Chapter 7
NIOS APIs for Windows95

Windows95 Interface

The NIOS Win32 API interface provides a high-performance Ring 3 (User) to Ring 0 (System) access mechanism which allows Win32 applications to invoke and use most exported NLM API functions.

This interface is not available for Windows v3.1x Win32s. The **DeviceIoControl** Win32 API function is not supported on Win32s.

Note: This interface will only be provided on versions of Windows95 and above that are based on a Vxd driver architecture. This interface is NOT and will not be provided on Windows NT. Win32 applications that wish to also run on NT should insulate themselves from this API by using a DLL.

The following steps are used by a Win32 application to gain access to the NIOS services.

Step 1: Locate the NIOS driver by using the Win32 **DeviceIoControl** API function. This is accomplished by first opening the NIOS device using code similiar to the following:

```
devHandle = CreateFile(
         "\\\\.\\NIOS",
         GENERIC_READ | GENERIC_WRITE,
         FILE_SHARE_READ | FILE_SHARE_WRITE,
         NULL,
         OPEN_EXISTING,
         FILE_ATTRIBUTE_NORMAL,
         NULL );

if ( devHandle == INVALID_FILE_HANDLE)
             NIOS driver isn't loaded error
```

Step 2: If NIOS is present, invoke the Win32 **DeviceIoControl** API function to obtain two function addresses that will be used to issue further requests to the NIOS interface. You must provide the address of a **NiosWin32EntryPoints** structure which will be filled out by NIOS on return.

An example call to DeviceIoControl is:

```
NiosWin32EntryPoints NIOS;
```

```
DeviceIoControl(
       devHandle,
       WIN32_GET_NIOS_INTERFACE,
       NULL,
       NULL,
       &NIOS,
       sizeof( NiosWin32EntryPoints),
       NULL,
       NULL );
```

Step 3: Use **NiosWin32EntryPoints.Win32NiosFarCall** to issue various requests to NIOS, such as resolving the addresses of NLM API functions, locking memory,etc. Use **NiosWin32EntryPoints.Win32InvokeCNlmApi** to invoke NLM API functions.

The following is an example using this interface to invoke the NIOS exported function **NiosGetVersion**:

```
   Resolve the API function we want to call.

NiosGetVersionAddr =
       NIOS.Win32NiosFarCall(
                WIN32_NIOS_BEGIN_USE_API,
                NiosGetVersion");

  Call the NLM function.

NiosVer = NIOS.Win32InvokeCNlmApi(
                NiosGetVersionAddr,
                0);

   Tell NIOS we're no longer using the
   function.

NIOS.Win32NiosFarCall(
                WIN32_NIOS_END_USE_API,
                NiosGetVersionAddr);
```

For more information, see the **Win32NiosFarCallHandler** and **Win32InvokeCNlmApi** function calls in Chapter 7, *NIOS APIs for Windows95*.

Win32InvokeCNlmApi

Description

Calls (Invokes) an exported NLM function that uses the "C" calling conventions. This function is called by Win32 applications.

Syntax

```
UINT32
(*Win32InvokeCNlmApi)(
            UINT32    nlmApiAddress,
            UINT32    apiParmCount,
            ...);
```

Parameters

apiAddress	Address of NLM API to invoke. This is a value obtained from the WIN32_NIOS_BEGIN_USE_API function.
apiParmCount	Number of UINT32 stack parameters needed for call. This value defines the number of UINT32 values that need to be copied from the application's stack onto the Ring 0 protected mode stack prior to invoking the specified NLM API.
...	Parameters to NLM API.

Returns

Return code defined by the NLM API.

Remarks

The steps required for a Win32 application to gain access to NIOS services are listed in Chapter 2 under the heading *Windows 95 Interface*.

See Also

Win32LoadModule

Description

Loads an NLM when called by Win32 applications.

Syntax

```
UINT32
Win32LoadModule (
          UINT32        loadOptions,
          UINT8      *modulePathSpec,
          UINT8      *commandLine,
          UINT32        nlmFileOffset,
          modHandle     *retModHandle,
          void          (*msgHandler)(
                        modHandle    module,
                        UINT8        *prefix,
                        UINT8        *msg)  );
```

Parameters

loadOptions	Bits defining loading styles. All undefined bits must be set to 0. LOPTION_DEBUG_INIT Executes a Int 1 before the loader invokes the module's init routine. LOPTION_ERROR_MSGS Stdout error messages are enabled LOPTION_BANNER_MSGS Stdout signon messages are enabled
modulePathSpec	[path\]name of module to load (with extension).
commandLine	Pointer to any parameters that will be passed to the loading module. This is a ASCIIZ string.
nlmFileOffset	Offset from the start of the modulePathSpec file where the NLM image starts. Typically this will be 0 for straight .NLM files.
retModHandle	Pointer to a modHandle that will be set to the newly loaded module's handle on success. If NULL the module handle will not be returned.

msgHandler	Address of function which will be called when a text message is displayed during the load process. Parameters to this function are Ring-0 linear addresses, therefore the handler should use the appropriate NIOS functions to copy the memory.

Returns

LOADER_SUCCESS	Module was loaded successfully
LOADER_NO_LOAD_FILE	Open load file failed
LOADER_IO_ERROR	File I/O error during read
LOADER_INSUFFICIENT_MEMORY	Not enough memory to load module
LOADER_INVALID_MODULE	Invalid NLM module
LOADER_UNDEFINED_EXTERN	Referenced undefined external item
LOADER_DUPLICATE_PUBLIC	Exported public is already defined.
LOADER_NO_MSG_FILE	Open msg file failed
LOADER_INVALID_MSG_MODULE	Msg file is malformed
LOADER_MODULE_ALREADY_LOADED	Module cannot be loaded more than once
LOADER_BAD_REENTRANT_MODULE	Reentrant load failed because the module is not the same version as the first module.
LOADER_MODULE_INIT_FAILED	Module failed to initialize.
LOADER_LOAD_REFUSED	A loaded NLM refuses to allow this NLM to load.

Remarks

All input pointer parameters are local Win32 application memory addresses.

Windows applications that need to load an NLM typically will use this function instead of **NiosLoadModule** since they will want to obtain text ouput messages from the NLM and loader while the load is taking place. It is possible to invoke **NiosLoadModule** with the LOPTION_ERROR_MSGS and LOPTION_BANNER_MSGS set to 0 from a Windows application since this causes a silent load to take place.

All pointer parameters passed to this function do NOT need to be mapped using the WIN32_NIOS_MAP service.

The steps required for a Win32 application to gain access to NIOS services are listed in Chapter 2 under the heading *Windows 95 Interface*.

See Also

Win32NiosFarCallHandler

Description

Invokes NIOS services. This function is called by Win32 applications.

Syntax

```
UINT32
(*Win32NiosFarCall)(
            UINT32    function,
            ...);
```

Parameters

function	WIN32_NIOS_???? value. Refer to NLMAPI.H and NLMAPI.INC
...	Other parameters as needed.

Returns

Values specific to each function.

0x80000000	Invalid function request value.

Remarks

The steps required for a Win32 application to gain access to NIOS services are listed in Chapter 2 under the heading *Windows 95 Interface*.

See Also

WIN32_NIOS_BEGIN_USE_API
WIN32_NIOS_COPY_MEM
WIN32_NIOS_COPY_STRING
WIN32_NIOS_END_USE_API
WIN32_NIOS_MAP
WIN32_NIOS_UNMAP

Win32UnloadModule

Description

Unloads as NLM when called by Win32 applications.

Syntax

```
UINT32
Win32UnloadModule (
        modHandle    modHand,
        UINT32       unloadOptions,
        void         (*msgHandler)(
                     modHandle    module
                     UINT8        *prefix,
                     UINT8        *msg) );
```

Parameters

modHandle	Module to unload. This is a flat linear address of a module handle for the NLM to unload.
unloadOptions	Bits defining unload options. All undefined bits must be set to 0. UOPTION_ERROR_MSGS Stdout error messages are enabled
msgHandler	Address of function which will be called when a text message is displayed during the unload process. Parameters to this function are Ring-0 linear addresses, therefore the handler should use the appropriate NIOS functions to copy the memory.

Returns

UNLOAD_SUCCESS	Module was unloaded
UNLOAD_MODULE_FORBIDS_UNLOAD	Module doesn't allow unload
UNLOAD_MODULE_BEING_REFERENCED	Another module is using this module.
UNLOAD_INVALID_MODULE_HANDLE	Module handle is invalid
UNLOAD_RESOURCES_NOT_FREED	Module didn't free resources
UNLOAD_MODULE_CANT_UNLOAD_NOW	

	Module is temp. unable to unload
UNLOAD_UNLOAD_REFUSED	A loaded NLM refuses to all this NLM to load.

Remarks

All input pointer parameters are local Win32 application memory addresses.

The steps required for a Win32 application to gain access to NIOS services are listed in Chapter 2 under the heading *Windows 95 Interface*.

See Also

WIN32_NIOS_BEGIN_USE_API

Description

Determines the 32-bit flat linear address of the specified NLM API name.

Syntax

```
UINT32
(*Win32NiosFarCall)(
            UINT32     WIN32_NIOS_BEGIN_USE_API,
            UINT8 *apiName);
```

Parameters

apiName	Name of the API you want to call. This is a case insensitive ASCIIZ string. For example, *NiosGetVersion*. This pointer does not need to be mapped using WIN32_NIOS_MAP.

Returns

Zero	API does not exist.
Non-zero	Linear address of API

Remarks

After determining the 32-bit linear address, the returned address can then be used with the **Win32InvokeCNlmApi** entry point to actually invoke the NLM function from a Win32 application. This function records a dependency for the NLM module that the API function exists in, therefore it is important that the Windows application use the WIN32_NIOS_END_USE_API function before terminating.

The steps required for a Win32 application to gain access to NIOS services are listed in Chapter 2 under the heading *Windows 95 Interface*.

See Also

Win32NiosFarCall
WIN32_END_USE_API

WIN32_NIOS_COPY_MEM

Description

Copies the contents of memory at the specified Ring zero linear address into the specified Ring three buffer for the given length.

Syntax

```
void
(*Win32NiosFarCall)(
            UINT32    WIN32_NIOS_COPY_MEM,
            void      *destBuffer,
            UINT32    ring0Buffer,
            UINT32    length);
```

Parameters

destBuffer	Ring 3 application buffer to copy to. This ptr does not need to be mapped using WIN32_NIOS_MAP.
ring0Buffer	Linear address of Ring 0 buffer to copy from.
length	Number of bytes to copy.

Returns

Nothing

Remarks

The steps required for a Win32 application to gain access to NIOS services are listed in Chapter 2 under the heading *Windows 95 Interface*.

See Also

WIN32_NIOS_COPY_STRING

Description

Copies the string pointed to by the Ring zero *pmBuffer* address into the specified Ring three application buffer.

Syntax

```
void
(*Win32NiosFarCall)(
            UINT32    WIN32_NIOS_COPY_STRING,
            void      *destBuffer,
            UINT32    ring0Buffer);
```

Parameters

destBuffer	Ring 3 application buffer to copy to. This ptr does not need to be mapped using WIN32_NIOS_MAP.
ring0Buffer	Linear address of Ring 0 string to copy.

Returns

Nothing

Remarks

The steps required for a Win32 application to gain access to NIOS services are listed in Chapter 2 under the heading *Windows 95 Interface*.

See Also

WIN32_NIOS_END_USE_API

Description

Signals that the Windows application is no longer going to use the specified NLM API function.

Syntax

```
void
(*Win32NiosFarCall)(
            UINT32    WIN32_NIOS_END_USE_API,
            UINT32    apiLinAddress);
```

Parameters

apiLinAddress	Linear address of NLM API function.

Returns

Nothing

Remarks

This function deletes the dependency previously created using WIN32_NIOS_BEGIN_USE_API.

The steps required for a Win32 application to gain access to NIOS services are listed in Chapter 2 under the heading *Windows 95 Interface*.

See Also

Win32NiosFarCall
WIN32_BEGIN_USE_API

WIN32_NIOS_MAP

Description

Converts the specified linear address local to a calling Win32 process into a globally accessible linear address range which can be accessed int eh context of any process, including hardware interrupt time.

Syntax

```
void
*(*Win32NiosFarCall)(
          UINT32    WIN32_NIOS_MAP,
          void      *appMemPointer,
          UINT32    length);
```

Parameters

length	Length of memory block to map. It is important that the proper length of the memory block be passed into this function, otherwise portions of the memory block may be left unlocked, and unaliased.

Returns

Zero	Invalid *appMemPointer*, or not enough free physical memory available to comple the operation.
Non-zero	Global/locked linear address for the memory.

Remarks

Tthis function also page locks the memory (makes and keeps it present). Page locking is necessary to keep the linear address ranges mapped to the same physical memory locations, as well as allowing the memory to be safely accessed at interrupt time.

The returned linear address from this function can be used as a memory pointer parameter to any exported NLM function.

In general, memory passed to an NLM API function should be mapped using this function. However there are exceptions. To properly take advantage of these exceptions it is necessary to understand which catagory the NLM API function fits in.

Simple functions

These functions access memory pointer parameters synchronously. This means that the function does not directly or indirectly (by passing them to other functions) access the memory in any other context other than the caller's process/memory context. Generally these are simple functions that perform a primitive operation.

Examples of this type of function are: **NiosGetSystemDirectory**, **NiosMemCpy**, **NiosStrCpy**, **NiosPrintf**, etc.

It can be difficult to qualify whether a function uses memory parameters in this manner. In general if the NLM API function documentation does not explicity say that memory parameters can be application local/non-locked memory, then the Win32 application should use the WIN32_NIOS_MAP function.

Realize that most NLM API functions were written to be called at Ring 0 with locked/global memory parameters.

Mapper functions

This type of NLM API function was developed to be callable from a Win32 application. The NLM API developer may develop the API in one of two ways. One is to require that the calling Ring 3 application lock and globalize memory parameters prior to invoking the function. The other is where the NLM API function has been developed to properly handle non-locked/local memory parameters and performs the necessary locking/aliasing internal to the function, therefore freeing the application from having to deal with locking/globalizing issues.

In all cases, if a Win32 application developer wishes to skip calling the WIN32_NIOS_MAP / WIN32_NIOS_UNMAP services for a particular NLM API function, then the developer must ensure that the NLM API function is capable of handling local/non-locked memory parameters. If you don't know whether the function can except local memory parameters, find out, otherwise always lock and globalize memory parameters. The reason this is so important is that if you go ahead and pass local memory to a function that can't handle it, the function may work properly 99% of the time, however if the system state is just right the local memory parameter will cause a system failure.

Block Memory Issues

If a memory block needs to be mapped using this function there are a few other issues that need to be understood.

Because this function page locks the specified memory block physical memory is committed to the memory block. This prevents the system virtual memory pager from using the physical memory for other needs to uses in the system. Therefore, it is important that a Win32 application not keep a large amount of memory locked down (mapped) at any one time.

Performance considerations weighed against the negative impacts of having too much memory locked will dictate when a particular memory block is locked and unlocked in the lifetime of the application.

On the one extreme where the highest performance is needed, memory blocks can be locked down (mapped) when the application starts and unlocked (unmapped) when the application terminates. This frees the application to use the memory blocks as parameters to NLM API functions at anytime in the applications lifetime without having to incur the overhead of locking and unlocking each time the memory block is used.

The other extreme is where a memory block is locked down (mapped) immediatelly before invoking the NLM API function, then unlocked (unmapped) right after the NLM API function returns.

An obvious inbetween policy to use when a memory block is used in a set of calls to NLM API functions is to lock the memory block at the start of the operation, invoke n number of NLM API functions using the locked memory, then unlock the memory block after the set of calls have completed. This minimizes the overhead of the lock/unlock operations and also the actual time the memory is kept locked down to reasonable levels.

It is left up to the Win32 application developer to decide which is the best policy or mix of policies to use.

Unmapping Memory

Memory that is mapped using this function must be subsequently unmapped when it's no longer going to be used, such as when the application terminates. It is extremely important that mapped memory be unmapped.

The appMemPointer is a Win32 application memory pointer which needs mapped.

The steps required for a Win32 application to gain access to NIOS services are listed in Chapter 2 under the heading *Windows 95 Interface*.

See Also

Win32NiosFarCall
WIN32_NIOS_UPMAP

WIN32_NIOS_UNMAP

Description

Unlocks the application memory block and destroys the global linear range alias created through a previous call to the WIN32_NIOS_MAP service.

Syntax

```
void
*(*Win32NiosFarCall)(
            UINT32    WIN32_NIOS_UNMAP,
            void      *globalPointer,
            UINT32    length);
```

Parameters

length	Length of memory block to unmap. It is important that this value equal the length value passed to the WIN32_NIOS_MAP function.

Returns

Non-zero	Unmap operation successful.
Zero	Invalid *globalPointer* and/or length parameter.

Remarks

The *globalPointer* Address was returned from a previous call to WIN32_NIOS_MAP.

The steps required for a Win32 application to gain access to NIOS services are listed in Chapter 2 under the heading *Windows 95 Interface*.

See Also

Win32NiosFarCall
WIN32_NIOS_MAP