MAY 1994

NOVELL® RESEARCH

The Functions and Operations of the NetWare DOS Requester 1.1

John Froelich
Software Test Engineer Associate
VLM Development Team

Ed Liebing
Senior Editor
Systems Research Department

Brad Young
Product Support Engineer
Worldwide Customer Services and Support

This AppNote explains the DOS Requester's theory of operations as well as some noteworthy differences between the NetWare Shell and the DOS Requester. Next, the AppNote looks at how the DOS Requester initializes and how it performs a read request. Finally, example settings of workstation NET.CFG files help explain performance and memory considerations.

Novell, Inc.
122 East 1700 South
Provo, Utah 84606 USA

Contents

Introduction
- What's New in the DOS Requester 1.1

The DOS Requester Theory of Operations
- The VLM Manager
- Multiplexors
- Child VLMs
- Categories of Services
  - The DOS Redirection Layer
  - The Service Protocol Layer
  - The Transport Protocol Layer

How the Shell and the DOS Requester Work with DOS

How the VLMs Initialize
- How the DOS Requester Performs a Read Request

DOS Requester Performance Considerations
- DOS Requester Performance Examples
  - Example 1: Optimization Considerations
  - Example 2: Memory Considerations
  - Example 3: The Best Compromise

Conclusion

Acknowledgements

We would like to thank Bart Reese, Ian Stiles, and Jay Sevison for their assistance with the technical aspects of this AppNote.

Introduction

In April of 1993, Novell Research published its special NetWare 4.0 edition of *NetWare Application Notes.* The issue contained a rather lengthy write-up on the new DOS Requester 1.0 that shipped with NetWare 4. Since then, the DOS Requester has gone through a number of enhancements to improve its overall performance and functionality, and has now been upgraded to version 1.1.

This AppNote takes the next step in explaining how the DOS Requester works. The Introduction first lists the new features that come with the DOS Requester 1.1. The AppNote then discusses the DOS Requester's theory of operations and how the NetWare Shell and the DOS Requester work with DOS. We then examine how the DOS Requester initializes and how it performs a read request.

Finally, we look at three NET.CFG files to show some different ways you can configure the DOS Requester. The sample files emphasize performance optimization, memory optimization, and a balance between the two.

What's New in the DOS Requester 1.1

The following is a list of the new features added to the DOS Requester since the release of NetWare 4.01:

- Connection timeout optimization
- Personal NetWare client functionality
- Network Management Responder (NMR.VLM)
- Optimized Packet Burst over WAN and other slow links
- IPX short-lived socket support
- New NET.CFG parameters for increased flexibility
- Performance optimization for WAN connections
- Overall optimized performance and software compatibility

The DOS Requester Theory of Operations

A VLM is a modular program that contains logically grouped features in order to perform logically grouped functions. For example, transport-related functions such as sending a packet and receiving a packet fit logically into the IPXNCP.VLM module.

The DOS Requester architecture includes various categories of services and components. Individual architectural pieces include the following:

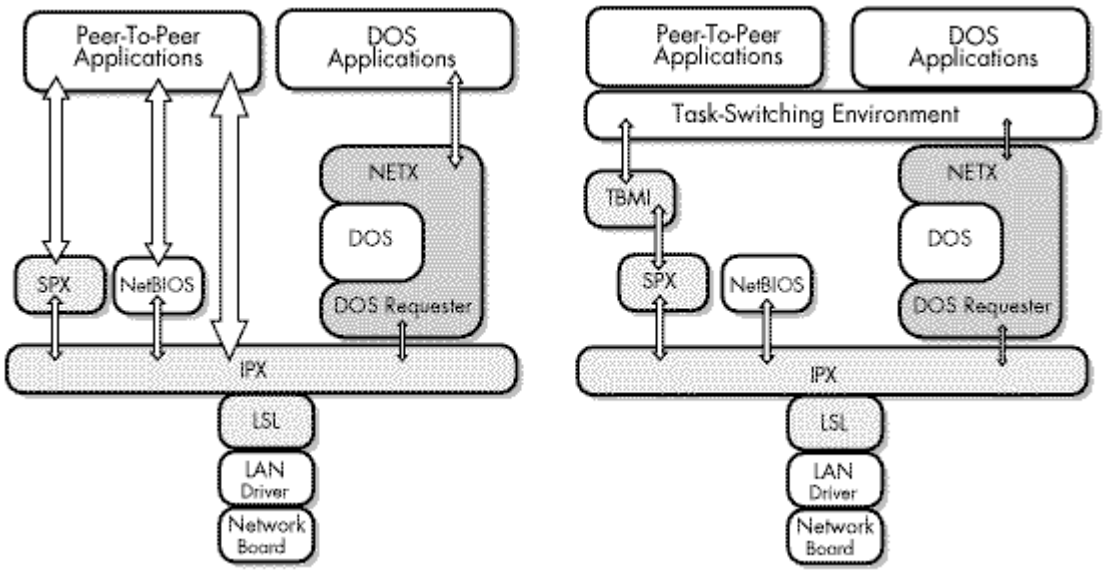
- The VLM Manager (VLM.EXE)
- Multiplexor VLMs
- Child VLMs
- "Standard" VLMs

VLM.EXE is a memory manager that coordinates and controls the VLMs at each of the Requester layers. VLM.EXE loads and unloads the VLM modules and handles memory services for the modules.

A *multiplexor* VLM module provides a common API interface to dissimilar child VLM modules. A *child* VLM module contains the logical grouping of functionality and loads prior to its multiplexor module. A *standard* VLM module offers functionality that does not fall into the category of a multiplexor or a child VLM module. An example of a standard module is AUTO.VLM, which offers automated reconnection capabilities.

The DOS Requester interacts directly with the ODI network communications modules and with DOS. All of these pieces comprise the overall networking environment. This concept is graphically represented in both task-switching and non-switching environments in Figure 1.

Figure 1: The DOS Requester interacts with DOS in both task-switching and non-switching environments.

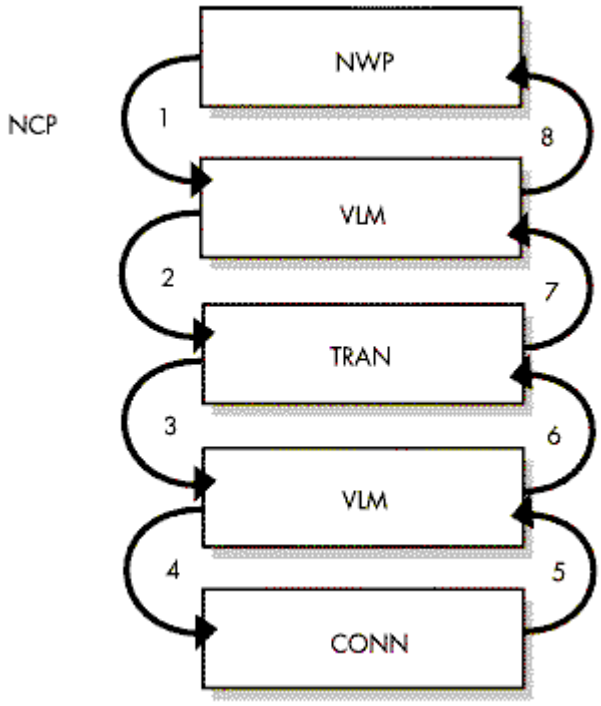


The VLM Manager

The VLM Manager oversees the operations of the VLM modules and handles memory management services for them. Applications making calls to the DOS Requester have *all* their calls routed through VLM.EXE, which directs the requests to their proper destinations. VLM.EXE also ensures that replies return to their respective callers.

The VLM Manager ensures that the API calls between other modules are properly routed. The VLM Manager must therefore know if the modules required to satisfy an API call are loaded into memory. It must police VLM modules that call other modules and handle asynchronous calls to the VLMs and their related functions. Even when a child module calls its multiplexor, the call goes to the VLM Manager, which then calls the specified multiplexor.

Figure 2: The VLM Manager ensures that API calls between other modules are properly routed.



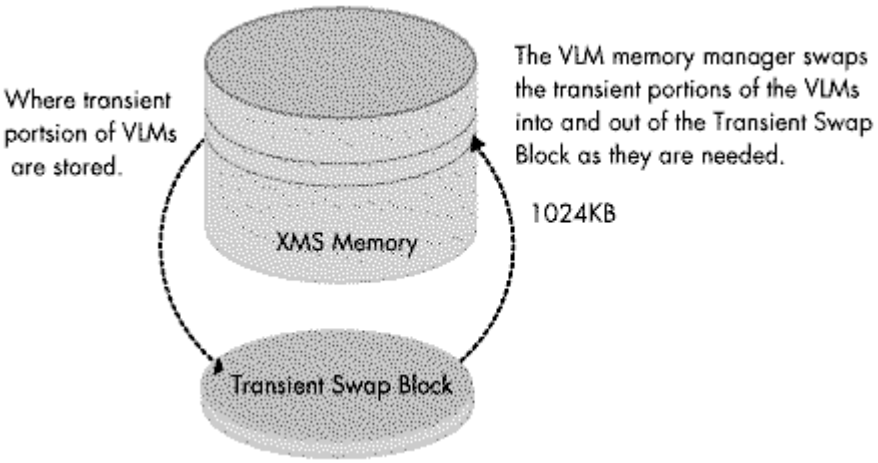
The VLM Manager provides the API interface to call a function and a VLM module by its assigned number. For example, when a request calls for a module, the caller passes the caller's ID (0 for an application, a non-zero for a VLM), the destination VLM's ID, and the function number.

Modules that are not part of the DOS Requester model can also use the VLM Manager as a TSR memory manager. NMR.VLM, the NetWare Management Responder module, is a good example of a module that uses VLM.EXE in this way.

The VLM Manager decides whether the VLM modules use extended memory, expanded memory, conventional memory, or any memory type supported. This frees the VLM modules from memory model decisions and concerns.

The VLM Manager handles *transient block swapping* and uses it to swap the transient segments of the VLM modules into and out of XMS or EMS memory. Each VLM has a portion of its code (called the global segment) that must be maintained in conventional memory, such as interrupt handlers, ESRs (Event Service Routines), buffers passed as pointers, and asynchronous event handlers. Lesser-used function calls are located in the transient segment and are placed into XMS or EMS memory (if you are loading one of these memory managers). These are swapped back into the transient swap block in conventional memory or in upper memory when the module's functions need to be executed. The module is then swapped back out to XMS or EMS memory when another module is placed in the transient swap block. The effect of swapping the transient portion back out is that transient data that may have changed is preserved.

Figure 3: The VLM Manager uses module swapping to swap transient portions of VLM modules into and out of memory.



Transient block swapping allows for a much smaller conventional memory footprint, with much of the lesser-used code residing in a swapped-out state (usually XMS or EMS) until it is required. Swapping, therefore, helps balance UMB (upper memory block) and conventional memory use with performance.

The VLM Manager is also responsible for load time configuration APIs. For instance, a user may want CONN.VLM to have 16 connections in its connection table, or FIO to support a larger cache size and a larger number of buffers. In these instances, the VLM Manager reads pertinent information from the NET.CFG file, returns the configuration data to the specific VLM, and expands or contracts the VLM accordingly.

Multiplexors

Multiplexors can be understood as "parent" VLMs and route calls to their registered child VLMs. Multiplexors insulate an application from the idiosyncrasies of a child VLM's services (such as login or attachment variations between Directory Services, bindery-based servers, and Personal NetWare).

The NWP.VLM module acts as the parent multiplexor to NDS.VLM, BIND.VLM and PNW.VLM. If a workstation issues a call to attach to a bindery server, NWP.VLM routes the call to BIND.VLM for processing. On the other hand, if a workstation makes a call to attach to *any* type of server (referred to as a "wildcard" call), NWP.VLM traverses the child VLMs in the order they were loaded until one of the modules can satisfy the call.

Child VLMs

Child VLMs are different implementations of a logical grouping of functionality. Each server type has its own child VLM:

- BIND.VLM for bindery-based/pre-NetWare 4 servers
- NDS.VLM for NetWare 4 Directory Services-based servers
- PNW.VLM for Personal NetWare-based servers

Different implementations of transport protocols can also have individual VLMs. For example, IPXNCP.VLM handles IPX services, and future releases may include a VLM to handle TCP/IP services, with both modules using the TRAN.VLM (the transport protocol) multiplexor.

Note: Although NETX.VLM is the VLM's functional replacement for NETX.COM or NETX.EXE, IPXNCP.VLM is not a replacement for IPXODI.COM. In fact, IPXNCP.VLM requires that

IPXODI.COM or IPX.COM be loaded.

The load order of child VLMs can also determine the behavior of "wildcard" calls. If a workstation loads the BIND.VLM module before NDS.VLM, the preferred attachment to a server for the "wildcard" AttachToServer call will be a bindery type as opposed to NDS, even if you are only logging in to a NetWare 4 server.

You also see this type of behavior when using PREFERRED SERVER or PREFERRED TREE statements in the NET.CFG or when using /PS= or /PT= parameters when you load VLM.EXE at the command line. If BIND.VLM is loaded first, a PREFERRED SERVER parameter overrides a PREFERRED TREE parameter. If NDS.VLM is loaded before BIND.VLM, the opposite occurs.

For this reason, the DOS Requester 1.1 contains the NET.CFG parameter Netware Protocol = <vlm>[, <vlm>...], which provides an easy way to maintain these scenarios. For example, Netware Protocol = BIND,NDS places bindery services before Directory Services. (You can also exclude any of the child VLM modules of NWP.VLM by not including that module in the list.)

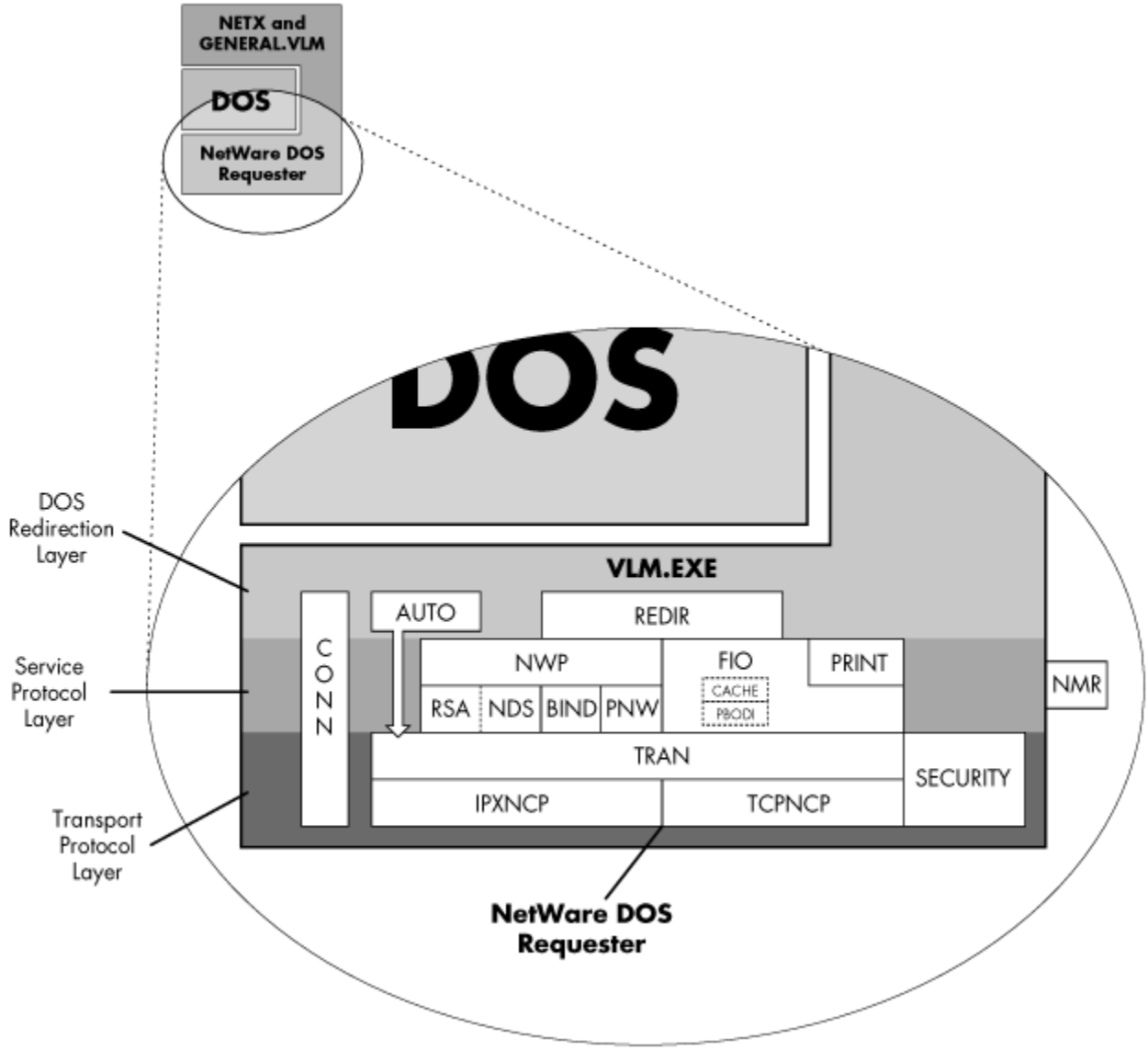
Categories of Services

Each VLM operates at a different protocol layer with a specific category of service. These categories of services form three distinct layers in the Requester architecture:

- The DOS Redirection Layer
- Service Protocol Layer
- Transport Protocol Layer

Modules at a given layer do not need to be concerned with a particular configuration that resides below them.

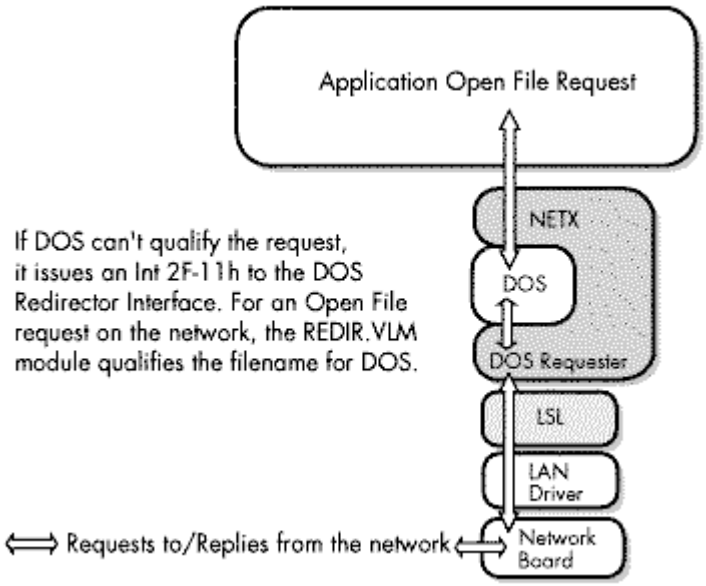
Figure 4: VLM services form three distinct layers in the Requester architecture.



The DOS Redirection Layer. This layer provides DOS redirection services for the DOS Requester with most of the functionality bundled into the REDIR.VLM module.

With the DOS Requester, DOS uses the same mechanism to recognize a network drive as it does to recognize CD-ROM drives. CD-ROM drivers use the DOS Redirector interface to make the drive available to the workstation. Network redirectors use the same interface to make file servers available for clients. REDIR.VLM provides the redirector support for the DOS Requester and makes the NetWare server's resources appear as DOS resources to the user.

Figure 5: The REDIR.VLM module provides redirector support for the DOS Requester.



The Service Protocol Layer. The Service Protocol Layer includes parallel protocols, file services, and print services. The NWP.VLM module is the NetWare protocol portion for the DOS Requester and consists of the NDS-, bindery-, and Personal NetWare-based server types. The FIO.VLM module is the file transfer portion and handles NetWare file I/O traffic. The print handler portion, the PRINT.VLM module, provides printer redirection services. RSA.VLM is not loaded by default, and is only needed for auto-reconnect services to a directory services-based server. Utilities such as LOGIN.EXE take care of initial authentication services to directory services-based servers.

NWP.VLM is the NetWare protocol multiplexor that handles the network-server implementations, which include the Directory Service module (NDS.VLM), bindery module (BIND.VLM), or Personal NetWare module (PNW.VLM). Services provided by the NWP.VLM module include establishing and destroying connections, as well as logins and logouts.

The file input/output module (FIO.VLM) also resides within the Service Protocol Layer and provides a basic file transfer protocol. The module's additional capabilities include cached or non-cached read/writes, Packet Burst mode-based read/writes, Large Internet Packet support, and others. The ability to switch these features on or off allows users to choose performance over memory usage, or vice versa. The PRINT.VLM module provides network printing services. Because PRINT.VLM uses FIO.VLM for its file writes, the printing services can be handled in the following ways:

- Non-cached
- Cached
- Redirected via Packet Burst protocol to the server

The FIO.VLM modules gives the PRINT.VLM module added performance by allowing print jobs to use larger cached writes for faster printing.

The Transport Protocol Layer. The Transport Protocol Layer is responsible for maintaining connections, providing packet transmissions and receptions between connections, and providing other transport-specific services. The term *transport-specific* refers to similar services implemented by different transport protocols. The IPXNCP.VLM module uses IPX as its transport protocol. The TRAN.VLM module shields the layers above it from the information handled by the transport protocol modules.

The NetWare DOS Requester architecture allows VLMs of a given layer to coexist and to perform their respective functions without conflict. For example, the TRAN.VLM, IPXNCP.VLM, and SECURITY.VLM modules operate at the Transport Protocol Layer. Similarly, service-protocol VLMs like FIO.VLM and PRINT.VLM operate at the Service Protocol Layer.

The *Connection Table Manager* (CONN.VLM) spans all three Requester functional layers and provides the connection services required by all levels of the Requester model.

How the Shell and the DOS Requester Work with DOS

The old NetWare Shells, NETX.COM and NETX.EXE, XMSNETX.EXE, and EMSNETX.EXE, all run "in front" of DOS. That is, the Shell intercepts interrupts (requests for service) that are directed to DOS and acts on those requests associated with a NetWare resource. If the request is not for the network, the Shell passes the interrupt (request) to DOS.

The NetWare DOS Requester operates differently than the NetWare Shell because the Requester runs "behind" DOS. DOS gives network requests to NetWare through the DOS Redirector Interface (Int 2Fh function 11). This interface is the mechanism (by design) which allows DOS to recognize foreign file systems, such as CD-ROMs and networks, and can in theory be used by any number of redirectors. The NetWare DOS Requester interacts with this interface through the REDIR.VLM module.

DOS calls redirectors when DOS itself cannot service a request. For example, when an application makes a request to DOS, DOS first attempts to qualify the request in order to determine ownership of the requested resource, such as a drive letter, a file handle, or a print device. If DOS determines that it does not own the resource requested, DOS polls the redirectors to allow them to determine ownership among themselves. If multiple redirectors are loaded, each redirector in turn attempts to qualify the request. If a redirector claims ownership to the requested resource, DOS then makes the appropriate calls to that redirector so it can complete the request; otherwise, the application receives an error for the request.

For example, if the workstation is running the DOS Requester and an application makes a DOS Int 21-3Fh "Open File" request, DOS attempts to qualify the request. If DOS can't, it issues an Int 2F-1123h call to the REDIR.VLM module in order to qualify the filename. When DOS receives a successful return code from REDIR, DOS issues an Int 2F-1116h call to open the file. The REDIR.VLM then requests other services within the DOS Requester that forward the call to the file server. The file server returns the resulting information and status codes to the DOS Requester, which then returns the information to the REDIR.VLM module, then to DOS, and then to the application.

If the workstation also loads the NetWare Shell emulator, NETX.VLM, the emulator acts like the NetWare Shell and intercepts Int 21h requests. Requests that are within the range of the old NetWare APIs (B0h through E3h) are routed immediately to the network and bypass DOS entirely. Otherwise, the request goes to DOS for servicing.

NetWare Shell emulation makes it possible for the DOS Requester to run applications that use the old Shell calls. However, the functionality of the NETX.VLM module differs from the NetWare Shell because the Shell intercepts *all* Int 21h calls and if the call involves a NetWare resource, the Shell processes them without any DOS intervention.

Note: Because of the different methods they use to manage resources, the NetWare Shell (NETX.EXE) and the DOS Requester *cannot* coexist and cannot be loaded together.

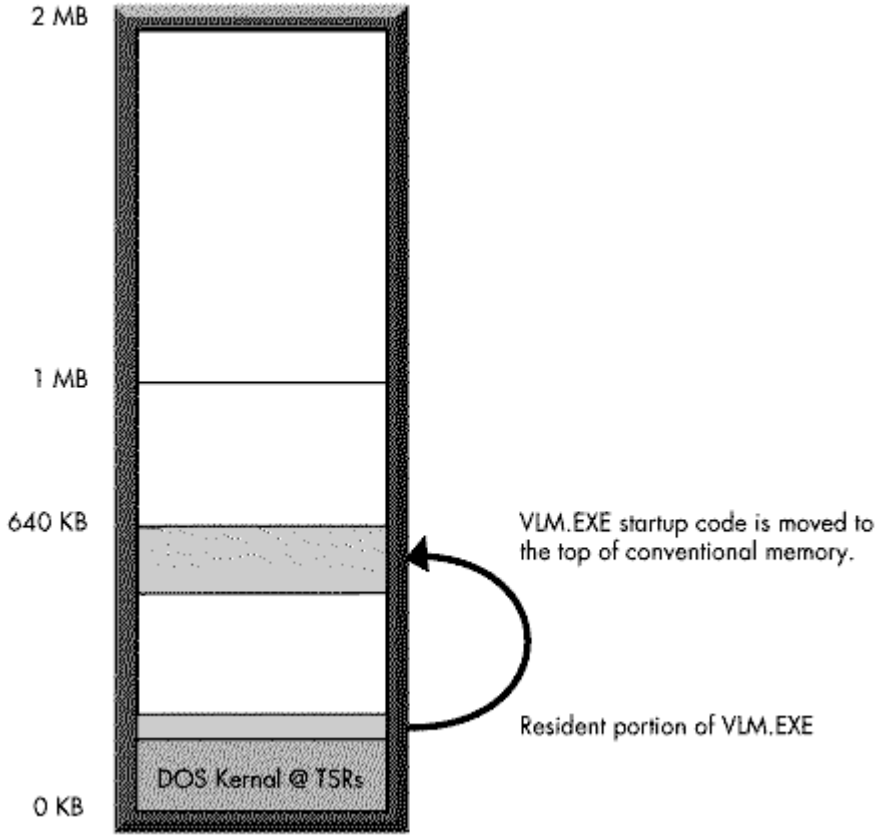
In order for the NetWare Shell to completely circumvent DOS, it uses its own set of internal resource tables for network file and print services. (For an explanation of how the NetWare Shell and the DOS Requester handle drive mappings, see the "Search Drive Mappings and the PATH Variable" and the "Still Searching . . . " NetNotes in the October 1993 and December 1993 issues of *NetWare Application Notes,* respectively.)

In contrast, the DOS Requester is tightly integrated with DOS and, as a redirector, shares resource tables with DOS. This eliminates the need to maintain redundant tables, decreasing memory requirements for the DOS Requester. However, if you load the NETX.VLM module, some of the NetWare Shell's internal tables are also maintained for backward compatibility.

How the VLMs Initialize

Because the DOS Requester is designed in a modular fashion, it can be set up in a variety of memory configurations. The VLMs can be loaded into extended memory, expanded memory, or conventional memory, and they will use upper memory blocks (rather than conventional memory) when such memory is available. The following is a description of how the DOS Requester accomplishes this.

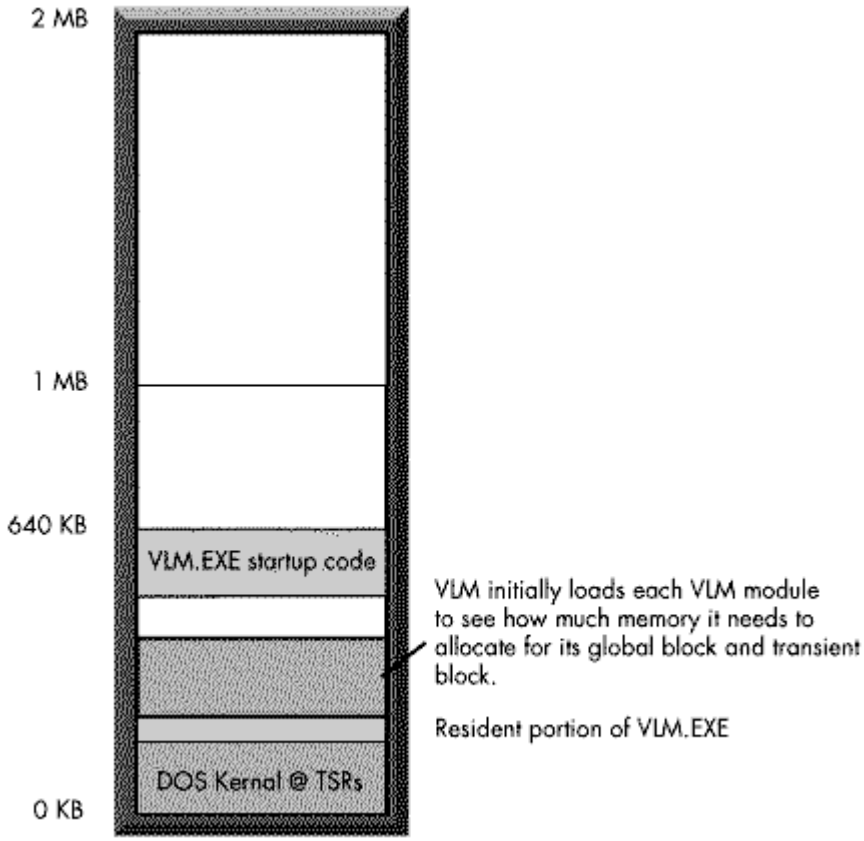
Figure 6: VLM.EXE relocates its startup code.



As shown in Figure 6, during the pre-initialization process, VLM.EXE relocates its startup code at the top of conventional memory so it can load the VLM modules and then remove itself without occupying memory blocks unnecessarily.

At this point, VLM.EXE performs several checks and initializations before loading the startup information. These checks include detecting whether the DOS Requester has already been loaded, whether it has been loaded under a task switcher, and whether the command line or the NET.CFG file contains specific parameter settings.

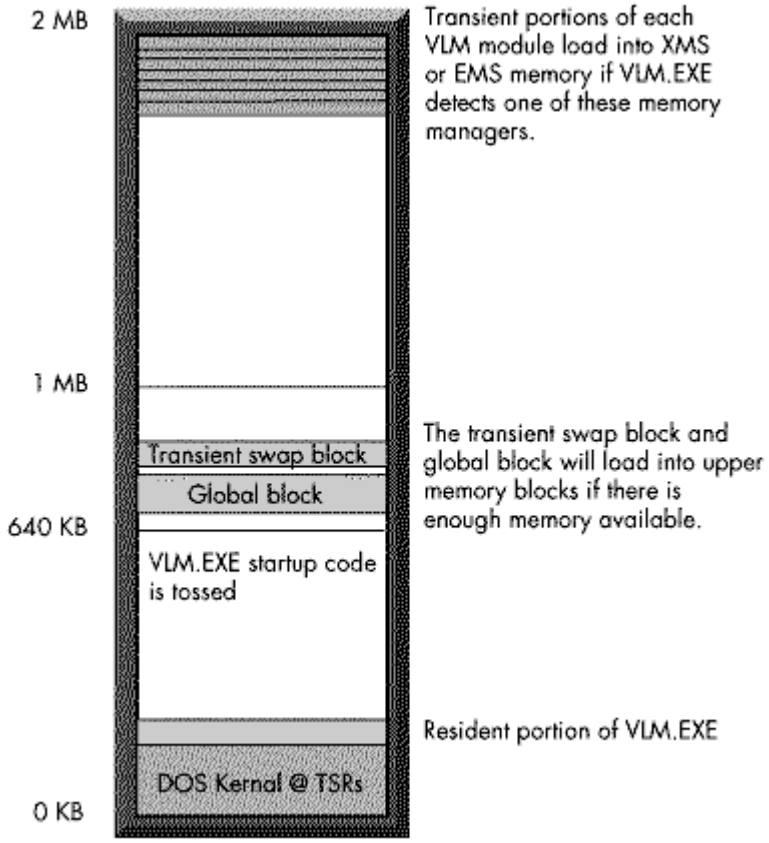
Figure 7: The VLM Manager checks memory requirements for each module.



The VLM manager is particularly interested in which VLMs are loading. VLMs are load-order dependent and follow the hard-coded defaults built into VLM.EXE unless you specify otherwise in the NET.CFG.

VLM.EXE performs a pre-initialization process by temporarily loading each VLM module and passing control to the module's initialization code. The module reports its ID and global, transient, and startup memory segment sizes. As this step happens, you see a series of dots displayed on the screen.

Figure 8: The VLM Manager takes advantage of XMS or EMS if either is present.



After all the VLMs pre-initialize, VLM.EXE begins the "real" initialization phase by calculating the total memory requirements for the *global blocks* and transient *swap block.* The global blocks, which are comprised of the global segments of each of the VLM modules you are loading, are roughly the size of the sum total of global segments. The transient swap block, during initialization, is equivalent in size to the largest VLM module being loaded. After initialization, the transient swap block is reduced in size to match that of the largest individual VLM transient segment.

VLM.EXE also determines which type of memory (XMS, EMS, or conventional memory) to allocate for storing the transient portion of the VLM modules. For example, once the DOS Requester pre-initializes, it knows how much memory to request from the XMS or EMS memory manager in order to store the transient portion of all the modules. If you don't have an XMS or EMS memory manager loaded, the transient portions of the modules are loaded into conventional memory, which can severely limit the size of the applications you can load.

The command line parameters /mx, /me, or /mc (for extended, expanded, or conventional memory) determine where the transient portions of the VLM modules are located. VLM.EXE uses this priority pattern for allocating memory when you don't specify a memory type or if the memory type you have specified is not available.

Each VLM module temporarily loads into the transient swap block and executes its initialization code, which includes initializing internal variables, hooking interrupts, notifying other VLMs of its presence, and detecting the presence of other dependent VLMs.

If the initialization of that module is successful, VLM.EXE splits the VLM module into its transient and global components, based on the memory requirements initially supplied by the VLM during the pre-initialization phase. The VLM's global and transient blocks are at this point relocated to their respective memory

locations, with the global block being relocated to conventional memory or UMBs and the transient block being relocated to XMS, EMS, UMBs, or conventional memory, depending on the memory configuration. If all this is successful, the initialization code is jettisoned.

How the DOS Requester Performs a Read Request

Assuming that an application has successfully opened a file on the server and that the pointer within the file is positioned to where a read begins, let's examine how the NetWare DOS Requester performs a read request.

First, the application makes a call to DOS through its Int 21 service using the 3Fh "Read from file" function. Upon entry, the registers contain the function number (3Fh), the file handle that is passed back from the "Open File" request, the number of bytes to read, and an address to a buffer in memory to receive the data read from the file.

The Int 21 call is first intercepted by the NETX.VLM module (if loaded) to see if the call is within its range of serviced functions (B0h through E3h). Since Int 3Fh is outside this range, the NETX.VLM passes control to DOS.

DOS ascertains that the handle came from a remote device, which can be the network or a CD-ROM (DOS doesn't know which nor does it care). DOS generates a call using the Network Redirector Service Interrupt 2F to read from a remote device. DOS passes control to the REDIR.VLM module, which determines whether the read request is for the network. To do this, the REDIR.VLM module looks at the information stored in DOS's system file table. After determining that the request is for the network, REDIR sets itself up to handle possible record locking scenarios and passes control to the FIO.VLM module.

FIO.VLM receives the read call from REDIR.VLM via the VLM Manager. The FIO module gathers specific file information based on the settings used to open the file (is it read-only, is it sharable, etc.). At this time the FIO module determines whether the DOS Requester can cache reads, whether Packet Burst is enabled, and whether the call can effectively use Packet Burst. Packet Burst information is based on the packet frame size, the amount of information that FIO can pull from workstation cache (if a cache buffer is free to receive the read), and the size of the read request itself.

If the request can be completely fulfilled from data already found in the workstation's cache, the FIO.VLM module retrieves the data from the cache buffer and sends the data to the application without ever sending a packet across the wire. If the request can be partially fulfilled from data found in the workstation's cache, the FIO module retrieves that data and sends a request for the balance to the server through the IPXNCP module and the ODI stack.

If the workstation has caching enabled and there is an available cache buffer for the file (based on the NET.CFG settings and the type of file access being performed), the FIO module implements a read-ahead algorithm. This algorithm maximizes the efficiency of network read requests and cache, especially if the requested data is in relatively small increments.

Let's say an application has a request for a 2000-byte read and the cache buffer size is roughly 1450 bytes, which matches (roughly) Ethernet's 1514 frame size (minus 64 bytes of overhead). The first 1500 bytes of incoming data is handed to the application and is not cached. However, the second 1500-byte packet is cached, and the first 500 bytes are sent to the application. But the last 1000 bytes (approximately) of the second packet are the read-ahead for the next read request. If the next application request is within the 1000 bytes, it is fulfilled from the data stored in the cache buffer. If the request cannot be fulfilled from cache, the FIO module will clear the cache and perform the next read following the procedure explained above, which starts the process over again.

If the amount of data requested warrants using Packet Burst, the request is passed into the packet burst procedure of the FIO.VLM module and the request is bursted across the network. Packet Burst can greatly enhance the efficiency of read or write requests by using a one-request/multiple-reply method for data retrieval as opposed to the one-request/one-reply method that the NCP Read normally implements. If the

workstation is not using Packet Burst, the workstation passes many more packets between itself and the server to fulfill the request. (For more information on Packet Burst technology, see the "Packet Burst Update: BNETX vs. VLM Implementation" AppNote in the November 1993 issue of *NetWare Application Notes*.)

After the FIO.VLM module completes the read, REDIR.VLM updates pertinent information in DOS's system file table, and the data and return codes are passed back through DOS to the application. The data requested is now in the application-supplied area of memory.

DOS Requester Performance Considerations

The DOS Requester and the NetWare Shell exhibit several different characteristics during the initial attaching and logging in process that tend to create a perceived speed gap between the two. Both the NetWare Shell and the DOS Requester query the network to find the nearest server for an initial attachment. They both obtain information from the initially attached server, and, if you specify a preferred server, they both obtain information about the preferred server and attach to it.

In addition, the DOS Requester executes security validations and performance enhancements that require this information to be sent over the network. These initial validations slow down the login process, but the performance enhancements greatly increase performance once you are logged in.

For example, the DOS Requester must perform NDS-specific functions during login that the NetWare Shell cannot, such as background authentication to multiple servers within the Directory tree, resolving context-specific information for NDS-style drive mappings, and so forth. NETX.EXE can only utilize bindery-based services on NetWare 4 servers.

On the performance aspects, the DOS Requester adjusts for Large Internet Packet support if you have LIP support enabled. You enable LIP support through the Large Internet Packets = ON (default) setting in the NET.CFG and if you have LIP support at the routing server. To negotiate LIP support, the DOS Requester performs an "LIP Echo" sequence to the destination server in order to establish the largest packet size that can pass through the router. This allows the DOS Requester to utilize a larger packet size for transferring data and thereby improve performance. If the NetWare Shell detects that the preferred server is more than one hop away (going through one or more routers), the packet size is forced to 512 bytes to compensate for the possibility of smaller frame sizes along the route.

The DOS Requester also negotiates a Packet Burst connection with the server if you have packet burst enabled through the PB Buffers = 1 or more (anything other than 0) in the NET.CFG and if you have Packet Burst enabled at the preferred server. Packet burst processes large read requests much more efficiently and with much greater overall throughput, especially with WAN links. The NetWare Shell with Packet Burst enabled (BNETX.EXE) supports packet burst but in a limited fashion (and uses more memory than the regular NETX.EXE shell).

The DOS Requester manages both DOS's internal tables and the NETX.VLM module's internal tables in its effort to maintain full compatibility with the NetWare Shell. In some instances, this setup requires the DOS Requester to issue multiple requests in order to update the tables it uses, while the NetWare Shell issues just one request to maintain its internal tables.

You only need to load the NETX.VLM module with the DOS Requester if you want to supply backward compatibility with NetWare utilities that were written before NetWare 4 and with applications that are written to the pre-NetWare 4 specification. But in most cases, the NETX.VLM module needs to be loaded in order to handle the bulk of the NetWare-aware applications and utilities currently available on the market. If you are exclusively logging in to NetWare 4 servers through NDS LOGIN.EXE and are only using NDS utilities, you do not need to load the NETX.VLM module.

DOS Requester Performance Examples

Even given the above ramifications, the DOS Requester has had a checkered past to its overall speed and performance. With that in mind, here are three examples showing how to set the NET.CFG with different considerations.

Example 1 shows how to optimize the DOS Requester for the most speed but without memory considerations. Example 2 shows how to optimize a workstation's memory when running the DOS Requester. Example 3 tries to find a happy medium between Example 1 and Example 2.

Example 1: Optimization Considerations. The first NET.CFG configuration gives you the largest memory hit but offers the best performance. For the absolute best performance, use the /MC parameter when loading VLM.EXE, such as VLM /MC <Enter>. The /MC parameter loads all the VLM modules into conventional memory, which amounts to about 100 KB. (If you have to do this, load everything else that you can into upper memory so the memory hit isn't so bad.)

While all performance parameters are shown here, those parameters marked with an asterisk (*) do not need to be set because their default settings are already set for maximum performance. However, they are shown here for the sake of clarity. Parameters not listed here do not adversely affect performance.

In the following listing, as in examples 2 and 3, all lines would be indented under "NetWare DOS Requester."

```
 CACHE BUFFERS                = <FILES= in CONFIG.SYS - 5, or 64>
*CACHE BUFFER SIZE            = <max media size of NIC - 64>
*CACHE WRITES                 = ON
 CHECKSUM                     = 0 (only for use with 802.2 clients)
*LARGE INTERNET PACKETS       = ON
*LOAD LOW CONN                = ON
*LOAD LOW IPXNCP              = ON
*MINIMUM TIME TO NET          = 0
 NETWARE PROTOCOL             = NDS,BIND,PNW
*PB BUFFERS                   = 3 (any amount from 1 to 9)
 PRINT BUFFER SIZE            = 256
 SIGNATURE LEVEL              = 0
*TRUE COMMIT                  = OFF
```

Setting Cache Buffers to match the Files= entry in the CONFIG.SYS (minus 5) gives almost a cache buffer for every file that you may have open (the maximum setting is 64). But what you set this to depends on if you are running Windows off the network or if you have a lot of network applications and files open concurrently.

It also depends on how much memory you have in conventional memory to give to cache buffers, because cache buffers are based on the maximum media size of the packets the workstation is sending. So if you're using Ethernet packets set to 1514 bytes, then each cache buffer will be about 1450 bytes (minus 64 bytes). If you set Cache Buffers to 64, you stand to tie up about 90 KB of conventional memory for cache buffers, control blocks, and other miscellaneous DOS Requester overhead.

The maximum cache available is 64 KB and is calculated by multiplying Cache Buffer Size by Cache Buffers. The FIO.VLM adjusts your cache buffers accordingly. If you set the number of Cache Buffers too high, you receive a warning (if your Message Level parameter in the NET.CFG is set to 3 or higher) telling you that the FIO.VLM is reducing your cache blocks by an amount. This is necessary for the FIO to load.

Note: Each VLM can be a maximum of 64 KB for the combination of the transient and global areas. Cache buffers are allocated after the other FIO.VLM requirements are met. Therefore, maximum cache buffer memory available equals 64 KB minus the other FIO.VLM allocations.

The Cache Writes = ON parameter fills the local cache buffers before writing data to the network, so using

this default value is a good idea. If you don't need the added protocol checking, set the Checksum setting to 0 (it defaults to 1). For greatest performance, the next four settings, Large Internet Packets through Minimum Time To Net, should all be left to their default settings.

For the NetWare Protocol entry, put the server type that you use the most as first in the list. For example, set Netware Protocol = BIND,NDS if you primarily use 3.x servers, or NDS,BIND,PNW if you wish to log in under Directory Services first. To be set to ON, the PB Buffers= entry should be set to a number from 1 to 9 (OFF = 0). For best performance, be sure to have this turned ON.

Setting the Print Buffers Size to its maximum of 256 bytes gives you the best performance for printing. The Signature Level parameter offers NCP packet signatures, which can affect the overall performance of every packet a workstation receives and sends. For maximum performance, set Signature Level = 0 in the NET.CFG and at the server, type SET NCP Packet Signature Option = 0 to disable server packet signatures.

The True Commit entry increases data integrity but decreases performance because you have to wait for data to be written to the server's disk, not simply to the server's cache. For best performance, use the default of OFF.

Example 2: Memory Considerations. The memory configuration example uses the absolute least amount of memory for running the VLMs. However, please note that by doing this, you will see performance degradation.

For the best use of memory, have XMS memory loaded (for MS-DOS, load HIMEM.SYS in the CONFIG.SYS) and have a large contiguous block of UMBs (upper memory blocks) available (load EMM386.EXE in the CONFIG.SYS).

One way to allocate more UMBs is to disable expanded memory, which eliminates the allocation of an EMS page frame, saving you 64 KB of memory in the UMB area. (To do this, use the /NOEMS switch with MS-DOS's EMM386.EXE.)

Also be sure to have the lines DOS=HIGH and DOS=UMB in the CONFIG.SYS so MS-DOS will load into HMA (High Memory Area), thereby freeing up more conventional memory. The DOS=UMB parameter enables the UMBs for loading TSRs and other things, thereby freeing up more conventional memory as well.

Again, those parameters preceded with an asterisk (*) don't need to be set because their default settings are already set for maximum performance.

```
*AUTO LARGE TABLE          = OFF
 AUTO RECONNECT            = OFF
 CACHE BUFFERS             = 0 (performance hit)
 AVERAGE NAME LENGTH       = <calculated value, 48--default>
 CONNECTIONS               = <calculated value, 8--default>
 LOAD LOW CONN             = OFF (performance hit)
 LOAD LOW IPXNCP           = OFF (performance hit)
 EXCLUDE VLM               = <vlm>
 NETWORK PRINTERS          = 0
 PB BUFFERS                = 0 (performance hit)
 PRINT HEADER              = 0
 PRINT TAIL                = 0
 SIGNATURE LEVEL           = 0
```

As you can see, while many of these settings minimize the DOS Requester's memory requirements, they come with substantial performance hits.

The Average Name Length can be trimmed down by taking the sum of the lengths of all the server names

you attach to, dividing by the number of servers you attach to, and then adding one. This entry works in conjunction with the Connections= parameter.

The Connections parameter shows the number of servers you attach to. The default is eight, but if you're using NDS, you may need to increase this number in order to handle "tree walking" through large corporate trees.

Use the Exclude VLM = <vlm> to eliminate optional VLMs that you don't need, which may include AUTO.VLM, NDS.VLM, BIND.VLM, or PNW.VLM. Some VLM modules may be optional to your configuration and do not need to be loaded. There are several ways to *not* load an optional module, including the following:

- Include the line EXCLUDE VLM = <vlm> under the "NetWare DOS Requester" heading in the NET.CFG file. This is the recommended way to disable a module.
- Rename the module with a different extension (i.e., .SAV).
- Delete the module (not recommended).

Optional VLMs include the following:

AUTO.VLM
PRINT.VLM (unless you want to print)
SECURITY.VLM
NDS.VLM (unless you are running Directory Services)
BIND.VLM (unless you connect to 3.1x servers and below)
PNW.VLM (unless you connect to Personal NetWare)
NETX.VLM
RSA.VLM
WSSNMP.VLM*
WSREG.VLM*
WSASN1.VLM*
WSTRAP.VLM*
MIB2IF.VLM*
MIB2PROT.VLM*
NMR.VLM*

(Modules marked with an asterisk are for SNMP services and will be covered in a future AppNote.)

Set the Network Printer to 0 only if you want to disable printing. The PRINT.VLM module will not load in this case. For the Print Header and Print Tail entries, you can set them to 0 if you are using network printing but you don't use any print job definitions set up through the PRINTCON utility.

Example 3: The Best Compromise. The best compromise of memory and performance is to use the default settings for the NET.CFG parameters, except when circumstances warrant modification. When this happens, pull from Examples 1 and 2. Example 3 is a list of the default parameters that affect both memory and performance. Since these are the default settings, all settings are marked with an asterisk.

```
*AUTO LARGE TABLE          = OFF
*AUTO RECONNECT            = ON
*AVERAGE NAME LENGTH       = 48
*CACHE BUFFERS             = 5
*CACHE BUFFER SIZE          = <max media size of NIC - 64>
 *CACHE WRITES             = ON
*CHECKSUM                  = 1
*CONNECTIONS               = 8
*LARGE INTERNET PACKETS    = ON
```

```
*LOAD LOW CONN              = ON
*LOAD LOW IPXNCP            = ON
*MINIMUM TIME TO NET        = 0
*NETWARE PROTOCOL           = NDS BIND PNW
*NETWORK PRINTERS           = 3
*PB BUFFERS                 = 3
*PRINT BUFFER SIZE          = 64
*PRINT HEADER               = 64
*PRINT TAIL                 = 16
*SIGNATURE LEVEL            = 1
*TRUE COMMIT                = OFF
```

Conclusion

This Application Note details much of the pertinent theory on how the DOS Requester 1.1 works in a client environment. In particular, we covered the DOS Requester's theory of operations and how the NetWare Shell and the DOS Requester work with DOS. We also covered how the DOS Requester initializes, then how the DOS Requester performs a read request.

Finally, we looked at three NET.CFG examples that show some different ways to configure the DOS Requester. The examples looked at performance optimization, memory optimization, and the default settings.

This AppNote will be followed by another AppNote that explains the new NET.CFG settings. We'll also look at the new NET.CFG settings that come with Personal NetWare. Finally, we'll look at some tips, tricks, and traps on running the DOS Requester that you should know about.