt state preserved

Remarks

This function is callable at interrupt time if MS Windows isn't running.

If called from the foreground, this function yields to any waiting foreground events.

The caller must call **DosBeginNestExec** or **DosBeginNestExecWithCrs** before invoking this function.

See Also

DosBeginNestExec
DosBeginNestExecWithCRS

DosExecuteV86Int

Description	**DosExecuteV86Int** switches to real mode and executes a specified interrupt.
Assumes	*ebp* Points to CRS *al* Interrupt to execute Interrupt state undefined
Returns	*ebp* Pointer to CRS All other pm registers destroyed Interrupts state preserved
Remarks	This function is callable at interrupt time if MS Windows isn't running. If called from the foreground, this function yields to any waiting foreground events. The caller must call **DosBeginNestExec** or **DosBeginNestExecWithCRS** before invoking this function.
See Also	DosBeginNestExec DosBeginNestExecWithCrs

DosFlush

Description

Flushes all disk buffers using DOS function 0Dh.

Syntax

```
void
DosFlush( void);
```

Parameters

None

Returns

Nothing

Remarks

Note: Certain disk cache programs will not flush write behind data unless the target data files have been closed.

DOS must be in a callable state. Generally this function can be used inside of an NLM's initialization function as well as during an event scheduled using the **DosScheduleDosAvailEvent** service.

This function yields.

See Also

DosCreate, DosOpen, DosClose, DosDelete, DosRead, DosWrite, DosSeek, DosRename, DosGetFileSize, DosDoesFileExist, DosSearchForFile

DosFastExecuteFarRet

Description

DosFastExecuteFarRet executes (simulates) a V86 retf instruction.

Assumes

#include dosvmm.inc

ebp Pointer to CRS

Returns

ebp.CrsSP	increased by four
ebp.CrsCS	set to seg value on stack
ebp.CrsIP	set to off value on stack
eax, edx	destroyed

All other registers preserved

Remarks

A non-inline version of this macro is available using the function **DosExecuteFarRet.**

See Also

DosFastExecutePop

Description

Use the **DosFastExecutePop** macro to execute (simulate) a V86 pop instruction.

Assumes

#include dosvmm.inc

ebp Points to CRS

Returns

ax	Value popped from stack
ebp.CrsSP	increased by two
edx	destroyed

All other registers preserved

Remarks

A non-inline version of this macro is available using the function **DosExecutePop.**

See Also

DosFastExecutePush

Description	Use the **DosFastExecutePush** macro to execute (simulate) a V86 push instruction.

Assumes

#include dosvmm.inc

ax	Value to push
ebp	Pointer to crs

Returns

ebp.CrsSP	decreased by two
crs.Stack	has ax value

All registers preserved

Remarks

A non-inline version of this macro is available using the function **DosExecutePush.**

See Also

DosFreeV86Callback

Description

DosFreeV86Callback deallocates a previously allocated V86 callback handler.

Syntax

```
#include <dosvmm.h>

UINT32
DosFreeV86Callback(
    modHandle     moduleHandle,
    UINT32        v86CallbackAddress);
```

Parameters

moduleHandle	Caller's module handle
v86CallbackAddress	Seg:Off to free

Returns

!0	Callback does not exist
0	Callback was successfully freed

Remarks

See Also

DosGetCurrVmHandle

Description

DosGetCurrVmHandle returns a pointer to the currently executing VM's control block.

Assumes

#include <dosvmm.inc>

Returns

EBX -> VM control block
All other registers preserved
Interrupt state preserved

Remarks

Note that NLMs should not save the VM CB pointer value and use it later in a different thread of execution, since its value can change when the user enters and/or exits the MS Windows environment. An NLM should use the VM ID (VMCBVmId) for this purpose.

This function is callable at interrupt time in DOS and MS Windows environments.

See Also

DosGetNextVmHandle

DosGetExeContext

Description

DosGetExeContext returns whether the caller can invoke non-interrupt-time callable APIs in the current execution context.

Syntax

```
#include <dosvmm.h>

UINT32
DosGetExeContext(
    void);
```

Parameters

None

All registers are preserved except eax

Returns

0	Execution is foreground
!0	Execution is in the context of a hardware interrupt

Remarks

This function is callable at interrupt time in DOS and MS Windows environments.

See Also

NiosScheduleForegroundEvent found in *NetWare Client NIOS Dictionary*

DosGetFileSize

Description	Returns the length of the specified file.

Syntax

```
#include <dosvmm.h>

UINT32
DosGetFileSize(
    UINT32      fileHandle);
```

Parameters

fileHandle	Handle of file for which to return size.

Returns

Size of file

0xFFFFFFFF	Error

Remarks

The current seek position is preserved.

DOS must be in a callable state. Generally this function can be inside of an NLM's initialization function as well as during as event scheduled using the **DosScheduleDosAvailEvent** service.

See Also

DosClose
DosOpen
DosCreate
DosRead
DosWrite
DosDelete
DosSeek
DosDoesFileExist
DosSearchForFile

DosGetNextVmHandle

Description

DosGetNextVmHandle returns a pointer to the next active VM control block. It is callable at interrupt time in DOS and MS Windows environments.

Assumes

EBX	Pointer to previous VM control block

Returns

EBX	Pointer to VM control block

All other registers preserved
Interrupt state preserved

Remarks

The list of VM's control blocks is circular; therefore the caller must check for end of list by comparing the returned value against the starting VM control block value to know when the last VM control block has been returned.

Note that NLMs should not save the VM CB pointer value and use it later in a different thread of execution since its value can change when the user enters and/or exits the MS Windows environment. An NLM should use the VM ID (VMCBVmId) for this purpose.

See Also

DosGetCurrVmHandle

DosHookExceptionInterrupt

Description

DosHookExceptionInterrupt hooks the specified protected-mode interrupt (0,2,4-31).

Syntax

```
#include <dosvmm.h>

UINT32
DosHookExceptionInterrupt(
    modHandle     moduleHandle,
    UINT32        intToHook,
    UINT32        referenceData,
    void          (*intHandler)(void));
```

Parameters

moduleHandle Caller's module handle.

intToHook Interrupt to hook (0,2,4-31).

referenceData Value passed to intHandler. This value can be anything the caller desires, or can be ignored if not needed.

intHandler Pointer to routine that is called when the specified interrupt occurs.

The entry and exit conditions are:

```
On entry:
edx      referenceData
Interrupts are disabled
CLD has been executed
ebp      -> stack frame
Stack frame
    EFlags
    CS
    EIP
    Error Code (if present)
    pushad regs
```

On return:

Z flag set if interrupt was serviced
else pass int to next handler
CLD preserved
Interrupts are disabled
All registers can be destroyed

Returns

0	Interrupt was hooked successfully
0xFFFFFFFF	Not enough free memory to hook the interrupt
0xFFFFFFFE	Invalid intToHook value
0xFFFFFFFD	Service not supported by NIOS environment

Remarks

The *intHandler* is only invoked when the specified exception is invoked while in protected mode and executing in NIOS. Exceptions generated in real mode are not seen by the caller's *intHandler*.

Callers that wish to hook one of the other processor interrupts (1,3,32-255) should do so by using either **DosHookPMInterrupt** or **DosHookV86Interrupt.**

See Also

DosHookPMInterrupt

Description

DosHookPMInterrupt hooks the specified protected-mode interrupt (1,3,32-255).

Syntax

```
#include <dosvmm.h>

UINT32
DosHookPMInterrupt(
    modHandle       moduleHandle,
    UINT32          intToHook,
    UINT32          referenceData,
    void            (*intHandler)( void));
```

Parameters

moduleHandle	Caller's module handle.
intToHook	Interrupt to hook (1,3,32-255).
referenceData	Value passed to intHandler. This value can be anything the caller desires, or can be ignored if not needed.
intHandler	Pointer to routine that is called when the specified interrupt occurs. The entry and exit conditions are:

```
On entry:
edx       referenceData
Interrupts are disabled
CLD has been executed
ebp       -> stack frame
Stack frame
    EFlags
    CS
    EIP
    pushad regs
```

On return:
Z flag set if interrupt was serviced
    else pass int to next handler
CLD preserved
Interrupts are disabled
All registers can be destroyed

Returns

0	Interrupt was hooked successfully
0xFFFFFFFF	Not enough free memory to hook the interrupt
0xFFFFFFFE	Invalid intToHook value
0xFFFFFFFD	Service not supported by NIOS environment

Remarks

If an interrupt is not serviced by a protected-mode handler it is passed on to the real-mode interrupt handlers. The *intHandler* is only invoked when the specified interrupt is invoked while in protected mode and executing in NIOS. Interrupts generated in real mode are not seen by the caller's *intHandler*.

Note that this function only hooks the interrupt in the context of the DOS NIOS IDT. Therefore, if the interrupt occurs in real mode or while Windows is active, the specified handler will *not* be invoked.

Callers who wish to hook one of the processor exception interrupts (0,2,4-31) should do so by using **DosHookExceptionInterrupt**.

Callers who wish to hook an interrupt regardless of the mode (real or protected) in which the interrupt occurred should use **DosHookV86Interrupt**.

See Also

DosHookV86Interrupt

Description

DosHookV86Interrupt hooks the specified V86 interrupt (0-255).

Syntax

```
#include <dosvmm.h>

UINT32
DosHookV86Interrupt(
    modHandle     moduleHandle,
    UINT32        intToHook,
    UINT32        referenceData,
    void          (*intHandler)( void));
```

Parameters

moduleHandle Caller's module handle.

intToHook Interrupt to hook (0-255).

referenceData Value passed to intHandler. This value can be anything the caller desires, or can be ignored if not needed.

intHandler Pointer to routine that is called when the specified interrupt occurs.

The handler's entry and exit conditions are:

```
On entry:
ebx     -> VM CB
edx     referenceData
ebp     -> CRS
Interrupts are disabled in DOS-only case
Interrupts are enabled if MS Windows is active
CLD has been executed

On return:
Z flag set if interrupt was serviced
    else int is passed to next handler
CLD preserved
Interrupt state undefined
All registers can be destroyed
```

Returns

0	Interrupt was hooked successfully.
0xFFFFFFFF	Not enough internal resources to complete the operation.
0xFFFFFFFE	Invalid intToHook value.
0xFFFFFFFD	2Fh specified as intToHook which is invalid.
0xFFFFFFFC	V86 interrupt was not hooked prior to MS Windows loading. The calling NLM cannot be loaded when MS Windows is active. The hook operation will be scheduled and executed when MS Windows is exited for Win 3.x.

Remarks

Use this service instead of **DosHookPMInterrupt** if the caller is interested in getting called whenever the specified interrupt is invoked--whether it be from protected mode or real mode--since the **DosHookPMInterrupt** service only calls the *intHandler* when the interrupt is invoked while in protected mode.

Note: Interrupt 2Fh should not be hooked using this API; instead use the **DosRegisterV86Int2F** function.

NLMs that need to pass a V86 interrupt down the chain and then receive control on the back end of the interrupt can use **DosCallWhenV86IntReturns** inside of their interrupt handler function.

When the handler is invoked, the current CrsCS, CrsIP, and CrsFlags will hold the current iret information. If the handler needs to modify the return flags it should do so by modifying the CrsFlags field.

See Also

DosUnHookV86Interrupt
DosCallWhenV86IntReturns

DosIsDosBusy

Description

Determines whether DOS is in a callable state.

Syntax

```
#include <dosvmm.h>

UINT32
DosIsDosBusy(
    void);
```

Parameters

None

Returns

0	DOS is NOT busy
!0	DOS is busy

Remarks

This service considers DOS to be callable if the InDOS and Critical Error flags are zero. (In an MS Windows environment, this information pertains to the currently running VM.)

DOS is always callable inside an NLM's initialization and unload procedures, since initialization and unload always execute in a DOS foreground context.

If this service determines that DOS is NOT callable, an NLM should use **DosScheduleDosAvailEvent** to schedule a callback that will be invoked when the current DOS function is finished or whenever an INT 28h (Idle Interrupt) is generated with the InDOS flag set to 1.

See Also

DosOpen

Description

Opens the specified file.

Syntax

```
#include <dosvmm.h>

UINT32
DosOpen(
    modHandle     module,
    UINT8         *pathSpec,
    UINT32        openAttributes);
```

Parameters

module	Caller's module handle
pathSpec	Pointer to ASCIIZ string describing the [path\]name of file to open
openAttributes	Defined in DOSVMM.H and DOSVMM.INC

Returns

File handle	
0xFFFFFFFF	Error opening file

Remarks

DOS must be in a callable state. Generally this function can be inside of an NLM's initialization function as well as during an event scheduled using the **DosScheduleDosAvailEvent** service.

See Also

DosCreate
DosClose
DosDelete
DosRead
DosWrite
DosSeek
DosRename
DosGetFileSize
DosDoesFileExist
DosSearchForFile

DosRead

Description

Reads data from the specified file.

Syntax

```
#include <dosvmm.h>

UINT32
DosRead(
    UINT32     fileHandle,
    UINT32     readOffset,
    void       *readBuf,
    UINT32     readSize);
```

Parameters

fileHandle	Handle of file to read from (returned by **DosOpen** or **DosCreate**). The file must have been opened for reading.
readOffset	Offset from start of file to read from. 0xFFFFFFFF if read occurs at current position.
readBuf	Pointer to buffer to hold read data.
readSize	Number of bytes to read (0 - 0xFFFFFFFC).

Returns

Number of bytes read

0xFFFFFFFF	Seek failed
0xFFFFFFFE	I/O error during read
0xFFFFFFFD	Function failure

Remarks

DOS must be in a callable state. Generally this function can be inside of an NLM's initialization function as well as during an event scheduled using the **DosScheduleDosAvailEvent** service.

See Also	DosClose DosOpen DosCreate DosGetFileSize DosWrite DosDelete DosSeek DosDoesFileExist DosSearchForFile

DosRegisterUserCmd

Description

This function installs a new DOS command, allowing custom commands to be added to the list of resident commands available in the active command processor (such as COMMAND.COM).

Syntax

```
#include <cmdcom.h>

UINT32
DosRegisterUserCmd(
      modHandle               moduleHandle,
      struct UserCmdStruc     *userCmdInfo,
      UINT32                  options);
```

Parameters

moduleHandle	Caller's module handle.
userCmdInfo	Pointer to UserCmd structure. The memory for this structure is owned by NIOS until the command is deregistered. The caller must set the UCHandler and UCText fields. UCOwner will be set by this function.
options	Reserved for future use. Must be set to zero.

Returns

0	Command installed successfully

Remarks

When the user enters the registered command, the specified callback is invoked to allow processing of the command. The handler is invoked as follows:

```
UINT32 (userCmdInfo->UCHandler)(
      struct UserCmdStruc     *userCmdInfo,
      UINT8                   *cmdLine,
      UINT32                  argCount,
      UINT8                   *argVector[])
```

userCmdInfo	Pointer to UserCmdStruc used to register the command.
cmdLine	Pointer to len-preceded string containing any parameters entered after the command. This string is not NULL terminated. Typically, the handler should ignore this parameter, instead making use of the parsed parameter information found in the next two parameters.

argCount	Count of parsed parameters found after the command. This value will be zero if no parameters were specified.
argVectors	Pointer to array of pointers to ASCIIz string parameters. argCount defines the number of entries in this array.

Each entry will be stripped of leading and trailing white space unless the information is found inside quotes. No upper/lower case conversions are performed. Any leading '-' character will be converted to the '/' character to allow the user to use either switch character.

Note that the buffers used to hold the individual parameters can be modified by the UCHandler (e.g., case conversions) as long as the handler does not access past the end of the string.

Returns: Command completion code. The low-order byte of this value is passed back as the ERRORLEVEL.

See Also

CMDCOM.H
CMDCOM.INC
DosDeRegisterUserCmd
DosEnumerateUserCmds

DosRegisterV86Int2F

Description

Installs a handler for the specified Int 2Fh AH value. The handler is invoked when an Interrupt 2Fh is executed in real mode with AH equal to the value in the I2FAhValue field of the passed-in structure.

Syntax

```
#include <dosvmm.h>

void
DosRegisterV86Int2F(
      modHandle                 moduleHandle,
      struct Int2FInfoStruc     *int2FInfo);
```

Parameters

moduleHandle Caller's module handle.

int2FInfo Pointer to **Int2FInfoStruc** with the I2FHandler and I2FAhValue fields set. The entry and exit conditions for the I2FHandler are:

```
On entry:
ebx     -> VM CB
ecx 0   if interrupt was seen with a hook in the V86
        vector table.
        !0   if interrupt was seen from a Windows
             protected mode V86 hook procedure.
edx     -> referenceData
ebp     -> CRS
Interrupts are disabled in DOS case only
Interrupts are enabled if MS Windows is active
CLD has been executed

On return:
Z flag set if interrupt was serviced
      else int is passed to next handler
CLD preserved
Interrupt state undefined
All registers can be destroyed
```

Returns

Nothing

Remarks

The memory for the int2FInfo structure is owned by NIOS until the handler is deregistered using the **DosDeRegisterV86Int2F** function.

This function should be used in place of hooking the V86 Int 2F interrupt vector directly to allow for more efficient Int 2F processing.

When the handler is invoked, the current CrsCS, CrsIP, and CrsFlags will hold the current iret information. If the handler needs to modify the return flags it should do so by modifying the CrsFlags field.

See Also DosDeRegisterV86Int2F

DosRename

Description

Renames the specified file.

Syntax

```
#include <dosvmm.h>

UINT32
DosRename(
     UINT8       *currentFilePath,
     UINT8       *newFilePath);
```

Parameters

filePath	ASCIIZ string containing full or partial path of the file to be renamed

Returns

0	If rename was successful
0xFFFFFFFF	Invalid path, file does not exist, new file already exists, new file on different disk, root directory full, or insufficient access rights

Remarks

This function will move the file to a different directory if the *newFilePath* is different and is on the same disk as *currentFilePath*.

DOS must be in a callable state. Generally this function can be inside of an NLM's initialization function as well as during an event scheduled using the **DosScheduleDosAvailEvent** service.

See Also

DosClose
DosOpen
DosCreate
DosRead
DosWrite
DosSeek
DosDelete
DosGetFileSize
DosDoesFileExist
DosSearchForFile

DosScheduleDosAvailEvent

Description

This function schedules an event that will be called when the currently executing DOS function has finished or when DOS issues interrupt 28h (idle interrupt) with the InDOS flag set to one.

Syntax

```
#include <dosvmm.h>

UINT32
DosScheduleDosAvailEvent(
     FEB      *eventBlock);
```

Parameters

eventBlock	Pointer to a FEBStruc with the FEBESR field set to point to a valid procedure that will be invoked when DOS is in a callable state. The FEB structure passed to this function is owned by NIOS until either the event completes or it is canceled.

Returns

All registers can be destroyed
Interrupts in any state

Remarks

If DOS is not busy when this function is called, the FEBESR is invoked immediately. In an MS Windows environment, the event can be serviced in a different VM than the VM that was active when this service was called.

Note that the FEBESR handler cannot call DOS functions below 0Dh or functions 3Fh,40h with a file handle which references a CON device since this causes stack reentrancy inside of DOS when the FEBESR wakes up because of a DOS INT 28h (Idle interrupt).

If the event handler needs to invoke BIOS or other non-reentrant functions, it must verify that the desired function is not busy (that is, would be reentered). When the FEBESR is invoked, this service guarantees that the video (Int 10h), disk (Int 13h), mouse (Int 15h), and keyboard (Int 16h) BIOS services are NOT busy. No guarantees are made for other BIOS services.

When the defined FEBESR is invoked, a nested ClientRegStruc is already set up for the NLM to use; therefore the NLM does not need to call DosBeginNestExec, etc., before issuing DOS calls.

The FEBESR is invoked as follows:

```
assumes:    For "C" ESRs:   void (*FEBESR)(
                                FEB *eventBlock,
                                CRS *crs)
            For "asm" ESRs:
                            esi  -> eventBlock
                            ebp -> Nested ClientRegStruc
            CLD has been executed
            Interrupts are enabled
```

Returns

0	DOS was not busy, event completed before this function returned to the caller
!0	DOS was busy, event was scheduled

See Also

DosCancelDosAvailEvent
DosIsDosBusy
DosCall
DosCallC

DosSearchForFile

Description

Searches the PATH environmental variable for the specified file. The current directory is tried first.

Syntax

```
#include <dosvmm.h>

UINT8
DosSearchForFile(
     UINT8       *filename);
```

Parameters

filename	Searches for the specified file. The current directory is tried first, then the NIOS System directory, the the PATH. This cannot contain wildcards. If successful, this buffer is used to hold the path and filename of the found file. This buffer must be large enough to hold the worst-case path\filename.

Returns

zero	File could NOT be found
non-zero	File was found (filename holds result)

Remarks

DOS must be in a callable state. Generally this function can be inside of an NLM's initialization function as well as during an event scheduled using the **DosScheduleDosAvailEvent** service. This function assumes that DOS is in a callable state. This function yields.

See Also

DosClose
DosOpen
DosCreate
DosGetFileSize
DosRead
DosDelete
DosSeek
DosWrite
DosDoesFileExist

DosSeek

Description

NiosSeek moves the file's read/write file pointer to the specified position.

Syntax

```
#include <dosvmm.h>

UINT32
DosSeek(
    UINT32    fileHandle,
    UINT32    seekType,
    UINT32    seekOffset);
```

Parameters

fileHandle	Handle of file in which to move pointer
seekType	One of the following: SEEK_SET Move *seekOffset* from beginning SEEK_CURRENT Move *seekOffset* from current position SEEK_END Move *seekOffset* from end
seekOffset	Number of bytes to move pointer from specified starting location (*seekType*)

Returns

New pointer position

0xFFFFFFFF	Seek failed

Remarks

DOS must be in a callable state. Generally this function can be inside of an NLM's initialization function as well as during an event scheduled using the **DosScheduleDosAvailEvent** service.

See Also	DosClose DosOpen DosCreate DosRead DosWrite DosDelete DosGetFileSize DosDoesFileExist DosSearchForFile

DosSharedBufAlloc

Description

DosSharedBufAlloc must be called before using the NIOS shared DOS buffer. This function is callable at interrupt time.

Syntax

```
#include <dosvmm.h>

UINT32
DosSharedBufAlloc(
    modHandle       module);
```

Parameters

module	Caller's module handle

Returns

0	Allocation was successful.
!0	Error allocating buffer. Not enough free memory available to preserve buffer contents.

Interrupt state preserved and not changed.

Remarks

This function is callable at interrupt time in a DOS-only environment. If called during interrupt time in a DOS-only environment, the caller must check to see if the buffer contents need preservation by checking the return code from **DosSharedBufGetInfo**. If it does, then the caller must preserve the contents and restore the contents before calling **DosSharedBufFree**.

This buffer can be used by anyone in the system needing temporary global DOS memory. The shared buffer must be allocated and freed in the same execution thread.

This function uses a semaphore to ensure only one access to the shared buffer at a time. If an attempt is made to allocate the buffer reentrantly in the context of the same VM, then the buffer contents will be preserved prior to returning. In this case the contents will be restored when the caller invokes **DosSharedBufFree**.

The fact that the buffer contents may be swapped out in the background could preclude its use in some situations, in which case a private DOS buffer must be allocated by the module.

The size and location of the shared buffer can be determined using **DosSharedBufGetInfo.**

See Also	DosSharedBufFree DosSharedBufGetInfo DosGetExeContext

DosSharedBufFree

Description

DosSharedBufFree must be called after using the NIOS shared DOS buffer.

Syntax

```
#include <dosvmm.h>

void
DosSharedBufFree(
    modHandle       module);
```

Parameters

module	Caller's module handle

Returns

Nothing
Interrupt state preserved and not changed

Remarks

If the buffer's contents had to be preserved, the contents will be restored by this function before returning.

If the buffer was allocated during interrupt context and the buffer was already in use, the caller must restore the contents itself prior to invoking this function.

This function is callable at interrupt time in a DOS-only environment.

See Also

DosSharedBufAlloc

DosSharedBufGetInfo

Description

DosSharedBufGetInfo returns information about the shared DOS buffer maintained by NIOS.

Syntax

```
#include <dosvmm.h>

UINT32
DosSharedBufGetInfo(
    SDBInfo     *sdbiStruc);
```

Parameters

sdbiStruc	Pointer to buffer set on return with a copy of the current shared DOS buffer information

Returns

zero	Buffer is unallocated.
non-zero	Buffer is currently allocated. Use of DosSharedBufAlloc will cause the current contents to be preserved if current execution context is not interrupt time.

Interrupt state preserved and not changed.

Remarks

NIOS manages a shareable conventional memory block (< 1 Meg) that can be allocated and used by NLMs. If an NLM only needs a block of conventional memory for a short period of time, it can use this service instead of allocating a block for itself, thus reducing overall conventional memory usage.

This function is callable at interrupt time in a DOS-only environment.

The minimum size of the Shared DOS buffer is 512 bytes. The actual size is returned in the *SDBSize* field of the returned *SDBInfo* structure.

See Also

DosSharedBufAlloc
DosSharedBufFree
DosGetExeContext

DosUnHookExceptionInterrupt

Description

Unhooks the caller from the specified exception interrupt chain.

Syntax

```
#include <dosvmm.h>

UINT32
DosUnHookExceptionInterrupt(
    modHandle     moduleHandle,
    UINT32        intToUnHook,
    void          (*intHandler)( void));
```

Parameters

moduleHandle	Caller's module handle
intToUnHook	Interrupt to unhook from (0,2,4-31)
intHandler	Pointer to caller's interrupt handler routine

Returns

0	Caller was unhooked successfully
0xFFFFFFFF	Caller was not hooking the interrupt
0xFFFFFFFE	Invalid intToUnHook value
0xFFFFFFFD	Service not supported by NIOS environment

Remarks

See Also

DosUnHookPMInterrupt

Description

Unhooks the caller from the specified interrupt chain.

Syntax

```
#include <dosvmm.h>

UINT32
DosUnHookPMInterrupt(
    modHandle     moduleHandle,
    UINT32        intToUnHook,
    void          (*intHandler)( void));
```

Parameters

moduleHandle	Caller's module handle
intToUnHook	Interrupt to unhook from (32-255)
intHandler	Pointer to caller's interrupt handler routine

Returns

0	Caller was unhooked successfully
0xFFFFFFFF	Caller was not hooking the interrupt
0xFFFFFFFE	Invalid intToUnHook value
0xFFFFFFFD	Service not supported by NIOS environment

Remarks

See Also

DosUnHookV86Interrupt

Description

Unhooks the caller from the specified V86 interrupt chain.

Syntax

```
#include <dosvmm.h>

UINT32
DosUnHookV86Interrupt(
    modHandle     moduleHandle,
    UINT32        intToUnHook,
    void          (*intHandler)(void));
```

Parameters

moduleHandle	Caller's module handle
intToUnHook	Interrupt to unhook from (0-255)
intHandler	Pointer to caller's interrupt handler routine

Returns

0	Caller was unhooked successfully
0xFFFFFFFF	Caller was not hooking the interrupt
0xFFFFFFFE	Invalid intToUnHook value

Remarks

See Also

DosVid16DeregisterGuiCB

Description

Cancels a previously registered GUI callback given the function.

Syntax

```
UINT32
DosVid16DeregisterGuiCB (
      UINT32          funcNum,
      UINT32          (far *callback) );
```

Parameters

funcNum Index to function list as follows:

```
0 = UINT32 NiosVidMessageBox (
            UINT8 far       *title,
            UINT8 far       *prompt,
            UINT32          buttons);

1 = UINT32 NiosVidInputDialogBox (
            UINT8 far  *title,
            UINT8 far  *prompt,
            UINT8 far       *input,
            UINT32          length);

2 = void far *NiosVidCreateDialogBox (
            UINT8 far  *title,
            UINT8 far       *prompt0:

3 = UINT32 NiosVidDestroyDialogBox (
            void far        *handle);

4 = UINT32 NiosVidUpdateDialogBox (
            void far        *handle,
            UINT8 far  *prompt,
            UINT8 far  *title);
```

callback Pointer to function address

Returns

0	Successful
0xFFFFFFFF	Invalid parameters

Remarks

See Also DosVid16RegisterGuiCB

DosVid16RegisterGuiCB

Description

Sets the address of the current GUI callback as defined by *funcNum*.

Syntax

```
UINT32
DosVid16RegisterGuiCB (
    UINT32          funcNum,
    UINT32          (far *callback) );
```

Parameters

funcNum Index to function list as follows:

```
0 = UINT32 NiosVidMessageBox (
        UINT8 far       *title,
        UINT8 far       *prompt,
        UINT32          buttons);

1 = UINT32 NiosVidInputDialogBox (
        UINT8 far  *title,
        UINT8 far  *prompt,
        UINT8 far       *input,
        UINT32          length);

2 = void far *NiosVidCreateDialogBox (
        UINT8 far  *title,
        UINT8 far       *prompt0:

3 = UINT32 NiosVidDestroyDialogBox (
        void far        *handle);

4 = UINT32 NiosVidUpdateDialogBox (
        void far        *handle,
        UINT8 far  *prompt,
        UINT8 far  *title);
```

callback Pointer to function address

Returns

0	Successful
0xFFFFFFFF	Invalid function or registry full

Remarks

See Also DosVid16DeregisterGuiCB

DosVidCallWhenPopupOk

Description

Schedules an event that will fire when the system is capable of displaying a popup message.

Syntax

```
#include <dosvmm.h>

UINT32
DosVidCallWhenPopupOk(
    FEB     *eventBlock);
```

Parameters

eventBlock	Pointer to FEBStruc with the FEBESR field set

Returns

zero	Event completed before this function returned
non-zero	Event was scheduled

Remarks

When the callback is invoked the current VM will be the focus VM and execution will be in the foreground sufficient for using the **DosVid** popup/keyboard services.

While scheduled, the *eventBlock.FEBStatus* field will be set to a non-zero value. When invoked, the *FEBStatus* field will be zero.

This function is callable at interrupt time in all environments.

See Also

DosVidCheckKey

Description

Determines if a key is waiting in the keyboard buffer; if there is, it is returned.

Syntax

```
#include <dosvmm.h>

UINT16
DosVidCheckKey(
    void);
```

Parameters

None

Returns

0xFFFF	No key available

else
Lower byte contains ASCII code or other translation
Upper byte contains scan code or special character ID

Remarks

The key is removed from the keyboard input queue.

Returned Scan/ASCII codes are compatible with BIOS Int 16h, functions 10h/11h. The key is NOT displayed by this service.

Keyboard BIOS must be callable.

This service is intended to be used only in the context of a popup.

See Also

DosVidGetKey
DosVidEmptyTypeAhead

DosVidCursorSet

Description

This service positions the cursor to the specified x,y coordinate in the specified popup.

Syntax

```
#include <dosvmm.h>

UINT32
DosVidCursorSet (
     PopupHandle    popupHandle,
     UINT8               newX,
     UINT8               newY );
```

Parameters

popupHandle	Specifies a handle returned from a previous call to **DosVidSaveScreen** or **DosVidPopup**
newX	Logical column within the specified popup where cursor should be positioned
newY	Logical row within the specified popup where cursor should be positioned

Returns

0	Cursor successfully positioned
0xFFFFFFFF	Invalid popupHandle parameter

Remarks

The cursor can be disabled by setting *newX* and *newY* to 0xFF.

See Also

DosVidEmptyTypeAhead

Description

Empties the keyboard typeahead buffer.

Syntax

```
void
DosVidEmptyTypeAhead(
    void);
```

Parameters

None

Returns

Nothing

Remarks

Keyboard BIOS must be callable.

This service is intended to be used only in the context of a popup.

See Also

DosVidGetKey

Description

Waits for a key press and returns the key value. While waiting this function relinquishes control by calling **NiosPoll**.

Syntax

```
#include <dosvmm.h>

UINT16
DosVidGetKey(
      void);
```

Parameters

None

Returns

Lower byte contains ASCII code or other translation
Upper byte contain scan code or special character ID

Remarks

Returned Scan/ASCII codes are compatible with BIOS Int 16h, functions 10h/11h.

Keyboard BIOS must be callable.

This service is intended to be used only in the context of a popup.

See Also

DosVidCheckKey
DosVidEmptyTypeAhead

DosVidGetPopupInfo

Description

Obtains miscellaneous information about an active popup created using either the **DosVidSaveScreen** or **DosVidPopup** service. This information can be used to determine how to write text into the popup.

Syntax

```
#include <dosvmm.h>

UINT32
DosVidGetPopupInfo(
    PopupHandle    popupHandle,
    PopupInfo      *popInfo );
```

Parameters

popupHandle	Handle of popup about which to return info. This handle must be a valid handle returned by **DosVidSaveScreen** or **DosVidPopup**.
popInfo	Pointer to PopupInfo structure which, on return, will be filled with information about the specified popup.

Returns

0	Function successful
0xFFFFFFFF	Invalid popupHandle parameter

Remarks

See Also

DosVidIsPopupOk

Description

Determines if it is possible to display a popup message in the current context.

Syntax

```
#include <dosvmm.h>

SINT32
DosVidIsPopupOk(
      void);
```

Parameters

None

Returns

>=0	Popup ok
<=0	Popup NOT ok

Possible Values - Popup Ok
MS Windows message mode should be used except for DVOK_FULL_DOS_BOX_AVAIL.

DVOK_SYSTEM_VM	Focus VM is system VM
DVOK_WINDOWED_DOS_BOX	Focus VM is windowed DOS box
DVOK_FULL_DOS_BOX_AVAIL	Focus VM is full screen DOS box
DVOK_FULL_DOS_BOX_BUSY	Focus VM is full screen DOS box, video/keyboard BIOS is available
DVOK_FULL_DOS_BOX_GRPH	Focus VM is full screen DOS box inside of Windows, video BIOS is busy

Possible Values - Popup NOT Ok

DVOK_WIN_NOT_FOCUS_VM	Windows active, focus VM is not current VM
DVOK_WIN_AT_HARD_INT	Windows active, at hardware interrupt time

DVOK_DOS_BAD_GRPH_MODE	DOS only, unsupported graphics mode
DVOK_DOS_BIOS_BUSY	DOS only, video BIOS is busy

Remarks

See Also

DosVidPopup

Description

Displays a popup message on the screen with the specified title, subtitle, prompt, and message text.

Syntax

```
#include <dosvmm.h

void
*DosVidPopup(
     UINT8       *titleStr,
     UINT8       *subtitleStr,
     UINT8       *promptStr,
     UINT8       *msg);
```

Parameters

titleStr	Title of popup. This should be a short message. If this parameter is NULL, no title message will be displayed.
subtitleStr	Subtitle of popup. This should be a short message. If this parameter is NULL, no subtitle message will be displayed.
promptStr	Prompt message. This describes some type of user action. If NULL, no prompt will be displayed.
msg	Main popup text. This is a message describing something to the user.

Returns

zero	Error creating popup
non-zero	Popup handle

Remarks

Control is given back to the caller once the popup is displayed. The caller must use the **DosVidRestoreScreen** service to remove the popup message and restore the previous screen contents.

Note that the size of the popup is dynamically calculated. The actual size is based on the length of the passed-in msg string along with the lengths of the title and prompt strings.

See Also

DosVidRestoreScreen
DosVidPopupExt
DosVidWriteToPopup

DosVidGetPopupInfo

DosVidPopupExt

Description

Displays a popup on the screen with the specified title, subtitle, prompt, and message text.

Syntax

```
#include <dosvmm.h>

UINT32
DosVidPopupExt(
     PopupHandle    popupHandle,
     UINT8               *titleStr,
     UINT8               *subtitleStr,
     UINT8               *promptStr,
     UINT8               *msg,
     UINT8               extraLines,
     UINT8               minColumns,
     UINT8               minRows,
     UINT8               popupStartColumn,
     UINT8               popupStartRow);
```

Parameters

popupHandle	Specifies a handle returned from a previous call to **DosVidSaveScreen**.
titleStr	Title of popup. This should be a short message. If this parameter is NULL, no tile message will be displayed.
subtitleStr	Subtitle of popup. This should be a short message. If this parameter is NULL, no subtitle message will be displayed.
promptStr	Prompt message. This describes some type of user action, such as which keys do what. If NULL, no prompt will be displayed.
msg	Main popup text. This is a message describing something to the user.
extraLines	Number of extra empty lines that should be built after the msg string. This allows the caller to dynamically add text to the bottom of the popup using the **DosVidWriteToPopup** service. If zero, no extra lines will be output.

minColumns	Minimum number of user space columns in the popup. If set to zero, the number of columns is dynamically calculated based on the contents of the passed-in string parameters.
minRows	Minimum number of user space rows in the popup. If set to zero, the number of rows is dynamically calculated based on the contents of the passed-in string parameters.
popupStartColumn	Specifies the X coordinate of where the popup will begin on the screen. If set to 0xFF, the popup will be horizontally centered on the screen.
popupStartRow	Specifies the Y coordinate of where the popup will begin on the screen. If set to 0xFF, the popup will be vertically centered on the screen.

Returns

0	Popup successfully created
0xFFFFFFFF	*popupStartColumn* is out of bounds
0xFFFFFFFE	*popupStartRow* is out of bounds
0xFFFFFFFD	*minColumns* is out of bounds
0xFFFFFFFC	*minRows* is out of bounds
0xFFFFFFFB	Not enough free memory to process popup
0xFFFFFFFA	Invalid *popupHandle* parameter

Remarks

Control is given back to the caller once the popup is displayed. The caller must use the **DosVidRestoreScreen** service to remove the popup message and restore the previous screen contents.

This function should be used instead of **DosVidPopup** when more control over the popup's format is needed.

Note: This service requires a *popupHandle* as an input parameter; therefore, the caller must invoke **DosVidSaveScreen** prior to using this service.

See Also

DosVidPopup
DosVidSaveScreen
DosVidRestoreScreen
DosVidWriteToPopup
DosVidGetPopupInfo

DosVidRestoreScreen

Description

Restores the contents of a portion of the screen previously preserved using the **DosVidSaveScreen** or **DosVidPopup** functions.

Syntax

```
#include <dosvmm.h>

void
DosVidRestoreScreen(
      PopupHandle     popupHandle);
```

Parameters

popupHandle Handle of popup returned from **DosVidSaveScreen**

Returns

0	Screen successfully restored, popupHandle freed
0xFFFFFFFF	Invalid popupHandle parameter

Remarks

This function also restores the cursor to the settings present at the time **DosVidSaveScreen** was called.

See Also

DosVidSaveScreen

DosVidSaveScreen

Description

Saves the contents of the specified rectangular portion of the screen (including current cursor information) to a dynamically allocated buffer.

Syntax

```
#include <dosvmm.h>

UINT32
DosVidSaveScreen(
    PopupHandle    *popupHandle);
```

Parameters

popupHandle	Pointer to a pointer which will be set on return to a handle describing the specified saved video region. This handle is used to restore the window contents.

Returns

0	Function was successful
0xFFFFFFFF	Unable to allocate save buffer
0xFFFFFFFE	Bad video mode
0xFFFFFFFD	Bad parameters

Remarks

This function disables the cursor until **DosVidRestoreScreen** is called. This function is provided for NLMs that need to access the display directly when the **DosVidPopup** service does not provide enough functionality.

Before using this service, the caller must verify that a popup is possible by using either the **DosVidIsPopupOk** or **DosVidCallWhenPopupOk** services.

Use of this function allows the caller to gain direct control of where text is placed on the screen. Using the returned *popupHandle*, the caller can use the **DosVidWriteToPopup** and/or **DosVidPopupExt** services to place text anywhere on the screen.

See Also	DosVidRestoreScreen DosVidPopup DosVidIsPopupOK DosVidCallWhenPopupOk DosVidWriteToPopup DosVidGetPopupInfo

DosVidSoundBell

Description

Rings the bell once.

Syntax

```
#include <dosvmm.h>

void
DosVidSoundBell(
      void);
```

Parameters

None

Returns

Nothing

Remarks

This is a synchronous call. In other words, it does not return until the bell is finished. This function enables interrupts and yields by calling **NiosPoll**.

See Also

DosVidStdOut

Description

Displays the specified prefix and message using DOS STDOUT.

Syntax

```
#include <dosvmm.h>

void
DosVidStdOut(
    UINT8      *prefix,
    UINT8      *msg);
```

Parameters

prefix Pointer to message prefix string. If NULL, no prefix will be displayed.
msg Pointer to message string to display.

Returns

Nothing

Remarks

DOS must be callable. This function displays the strings using the STDOUT file handle for the currently active Program Segment Prefix (PSP). This function yields.

See Also

DosVidWriteToPopup

Description

Writes the specified string contents to the specified (column, row) position inside the popup specified by *popupHandle*. The output will be truncated if the string exceeds the popup dimensions.

Syntax

```
#include <dosvmm.h>

UINT32
DosVidWriteToPopup(
    PopupHandle    popupHandle,
    UINT32             column,
    UINT32             row,
    UINT8              attribute,
    UINT32             len,
    UINT8              *str);
```

Parameters

popupHandle	Handle of popup returned from a previous call to either **DosVidSaveScreen** or **DosVidPopup**
column	Logical x position where string should be written to (for example, value of 0 is the left-hand corner of the popup)
row	Logical y position where string should be written to (for instance, value of 0 is the first row of the popup)
attribute	Standard color display attribute describing the background and foreground attributes to use when writing out string
len	Number of bytes in string

Returns

0	String output successfully.
0xFFFFFFFF	Invalid column and/or row parameters.
0xFFFFFFFE	String was output; however, it exceeded the popup size and was truncated.
0xFFFFFFFD	Invalid popup handle.

Remarks

See Also

DosVidSaveScreen
DosVidPopup
DosVidGetPopupInfo

DosWrite

Description

Writes to the specified file.

Syntax

```
#include <dosvmm.h>

UINT32
DosWrite(
    UINT32     fileHandle,
    UINT32     writeOffset,
    void       *writeBuf,
    UINT32     writeSize);
```

Parameters

fileHandle	Handle of file to write to (returned by **DosOpen**). The file must have been opened for writing.
writeOffset	Offset from start of file to write to: 0xFFFFFFFF if write should occur at current position.
writeBuf	Pointer to buffer holding data to write.
writeSize	Number of bytes to write (0 - 0xFFFFFFFC).

Returns

Number of bytes written

0xFFFFFFFF	Seek failed
0xFFFFFFFE	I/O error during write
0xFFFFFFFD	Function failure

Remarks

DOS must be in a callable state. Generally this function can be inside of an NLM's initialization function as well as during an event scheduled using the **DosScheduleDosAvailEvent** service.

See Also

DosClose
DosOpen
DosCreate
DosGetFileSize
DosRead
DosDelete
DosSeek
DosDoesFileExist
DosSearchForFile

Win16GetProcAddress

Description

Resolves the sel:off of an exported 16-bit Windows procedure.

Syntax

```
#include <nlmapi.h>

UINT32
Win16GetProcAddress(
        UINT8      *modName,
        UINT8      *procName,
        UINT32     *procSelOff);
```

Parameters

modName	Name of module, DLL, or application that exports the procName.
procName	Pointer to ASCIIZ procedure name to resolve. If the upper 16-bits of this value is zero, the low order 16-bits are interpreted as an ordinal value.
procSelOff	UINT32 set on return to the sel:off of the resolved Windows function if this service returns successfully.

Returns

Zero	Procedure successfully resolved.
0xFFFFFFFF	Unresolved procName.
0xFFFFFFFE	Serice not currently available. There is a window during MS Windows initialization that this service isn't available. An NLM can watch for the "WIN16 GETPROCADDR AVAIL" event to know when this service is available.
0xFFFFFFFD	Current execution ontext isn't the system VM, or the system VM isn't executing in protected mode.

Remarks

This function calls the Windows function **GetProcAddress** to service the request.

WinCallWhenPMIntReturns

Description

Used in an NLM's PM interrupt handler to obtain control on the back end of a current PM interrupt.

Assumes

edx Reference data
Interrupt in any state.
esi Points to callback handler. Called as follows:

```
        On entry:      ebx  Points to VM CB
                       edx  Reference data
                       ebp  Points to CRS
                       Interrupts are enabled.
                       CLD has been executed.

        On return:  CLD preserved
                       Interrupts are enabled.
                       All registers can be destroyed.
```

Returns

Z flag cleared.
Interrupt state preserved.
All registers can be destroyed.

Remarks

This function is callable only in the context of a protected mode (PM) interrupt under Windows Enhanced mode.

An NLM that uses this service must first call this service then return from its PM interrupt handler signalling that the interrupt was NOT consumed. This service is designed so that the NLM's interrupt handler can simply jump to this service and this service will return back from the handler with the Z flag cleared.

This service places a PM callback address on the current PM stack such that when the PM interrupt handling code iret's out of the interrupt, NLM handlers that have used this service will receive control. This occurs in a LIFO manner thus preserving the ordering that should occur when multiple NLMs hook the backend of the same PM interrupt.

When the handler is invoked, the current CrsCS, CrsIP and CrsFlags will hold the current iret information. If the handler needs to modify the return flags it should do so by modifying the CrsFlagas field.

WinHookPMInt21

Description

Displays the specified prefix and message using DOS STDOUT.

Syntax

```
#include <dosvmm.h>

UINT32
WinHookPMInt21 (
    modHandle    moduleHandle,
    UINT32       referenceData,
    UINT32       (*handler)(
                     VmCb      *vm,
                     UINT32    referenceData,
                     CRS       *crs));
```

Parameters

moduleHandle	Callers module handle.
referenceData	Data to be passed to the handler.
handler	Pointer to int 21h handler to install.

Parameters:

vm	Current vm handle.
referenceData	Data specified during registration.
crs	Pointer to client registers.

Returns:
Zero to consume the interrupt.
Non-zero to chain the interrupt.

Returns

0	on success.
0xFFFFFFFF	if out of memeory.

Remarks

This function can be called when Windows is not active; however, the hook only becomes active after the NE_WIN_SYS_VM_INIT NESL event. The hook becomes inactive once Windows destroys its PM int 21 chain, but it will reactivate the next time Windows loads (after the NE_WIN_SYS_VM_INIT event).

Every call to WinHookPMInt21 must be matched with a corresponding call to WinUnHookPMInt21.

NLMs that need to pass a PM Int 21h interrupt on down the chain and then receive control on the back end of the interrupt can use the WinCallWhenPMIntReturns service inside of their interrupt handler function to obtain this type of functionality.

This function is not available at interrupt time.

See Also

WinUnHookPMInt21
WinCallWhenPMIntReturns

WinUnHookPMInt21

Description

Displays the specified prefix and message using DOS STDOUT.

Syntax

```
#include <dosvmm.h>

UINT32
WinUnHookPMInt21 (
    modHandle     moduleHandle,
    UINT32        (*handler)(
                      VmCb      *vm,
                      UINT32    referenceData,
                      CRS       *crs));
```

Parameters

moduleHandle	Callers module handle.
handler	Pointer to int 21h handler to uninstall.

Parameters:

vm	Current vm handle.
referenceData	Data specified during registration.
crs	Pointer to client registers.

Returns:
Zero to consume the interrupt.
Non-zero to chain the interrupt.

Returns

0	on success.
0xFFFFFFFF	if handler is not registered by moduleHandle..

Remarks

See Also

WinHookPMInt21
WinCallWhenPMIntReturns