How To Write An NLM for the DOS/Windows Environment

Revision 1.0 Draft 1.2
March 1995

Preliminary Version

NetWare®

Disclaimer

Novell, Inc. makes no representation or warranties with respect to the contents or use of this manual, and apecifically disclaims any express or implied warranties of merchantability or fitness for any particular purpose. Further, Novell, Inc. reserves the right to revise this publication and to make changes to its content, at any time, without obligation to notify any person or entity of such revisions or changes.

Further, Novell, Inc. makes no representations or warranties with respect to any NetWare software, and specifically disclaims any express or implied warranties of merchantability or fitness for any particular purpose. Further, Novell, Inc. reserves the right to make changes to any and all parts of NetWare software, at any time, without obligation to notify any person or entity of such changes.

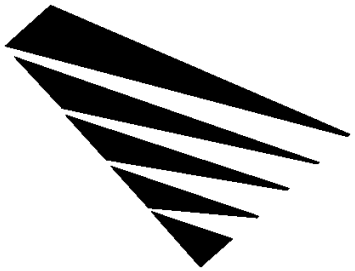
Trademarks

Novell, Inc. has made every effort to supply trademark information about company names, products, and services mentioned in this document. Trademarks were derived from various sources.

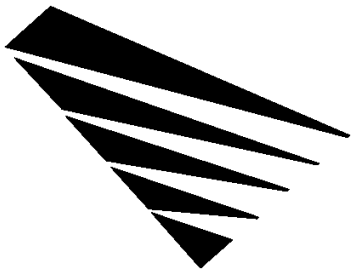
Copyright

Novell, Inc.
122 East 1700 South
Provo, Utah 84606 U.S.A.

Revision 1.0 Draft 1.2
March 1995 Edition



Contents



Preface

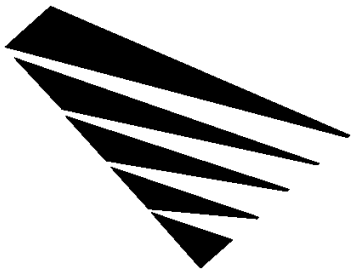
This is a document in the process of development. As such it is not yet the definitive work on writing client NLMs. It does, however, contain valuable information which we feel will help you begin building your own client NLMs.

Future editions will be greatly expanded and filled with programming helps from the very development engineers who created the concept of the client NLM here at Novell.

Any helpful suggestions would be gratefully accepted. Please call Keith Newman at (801) 429-7370.

This document assumes you have access to the following Novell publications.

- *NetWare Client NIOS Design Specification*
- *NetWare Client NIOS for DOS, MS Windows, and Windows 95 Design Specification*



Chapter 1
Introduction to Client NLMs

NetWare Loadable Modules

Originally, NetWare Loadable Modules (NLMs) were defined as programs built to run in server memory with the NetWare OS. Now, however, this definition has been expanded to include the DOS/MS Windows and Windows 95 environments.

This chapter introduces the client NLM by explaining many of the similarities and differences between server and client NLMs. This chapter also summarizes the services NIOS makes available to client NLMs.

Similarities between Server and Client NLMs

Client NLMs link with and add functionality to NIOS just as server NLMs link with and add functionality to the NetWare OS. In fact, client NLMs were designed to be as much like server NLMs as possible. There are, therefore, many similarities between server and client NLMs. These similarities include, but are not limited to, the following.

- Server and Client NLMs can be written entirely in C.
- Server and Client NLMs have the same executable format. (See Figure 1.1.)
- Server and Client NLMs use the same API export and import mechanism.
- Server and Client NLMs use the same language enabling tools.
- Server and Client NLMs can be loaded and unloaded as needed.

- Server and Client NLMs can use other NLMs to provide services that are needed.

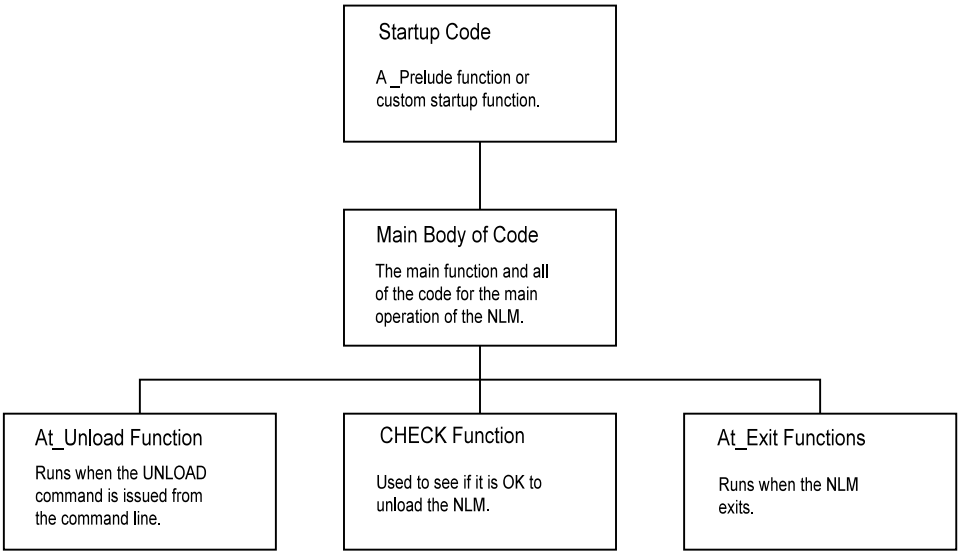


Figure 1.1: NLM Structure

Differences between Server and Client NLM Environments

Even though there are many similarities between the environments that NLMs exist in, there are also some significant differences.

- Currently there is no support for compression for on the client, but the capability will be added before v1.0 ships.
- NetWare server threads are not supported on the client.
- CLIB is not currently available on the client.
- Most of the server API functions are not available on the client.

NIOS Services

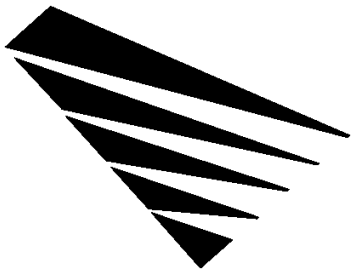
Developers may have many reasons to write their applications as a client NLMs, but certainly one reason would be to take advantage of the powerful services made available to client NLMs through the NetWare I/O Subsystem (NIOS).

NIOS provides a protected mode execution environment for DOS and Windows. In MS Windows, an NLM behaves like a Vxd having full access to Vxd services; in return, Vxds have direct access to NLMs. Furthermore, NIOS provides a consistent view of protected mode, whether or not DOS or Windows is active.

Services provided by NIOS can be categorized as follows.

- Configuration Services
- Debug Services
- Event Services
- Handle Management Services
- Hardware Interrupt Services
- Information Services
- Linked List Services
- Module Management Services
- Memory Management Services
- Popup Video Services
- Process Management Services
- Returnable Memory Management Services
- Statistics Services
- Time/Date Services
- User Interface Services
- Utility Services

For more information about specific services, see the *NetWare Client NIOS Design Specification*.



Chapter 2
Getting Started

This chapter explains the requirements for loading the NetWare NIOS software on your workstation. The directions assume that you are familiar with the following operating systems and hardware.

- PC-DOS, MS-DOS, DR-DOS, or Novell DOS 7 (released version),
- Microsoft Windows 3.1 Enhanced mode (Protected mode)
- MS Windows 95
- NetWare 3 or 4 compatible OS LAN drivers (4.1 drivers preferred)
- NetWare compatible network boards
- Memory management software (minimally HIMEM.SYS)

If you are planning to use the new MS Windows 95 operating system, you may skip to the section titled “Installing the Windows 95 Client”.

Preparing Your Workstation Hardware

Before attempting to install the NetWare NIOS software, you should ensure the workstation meets the following hardware requirements.

- An Intel 32-bit CPU (80386 or higher).
- A hard drive with 1.2 MB of storage or a 1.2 MB floppy drive and diskette.
- A network board installed in the workstation. If you need to install the board, be sure to record the board settings for use during installation. For information on installing the network board, see the manufacturer’s documentation.
- A cable connection to the network. Each type of network board requires unique cabling. See the manufacturer’s documentation packaged with your network board for requirements.

Note: Token ring network boards require a cable connection to the MAU before installing the operating system. Otherwise, the TOKEN driver will not load.

Determining Your Network Board Settings

The NetWare OS LAN driver requires specific information about the network board installed in your workstation. Record the values for the following settings before loading the NIOS software.

- Hardware interrupt. You should use an interrupt that is not already in use in your system. Interrupts commonly used for network boards are IRQ3 or IRQ5. See the manufacturer's documentation for more information.

- Base I/O port. Each hardware device installed in your workstation must have a different base I/O port setting. For more information on the base I/O port for your network board, see the manufacturer's documentation.

- Media frame type. All client workstations and servers using a single network address must use the same frame type. See the section titled "Determining Your Network Frame Type" in this chapter.

- Other settings. There may be other settings unique to the network board installed in your workstation. See the documentation provided with your network board for more information.

  Note: In most cases, you should leave your network board set to the factory default settings. If you need to change the default settings, see the manufacturer's documentation.

You can obtain setting information for your network board by using the following procedures.

- If you have an EISA or MCA network board, run the computer's setup or reference program. This program lists the values for your network board settings.

- If you have an ISA network board, look at the network board itself to obtain the specific settings. The documentation provided with your network board will direct you where to find each setting value.

- If you have a network connection already, type NVER at the command line to view the board settings.

Determining Your Network Frame Type

You can determine your network frame type by using the following procedures.

- If you already have a network connection, read the frame type parameter for the LINK DRIVER option in your NET.CFG file. NIOS will eventually do this automatically.

- If you a installing on a new workstation, ask your system administrator for the frame type for your particular network.

Note: The default frame type for Ethernet ODI drivers has changed. In NetWare 2 and 3, NetWare OS Ethernet LAN drivers defaulted to Ethernet 802.3 frame type. In NetWare 4, the default frame type has changed to Ethernet 802.2.

If you use the Ethernet 802.2 frame type on your workstation, it cannot connect to a network expecting the Ethernet 802.3 frame type. To eliminate a potential conflict, you can define both frame types (Ethernet 802.2 and Ethernet 802.3) on your network.

Setting Up NIOS

Once you have the proper hardware and software installed as described in the preceding sections of this chapter, you can install and configure the NetWare NIOS software. To do this, you must do the following:

- Obtain the appropriate NetWare 4 compatible OS LAN driver for your network board. NetWare drivers are available from a release of NetWare 4.x.

- Obtain the NetWare NIOS Client software.

Copying NIOS Files

Copy the following NIOS files to your NetWare client directory; the default directory name is NWCLIENT.

- NIOS.EXE
- LSL.NLM

- MSM.NLM
- ETHERTSM.NLM or TOKENTSM.NLM
- lan_driver.LAN (LAN driver for your network board.)
- IPX.NLM
- POLYPROC.NLM
- NSMUX.NLM
- SESSMUX.NLM
- CONNMAN.NLM
- NCP.NLM
- TASKMAN.NLM
- BINDERY.NLM
- NDS.NLM
- MOCKNW.NLM
- FILEDIR.NLM
- PRINT.NLM
- NETX.NLM
- VLMMAP.NLM

Creating A Batch File

Create a batch file for loading the NIOS software using the following procedures.

1. Load an ASCII editor with which to create the batch file. Save this file in the same directory that you copied the NIOS software files to.

2. Add a line for loading the NIOS abstraction layer, by typing

   NIOS

3. Add a line for loading the Link Support Layer, by typing

   LOAD LSL

4. If you are using NetWare 3.12 and 4.x OS LAN drivers, add a line for loading support for Novell's Ethernet or Token-ring Topology Specific Module (TSM), by typing

   LOAD ETHERTSM (For Ethernet LAN drivers)
   or
   LOAD TOKENTSM (For Token-ring LAN drivers)
   or
   LOAD FDDITSM (For FDDI LAN drivers)

5. Add a line for loading the LAN driver, by typing

LOAD *lan_driver* [driver_parameters]

Replace *lan_driver* with the appropriate driver for your network board.

Warning: You must use NetWare 4.x LAN drivers. Drivers for NetWare 3.11 are not compatible.

LAN driver parameters may be required or optional. Currently you must enter a value for each parameter setting if you have configured your network board with non-default settings. You must specify the new value at the command line when you load the driver.

Note: Future implementations of the client software will enable the LAN driver to prompt you for all required parameter settings.

Most drivers use one or more of the following parameters.

DMA =number
INT=number
NAME=board name
PORT=number
SLOT=number
FRAME=name
MEM=number
NODE=number
RETRIES=number

The following line is a sample LAN driver command for an NE2000 network board.

```
   Load ne2000 int=3 port=300 mem=DC000
frame=ethernet_802.3
```

Note: The NetWare 4 OS LAN drivers default to the ETHERNET_802.2 frame type. Previous versions defaulted to ETHERNET_802.3. Contact your system administrator for details.

Note: NE3200 users must specify the POLL=0 command line option when loading the NE3200.LAN driver for proper operation.

6. Add a line for loading IPX support, by typing

   LOAD IPX

7. Add lines for loading the NLMs that make up the 32-bit client shell, by typing

```
LOAD POLYPROC
LOAD NSMUX
LOAD SESSMUX
LOAD CONNMAN
LOAD NCP
LOAD TASKMAN
LOAD BINDERY
LOAD NDS
LOAD MOCKNW
LOAD FILEDIR
LOAD PRINT
LOAD NETX
LOAD VLMMAP
```

Sample Batch File

The following is a sample batch file for loading the NIOS and NetWare client software.

```
nios
load lsl
load msm
load ethertsm
load ne2000 int=3 port=300 mem=DC000 frame=ethernet_802.3
load ipx
load polyproc
load nsmux
load sessmux
load connman
load ncp
load taskman
load bindery
load nds
load mocknw
load filedir
load print
load netx
load vlmmap
```

Installing the Windows 95 Client

Note: The following procedures are temporary. An application for installing the client is currently in development.

Begin installing the Windows 95 client using the following procedure.

- Copy the NIOS.VXD file to your Windows system directory. For example, if you installed Windows 95 to the default C:\WINDOWS directory, NIOS.VXD would be copied to C:\WINDOWS\SYSTEM.

- Add the following to your SYSTEM.INI file under the [386Enh] section.

  ```
  device=nios.vxd
  nwhomedir=c:\nwclient
  ```

  Set NWHOMEDIR to where your NetWare client files (*.NLM and NET.CFG) are located; the default directory is C:\NWCLIENT.

- You can have NIOS autoload DEBUG.NLM by copying DEBUG.NLM to the C:\WINDOWS directory. Otherwise you can load it manually.

- Load the NLM files.

Three methods currently exist to install the NLM files under Windows 95.

- Manually load the NLMs in a DOS box after Windows 95 is up and running.

- Manually load the NLMs in the Windows 95 GUI environment by using the NIOSCTL.EXE Windows application.

- Create a WINSTART.BAT file in your Windows directory and list individual entries to load each NLM. Windows executes the WINSTART.BAT immediately before the GUI desktop is initialized. An example WINSTART.BAT is shown below.

  ```
  LOAD LSL
  LOAD MSM
  LOAD ETHERTSM
  LOAD NE2000.LAN int=5 port=320
  LOAD IPX
  LOAD POLYPROC
  LOAD NSMUX
  LOAD SESSMUX
  ```

```
LOAD CONNMAN
LOAD NCP
LOAD TASKMAN
LOAD BINDERY
LOAD NDS
LOAD MOCKNW
LOAD FILEDIR
LOAD PRINT
LOAD NETX
LOAD VLMMAP
F:LOGIN servername/username
```

Other NIOS Issues

Unloading Client Software

The NetWare Client NLM software can be removed from memory by using the UNLOAD command, for example:

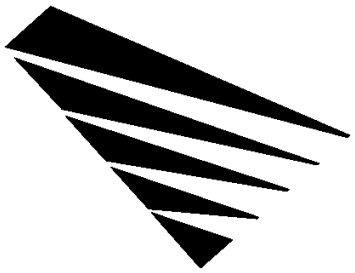
```
UNLOAD LSL
```

Compatibility With NOVDOS HIMEM.SYS

Due to a bug in the HIMEM.SYS (v2.3) driver in Novell DOS 7, you must load DOS into the HMA area if running the NIOS software on a Compaq machine. Add the following line to your CONFIG.SYS file, by typing

```
DOS=HIGH,UMB
```

You can also disable the HIMEM.SYS driver and load the EMM386.EXE file included with Novell DOS 7 instead.



Chapter 3
Client NLM Code Development

This chapter describes the basic layout of a NetWare Loadable Module (NLM) along with the tools and techniques used to build one.

The following data types are defined and used extensively in the client NLM environment:

```
#define   UINT32    unsigned int
#define   SINT32    signed int
#define   UINT16    unsigned short int
#define   SINT16    signed short int
#define   UINT8     unsigned char
#define   SINT8     signed char
```

NLM Required Functions

Initialization Function

An NLM must define and implement a minimum of two functions. The first function handles the module's initialization. It is invoked immediately after the NLM has been loaded into memory. This routine is invoked as a "C" function with a number of input parameters (discussed in detail later in this section).

All "C" functions in an NLM environment must preserve registers EBX, ESI, EDI, and EBP. On return the initialization function must signal in register EAX whether or not the module initialized successfully. A return value of 0 signals that the module successfully initialized and should remain resident in the system, otherwise a non-zero return value signals a failed initialization in which case the loader will remove the NLM from the system.

The initialization function is invoked as follows:

```
UINT32    ModuleInit(
       modHandle myModHandle,
       modHandle unusedScreenHandle,
       UINT8 *commandLine,
       UINT8 *moduleLoadPath,
       UINT32 unitializedDataLength,
       UINT32 customDataFileHandle,
       UINT32 (*readProc)(
          UINT32 customFileHandle,
          UINT32 customOffset,
          UINT8 *buf,
          UINT32 bytesToRead),
       UINT32 customDataOffset,
       UINT32 customDataSize,
       UINT32 numMsgs,
       UINT8 **msgs)
```

myModHandle — This parameter is the loading NLM's module handle which remains valid throughout the module's lifetime. Each active NLM is assigned a unique module handle. The handle is actually a pointer to a structure. Many of the fields in this structure are publicly defined and available for the NLM or other NLMs to look at (refer to NIOS.H or NIOS.INC for a definition of this structure). In most cases this structures follows the structure used by the NetWare OS, however there are a few minor differences. Many system calls require the module handle as an input parameter and is often used to assign and track ownership of system resources.

unusedScreenHandle — In the NetWare OS environment this parameter contains a handle to the system console screen. This is needed if the module wants to display information during its initialization sequence. In the client environment this parameter is not needed and is set to equal the NLM's module handle (myModHandle).

commandLine — This parameter points to an ASCIIz (Zero terminated) string containing any parameters specified when the NLM was loaded.

moduleLoadPath — This parameter points to an ASCIIz string containing the path from which the NLM was loaded from. This string does not include the name of the NLM.

unitializedDataLength	Not used in the client environment.
customDataFileHandle	The custom data file is appended to the end of an NLM by the NLM link. NetWare loads only the NLM's code data at load time, leaving the file open for the NLM to handle custom data. This handle points to a structure that the operating system uses to read the custom data.
readProc	Used to allow reading of a custom data file linked with the NLM. This is a pointer to the custom data **readProc** (see Appendix A).
customDataOffset	This is the starting offset of the custom data inside the NLM file.
customDataSize	This parameter contains the length of the custom data file.
numMsgs	This parameter is NOT passed to NLM's running in the NetWare server environment. In the server message information is obtained using the **ReturnMessageInformation** API (supported on the client and server). This is the number of language enabled messages present in the *msgs* parameter.
msgs	This parameter is NOT passed to NLM's running in the NetWare server environment. In the server message information is obtained using the **ReturnMessageInformation** API (supported on the client and server). This parameter is a pointer to an array of (UINT8 *) pointers. The index or message number that is used to access specific messages is assigned by the NLM language enabling tools.

Deinitialization (Unload) Function

The second required function is the module's deinitialization function, also referred to as the module's exit or unload handler. This function is invoked when a request has been made to the system to remove the NLM. The loader calls the exit function using "C" conventions. No input parameters are defined for the exit routine, and a return code is not used or defined.

The function must return any resources it has allocated and place itself into a nonactive state. Note that if an NLM's initialization function returns non-zero, meaning that the NLM will NOT stay resident, the NLM's exit function is NOT invoked. For completeness, the prototype for an NLM's unload function is:

```
UINT32    ModuleDeinit( void)
```

NLM Optional Functions

An optional third function, called the *check* procedure, can be implemented to allow an NLM to determine whether or not it is safe or desired to allow the NLM to unload. The check function is invoked by the loader when a request has been made to remove the NLM from memory. This function has no input parameters, is "C" callable, and must return 0 if it is okay to unload, otherwise the loader will refuse the request to remove the NLM, in which case the NLM's exit function will NOT be invoked.

Module Handle

NLM resource tracking is implemented in the NetWare server using *resource tags*. In the client NLM environment, resource tags are not supported or implemented. Resource tracking in the client is simpler and uses less overhead than resource tags. Basically when a resource is allocated by an NLM, the resource provider simply increments the *resourceCount* field in the caller's module handle structure. When the resource is freed, the count is decremented.

When an NLM unloads from the system, this count is checked and if it is not 0 then the loader displays a warning informing the user that the NLM didn't free all of its allocated resources. Most NIOS services require the caller's module handle as an input parameter. It is recommended that other NLMs exposing services to other NLMs increment and decrement the resourceCount field as resources are allocated and freed.

The *resourceCount* field is not implemented or supported in the NetWare server environment, therefore an NLM that is targeted to run in both environments should use conditional compile techniques to handle the differences in the way resources are tracked.

NIOS presents primitive functions which allows Ring 3 applications (DOS, Windows 16-bit, Win32) the ability to invoke exported Ring 0 NLM functions. Because many Ring 0 NLM functions require a module

handle as an input parameter and the fact that Ring 3 programs are not NLMs and do not have an NLM module handle, NIOS provides services which allow the Ring 3 application the ability to allocate a pseudo module handle which can be used in calls that require a module handle parameter (See NiosCreateModuleHandle and NiosDestroyModuleHandle).

Development Tools

This section describes the tools available to create an NLM executable. Basically the tools can be grouped into one of three catagories: Assemblers, "C" compilers, and linkers.

Assemblers

In the past the traditional Intel assembler used to create NLMs has been Pharlap 386ASM. Although this assembler is still widely used, the 32-bit client development team recommends the use of Borland's Turbo Assembler based on the facts that its faster, cheaper, and more versatile than 386ASM. If you choose to use 386ASM, refer to the NetWare OS NLM SDK for further information.

Borland's Turbo Assembler (TASM) comes in two flavors, TASM.EXE and TASMX.EXE. TASMX uses extended memory and allow assembly of large source files. TASM, and TASMX for that matter, must be invoked differently depending on which linker you choose to use. If you are using the Watcom WLINK linker you should use the following command line switches:

```
tasmx /m /q /mx myprog.asm
```

/m Tells TASM to use as many passes necessary to resolve all references and forward references. Although not required, use of this parameter generates the best machine code. In particular this eliminates the need to specify the *short* keyword on jump instructions to generate the shortest jump encoding. With /m the assembler uses the shortest jump encoding without inserting NOP instructions.

/q Tells the assembler to not include miscellaneous object records in the .OBJ file that are not needed for linking.

/mx Tells the assembler to use case sensitivity for global symbols. This parameter is not required although it is recommended.

If you are using the Novell linker (NLMLINKX) you should have the following switches and commands:

tasmx /m /q /mx /op myprog.asm,myprog.ob1;
borfix myprog.ob1 myprog.obj

/m /q /mx As defined above.

/op Tells the assembler to generate an .OBJ compatible with the Pharlap .OBJ standard.

Because the object records are not entirely comatible with NLMLINKX you must run the object file through the BORFIX utility before linking.

In general an assembly language source file used with TASM should contain the following directives, segment definitions, etc. Note that you can use whatever segment names you prefer, the names used below are for illustration only.

```
.386p     ; Enable protected mode 386 instructions.

          assume cs:OSCODE, ds:OSCODE, es:OSDATA, ss:OSDATA
OSDATA    segment USE32 READWRITE public 'DATA'

   YourData     db ?

OSDATA    ends

OSCODE    segment USE32 EXECREAD public 'CODE'

MyProc    proc   near
   CPush
   ; ... Your code
   CPop
   ret
MyProc    endp

OSCODE    ends
          end
```

"C" Compilers

The two "C" compilers available are: Watcom "C" 9.00 or above, and Metaware "C". Watcom is the most popular compiler used for NLM

development. Other "C" compilers, or even other language compilers such as C++, Pascal, etc. should work as well as long as they can generate 32-bit flat memory model code.

wcc386p /fo=myprog.obj myprog.c /ez /ot /3s /w4 /zp1 /zl /s

-3s 386 stack calling conventions
-ez generate PharLap EZ-OMF object files
-s remove stack overflow checks
-zl remove default library information
-zp1 pack structure members on 1 byte boundaries.

Linkers

In general two linkers exists that generate NLM executables. One is from Novell, called NLMLINKX, and the other from Watcom called WLINK which is include with their "C" compiler. Usage information for NLMLINKX is available in the NetWare OS SDK and also by typing NLMLINKX with no parameters.

The following sample link statements below show how to configure WLINK to generate an .NLM.

wlink @myprog.def

Contents of myprog.def:

```
format novell nlm 'My Programs Banner'
debug
name myprog.nlm
sort
option start=MyProgInit
option exit=MyProgDeinit
option map=myprog.map
option copyright '(C) Copyright 1995 My Company, Inc.
All Rights Reserved.'
option version=0.01.1
```

Please refer to the Watcom WLINK users guide for more information about WLINK and the options used in the above example.

Chapter 4
Client NLM Example

This chapter includes two versions of the famous Hello World program written as NLMs. The program demonstrates the basic NLM program structure.

Hello World Header

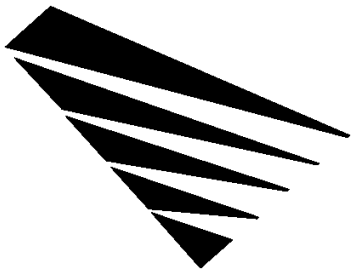
```
/*****************************************************************************
*
*     (C) Unpublished Copyright of Novell, Inc. All Rights Reserved.
*
*     No part of this file may be duplicated, revised, translated,
*     localized or modified in any manner or compiled, linked or uploaded
*     or downloaded to or from any computer system without the prior
*     written consent of Novell, Inc.
*
******************************************************************************
*
*     NetWare IO Subsystem (NIOS) Sample Hello NLM Include File
*
*
*****************************************************************************/

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// Declare Exportable Definitions Here
//

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// Declare Exportable Data Structures Here
//

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// Declare Exportable API Functions Here
//
```

Hello World Example

```
/****************************************************************************
*
*   (C) Unpublished Copyright of Novell, Inc. All Rights Reserved.
*
*   No part of this file may be duplicated, revised, translated,
*   localized or modified in any manner or compiled, linked or uploaded
*   or downloaded to or from any computer system without the prior
*   written consent of Novell, Inc.
*
****************************************************************************
*
*   NetWare IO Subsystem (NIOS) Sample Hello NLM
*
*
****************************************************************************/

#include <nios.h>
#include <debug.h>
#include <nstdlib.h>
#include <module.h>
#include "hello.h"


//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// Local Function prototypes
//
UINT32 IpxInit(
       modHandle myModuleHandle,
       modHandle unusedScreenHandle,
       UINT8 *commandLine,
       UINT8 *moduleLoadPath,
       UINT32 unitializedDataLength,
       UINT32 customDataFileHandle,
       UINT32 (*readProc)(
              UINT32 customFileHandle,
              UINT32 customOffset,
              UINT8 *buf,
              UINT32 bytesToRead),
       UINT32 customDataOffset,
       UINT32 customDataSize,
       UINT32 numMsgs,
       UINT8 **msgs);
void HelloDeinit (void);


//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// HelloInit
//
//     First routine called when HELLO.NLM is loaded.
//
//  Entry:
//     myModuleHandle             Module handle to be used in module oriented NIOS calls
//     unusedScreenHandle
//     commandLine                -> ASCIIZ command line to module
//     moduleLoadPath             -> ASCIIZ full path of the module being loaded
//     unitializedDataLength
//     customDataFileHandle
//     readProc
//     customDataOffset
```

```
//     customDataSize
//     numMsgs                    Number of messages in msgs variable
//     msgs                       -> array of localized messages for this module
//
// Exit:
//     0      Load was successful.
//     !0     There was an error.  Module load is to be aborted.
//
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

#pragma off (unreferenced)    // Disable unused parm warnings for a moment

UINT32 HelloInit(
       modHandle myModuleHandle,
       modHandle unusedScreenHandle,
       UINT8 *commandLine,
       UINT8 *moduleLoadPath,
       UINT32 unitializedDataLength,
       UINT32 customDataFileHandle,
       UINT32 (*readProc)(
              UINT32 customFileHandle,
              UINT32 customOffset,
              UINT8 *buf,
              UINT32 bytesToRead),
       UINT32 customDataOffset,
       UINT32 customDataSize,
       UINT32 numMsgs,
       UINT8 **msgs)

#pragma on (unreferenced)     // Now allow warnings.

{
   UINT32 i;

   // Print a little message
   for (i=0; i<10; i++)
      NiosPrintf (
         myModuleHandle,
         MT_INFORM,
         "Welcome to the world of NLM programming!\n\r");
   NiosPrintf (
      myModuleHandle,
      MT_INFORM,
      "\n\r\n\r");

   // Tell the NLM loader abort the load of this NLM
   return (1);
}


//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// HelloDeinit
//
//     This needed by the NLM Linker specified as the exit routine.
//     NOTE:  Since this NLM automatically aborts the load, this
//     routine should never be called.  However, the way NLMs are
//     supposed to work is that they load, stay resident, and then
//     unload at a later time.  This is the reason an exit procedure
//     is necessary.
//
// Entry:
//     Nothing
```

```
//
// Exit:
//     Nothing
//
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

void HelloDeinit (void)
{
    // NOT SUPPOSED TO GET HERE!
}
```



Chapter 5
Debugging Your Client NLM

This chapter discusses the tools and procedures you can use to debug your client NLMs in both the DOS and MS Windows environments. This chapter also includes information about NIOS Logging and read-only page protection under MS Windows 3.1.

Debugging Client NLMs in DOS

Currently the NIOS development team uses the WDEB386.EXE Microsoft debugger (available from the MS Windows DDK) to debug NIOS client NLM software in a DOS only environment. You can use the DEBUG.NLM program in conjunction with the WDEB386 program for debug information.

Using Microsoft's Windows Kernel Debugger

To use the WDEB386 program, add a line to your CONFIG.SYS file after all entries that load resident memory managers. The WDEB386 program requires at least one parameter setting for the program to initialize itself. Configure the WDEB386 program by adding any desired parameter to the line in your CONFIG.SYS file.

You can view available parameters for the WDEB386 program by typing WDEB386 at a DOS command line with no parameter settings. A help screen appears showing the various options available on the command line.

WDEB386 requires a terminal attached to the host PC using a NULL modem cable. Novell recommends that you use the /R:19200 option to boost the baud rate from the default 9600 to 19200.

The following line is an example of the line you should add to your CONFIG.SYS file.

```
DEVICE=C:\NWCLIENT\WDEB386.EXE /D:"y386env;ydislwr;codebytes;"
        /R:19200
```

Settings after the /D parameter are optional.

To use WDEB386 for DOS debugging on a system that runs Windows, you must comment out the VCD.386 device in the Windows SYSTEM.INI file.

Using Debug.NLM

To obtain symbol information and other useful debugging services, the DEBUG.NLM file must reside in the same directory as the NIOS.EXE file. Doing this allows the DEBUG.NLM program to be autoloaded when you run the NIOS.EXE program.

You can also manually load the DEBUG.NLM file, typing LOAD DEBUG after running the NIOS.EXE program.

Accessing Debug Functionality

There are three ways to access the debug functionaltiy in the NIOS software.

- Place an INT1 or INT3 inside of your NLM code.
- Use the debug hotkey Ctrl-F12.
- Use the /D option when loading your NLM. If the /D parameter is specified on the command line when loading a Client NLM, then the loader will break into the debugger immediately before calling the initialization function for the NLM.

Accessing Help in the Debugger

Once inside the debugger you can use the ? command to obtain help on all of the available debugger commands. The .NIOS command can be entered to gain access to commands made available by the DEBUG.NLM program. The ? command can be used after entering the .NIOS command to obtain help on the DEBUG.NLM commands.

Debugging NLMs and Real Mode Code

The debugger allows debugging of 32-bit NLM software and 16-bit real mode code. When specifying an address you can use the ampersand (&) modifier to specify a *segment:offset* address, a number sign (#) modifier to specify a *selector:offset* address, a percent sign (%) to specify a linear address, and a double percent (%%) modifier to specify a physical address. For example, “d & 112:0" display memory at real mode segment 112 offset 0. The debugger’s prompt character is different depending on which processor mode you are in (“>” or “_” for real mode and “#” for protected mode).

Debug register breakpoints are set using the BR command, and software breakpoints are set using the BP command. WDEB386 support full regular expression jump conditions off of ‘br’ also. For example, bp 80400000 “j by (&ebp + &ld)’dw ebp + 1c l1';g” instructs the debugger to stop when the byte at linear address ebp + 1Dh is greater than 0B3h and display AX on the stack , otherwise continue.

Note: The syntax is: bp address “j {expression> [’] cmds;cmds... [’];[’] cmds;cmds... [’]” (This syntax is complex, but can be very useful.)

Trace messages can be displayed on the debugger terminal by using either the **NiosDprintf** service.

Compatibility with Soft-ICE for DOS

The Nu-Mega’s Soft-ICE for DOS program requires some special conditions to coexist successfully with NIOS. You must load the S-ICE.EXE program in your CONFIG.SYS file with the “/EMM 1024" option. This enables the VCPI host interface in Soft-ICE to coexist peacefully with the NIOS software

Note: Due to bugs in earlier versions of the S-ICE.EXE program, use Soft-ICE v2.60 or later. Earlier versions will not function properly. Due to bugs in v2.60, you should disable DOS’s use of the HMA. Do this by removing the DOS=HIGH entry in your CONFIG.SYS. You cannot use Soft-ICE to debug DOS and MS Windows Client NLM software, only real mode programs.

Occasionally when using the NIOS software with Soft-ICE, some key strokes will appear dropped or garbled. This is a problem with the way the Soft-ICE program handles receiving-keyboard hardware interrupts from protexted mode. We are anticipating that Nu-Mega will correct this problem in future releases of their product.

Note: The Soft-ICE for Windows program disables its VCPI interface when the disables its VCPI interface when the BREAK ON command is entered. Therefore, the NIOS software cannot be used with the BREAK ON facility.

Debugging Client NLMs within MS Windows

To debug Client NLM software when running MS Windows in Enhanced mode, you must load an additional debugger when starting MS Windows. Use either Microsoft's WDEB386.EXE or Nu-Mega's Soft-ICE for Windows program (WINICE.EXE). Even if you have added a line for loading the WDEB386 program in your CONFIG.SYS file, you must load an additional debugger program when running within MS Windows. After exiting MS Windows, any active debugger program you had loaded before running MS Windows will be reactivated.

Note: If you are using the WDEB386 program in both CONFIG.SYS and for an MS Windows session, make sure you use the same command line options when loading WDEB386 the second time within MS Windows.

Using WDEB386 in DOS and MS Windows

If you load the WDEB386 program in your CONFIG.SYS file and load it again within MS Windows, you must remove the VCD Vxd from your SYSTEM.INI file. There is an incompatibility when loading two copies of the WDEB386 program and VCD.

To remove the VCD Vxd from your SYSTEM.INI file, open the SYSTEM.INI file in an ASCII text editor and enter a semicolon (;) character in front of the "device=*vcd" entry.

The incompatibility with dual loading of the WDEB386 program and VCD Vxd does not occur on all machines. If MS Windows hangs when booting or exiting, the incompatibility probably exists. This problem can also occur when loading the WDEB386 program in the CONFIG.SYS file running the WINICE program within MS Windows. Generally, when debugging within MS Windows, you should remove VCD.

Using DEBUG.NLM Within MS Windows

When the DEBUG.NLM program is loaded, the debug query commands are available. However, if you are using the WINICE program, the debug query commands will function only if you have installed a Microsoft Windows debug kernel that is larger than 1M, such as the WIN386.EXE file. If you use the WDEB386 program in MS Windows, the debug query functions work properly even while using the retail version of the Windows kernel.

To allow dot (.) commands to work with WINICE and the retail version

of Windows, simply patch WINICE.EXE at the following offsets, depending
on the version you are using. Patch the offsets with 0x90 and 0x90.

v1.10	0x20CDB,0x20CDC
v1.20	0x21EAA,0x21EAA
v1.30	0x2384C,0x2384D
v1.32 (Win 3.1)	0x239C7,0x239C8
v1.32 (Chicago)	0x31924,0x31925
v1.41	0x24A13,0x24A14

If you are using a version not listed above, search the WINICE.EXE file for the following byte pattern. Patch the second instance of this pattern as noted by the xx xx.

CD 22 3D 86 F3 00 00 xx xx

For v1.50 and higher patch the following offsets with the specified byte values.

v1.50	0x11624	31,C0,90,90,90

Note: The .NSTACK command will not function automatically when using the WINICE program, you need to provide the address of the current ESP. However, this command will work automatically when using the WEDB386 program.

NIOS Logging

The NIOS logging feature is a tool that enables developers and users to store or view diagnostic or debugging messages in a well known central location. NIOS logs messages to a file call NIOS.LOG in the NIOS system directory. The sytem directory is either the directory from which NIOS.EXE is loaded or, in the case of MS Windows 95, the *NwHomeDir* specified in SYSTEM.INI [386Enh].

Logging can be enabled in the following ways.

- Load NIOS.EXE with /L (DOS).
- Specify *NwEnableLogging=TRUE* in SYSTEM.INI [386Enh] for (Windows95).

- Type “ENABLE LOGGING” from the DOS prompt after you load NIOS.

- Call the API **NiosEnableLogging** with the parameter NIOS_LOG_ENABLE.

Logging can be disabled in the following ways.

- Logging is off by default.

- Type “DISABLE LOGGING” from the DOS prompt after you load NIOS.

- Call the API **NiosEnableLogging** with the parameter NIOS_LOG_DISABLE.

The maximum size of the logfile can be controlled with the NIOS LOG MAX SIZE field in NET.CFG; the default is 64k.

If logging is enabled, the following types of messages will be timestamped and logged.

- **NiosPrintf** with the MT_LOG_STATUS message type (always).

- **NiosPrintf** with the MT_DEBUG_OUT message type (if no debugger).

- **NiosDprintf** (if no debugger).

You can suppresss the timestamp by using the MTF_NO_TIMESTAMP flag (not available with **NiosDprintf**).

Logging is available to all modules (including internal NIOS modules) after NIOS’s internal memory module has initialized. This service is not available at interrupt time; if called at interrupt time, the message is discarded.

Page Protection Under Windows 3.1

Due to some incompatibilities (notably with Win32s), read-only page protection is turned off under Windows 3.1. You can, however, force it to remain on by setting FORCE WIN WRITE PROTECT = ON under the NIOS section in your NET.CFG.

By default NIOS's NLM code segment write protection is disabled. This is due to incompatibilities with this feature and NOVDOS's memory manager. To enable this feature, simply enter WRITE PROTECTION = ON under the NIOS section in your NET.CFG.

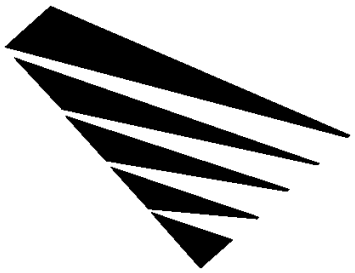
Note: The information in this section applies only to the debug version of NIOS.EXE.

Statistical Tools

NIOS provides two very useful statistical tools NLMSTATS.NLM and NSTATS.EXE. Though their name are similar, their uses are quite different.

NLMSTATS is an NLM that provides valuable information about the NLMs which are loaded in memory. NLMSTATS allows the developer to view information about memory usage, NESL event registration, and configuration parameter values.

NSTATS allows the developer to look at NLM statistics tables, counters, values, etc.; the NLM must make use of the NIOS statistics registry. (See *NetWare Client NIOS Design Specification* for information about NIOS statistics functions). You can use this program to view not only these statistics but other statistics available in the system.



Appendix A
readProc

readProc

Description

Allows the NLM to read custom data.

Syntax

```
UINT32    (*readProc)(
              UINT32    customFileHandle,
              UINT32    *customDataOffset,
              UINT8     *buf,
              UINT32    bytesToRead );
```

Parameters

customFileHandle	The file handle, supplied as *customDataFileHandle* to the NLM's initialization routine.
customDataOffset	The starting offset in the file, supplied as *customDataOffset* to the NLM's initialization routine.
buf	The location of where to read the file to.
bytesToRead	The amount of the data to read, supplied as customDataSize to the NLM's initialization routine.

Returns

Completion Codes (EAX)

0	Successful
nonzero	Failure

Remarks

Note: This routine can only be called during initialization. This routine could go to sleep and could return with interrutps enabled.

readProc allows NLMs to read any custom data that they may require into system memory during initialization. Before the NLM calls this routine, it must allocate memory for the file to be rea.

This routine's entry point is not exported by the operating system. The only time this entry point is valid is during the initialization routine. In fact, the entry point is passed as a local parameter and must be called indirectly.

The NLM linker actually appends the custom data file to the NLM in the .NLM file. NetWare only loads the NLM's code data at load time, leaving the file open for the NLM to handle custom data however it must.

To define the custom file, use the CUSTOM keyword in the NLM's definition file, followed by the file's name. The NetWare operating system passes the custom file's handle, starting address, and size to the NLM's initialization routine. NetWare also passes the address of the **readProc** routine. The NLM's initialization routine can then read the file into memory by calling **readProc**.