Network PC System Design Guidelines

A Reference for Designing Net PC Systems for Use with the Microsoft® Windows® and Windows NT® Operating Systems

Version 1.0b — August 5, 1997

Co-authored by Compaq Computer Corporation, Dell Computer Corporation, Hewlett Packard Company, Intel Corporation, and Microsoft Corporation

Revision History

Date	Description
March 10, 1997	V. 0.9. First public review
April 21, 1997	V. 1.0
May 23, 1997	V. 1.0a. Inserted port number for DHCP Request and ACK reply
August 5, 1997	Version 1.0b. Updated references; added missing values for DHCP Options in Appendix A; revised code samples in Appendixes C, D, E, F, and G.

Contents

Introduction to Network PC

This guide presents information for engineers who build or plan to build personal computers and expansion cards under the Network PC Design Initiative.

The Network PC is a new addition to the PC family—not a replacement—using Intel architecture and other microprocessor architectures that run the Microsoft® Windows® or Windows NT® Workstation operating systems. The Net PC will reduce the cost of business computing by optimizing the design for users who do not require the flexibility and expandability of the traditional PC, and by allowing organizations to centrally manage their information technology. Although the types of business users will vary, the Net PC will be ideally suited for those involved in activities such as data entry, transaction processing, and intranet and Internet access.

Following these design guidelines for the Net PC will allow PC manufacturers to deliver products with a baseline level of manageability and interoperability, and will offer greater certainty to information technology (IT) managers that specific steps have been taken to reduce total cost of ownership (TCO).

The Net PC supports either Windows 95 or Windows NT Workstation; however, more significant TCO reductions will be realized with Windows NT Workstation 4.0 and later versions. An important benefit of the Net PC design is its assurance of a seamless migration path to the rich system-manageability capabilities that are part of Windows NT Workstation 5.0.

The Net PC is designed to be a highly manageable platform, with instrumentation, network boot capabilities, controlled and managed upgrade capabilities, and a “sealed case” that prevents end-user access for changing the system hardware or software configuration. However, the Net PC preserves the corporate investment in existing Windows-based and Windows-compatible in-house application software while extending the computing platform to support Internet and intranet software based on Java and Microsoft ActiveX™ solutions.

The Net PC defined in these guidelines provides a complete hardware, software, and operating system solution to address the PC manageability issues in corporate environments, where the benefits of PC-based computing can be preserved and enhanced through greater centralized control.

Required vs. Recommended Features

In this guide, hardware features are described as follows:

- **Required:** These are the basic hardware features that must be implemented.
- **Recommended:** These features support or improve manageability or add functionality supported by the operating systems or software layers below the

operating system, such as the BIOS. Some recommendations apply to the general system, such as recommendations for improved industrial design.

- **Optional:** These features are neither required nor recommended.

In this guide, these terms have the following meanings:

- **Must** = Required
- **Should** = Recommended

Note: The requirements defined in this guide provide guidelines for designing Net PC systems. These design guidelines are not the basic system requirements for running the Microsoft Windows or Windows NT operating system.

Conventions and Terms Used in This Guide

The following conventional terms, abbreviations, and acronyms are used throughout this guide.

Add-on devices
Devices that are traditionally added to the base PC system to increase functionality, such as audio, networking, graphics, SCSI controller, and so on. Add-on devices fall into two categories: devices built onto the system board and devices on expansion cards added to the system through a system board connector such as ISA or PCI.

Desktop Management Interface (DMI)
A framework created by the Desktop Management Task Force (DMTF). DMTF specifications define industry-standard interfaces for instrumentation providers and management applications.

End user
The person who is using the Net PC to perform his or her job function, such as inputting data or running applications.

Instrumentation
A mechanism for reporting information about the state of PC hardware and software to enable management applications to ascertain and change the state of a PC and to be notified of state changes.

Intel architecture
Refers to computers based on 32-bit microprocessors that use the Pentium instruction set, such as Intel® Pentium®, Intel Pentium with MMX™ technology, Pentium Pro microprocessors, or similar processors.

Limited end-user access
In this document, several features are defined as “not accessible to end users,” meaning that the person who is using the system to input data or run other applications does not have the ability to change the configuration, purposefully or

inadvertently. Specific design guidelines might be provided in some cases, but in general, this term means that the configuration can only be changed, for example, by an administrator or service technician who has special network logon privileges, special software, or special tools. This term is equivalent to "no user-serviceable parts" in consumer electronics.

PC 97
Refers to the set of design requirements defined for the "Designed for Microsoft Windows" logo program, as specified in *PC 97 Hardware Design Guide* (Microsoft Press, 1996). References to PC 97 requirements in this guide include all changes, clarifications, and timelines for implementation of PC 97 requirements as published on http://www.microsoft.com/hwdev/desguid/pc97faq.htm.

RISC–based
Refers to computers based on Windows NT–compatible implementations of RISC processors, including computers with Digital Alpha 21064 (EV4) or higher processors.

System administrator
The person who administers the corporate network, servers, and clients, including configuration and management of Net PC systems.

System devices
Also system board devices. Devices on the system board, such as interrupt controllers, keyboard controller, real-time clock, direct memory access (DMA) page registers, DMA controllers, memory controllers, floppy disk controller (FDC), IDE ports, serial and parallel ports, PCI bridges, and so on. In today's PCs, these devices are typically integrated in the supporting chip set.

Web-Based Enterprise Management (WBEM)
Technology under development by BMC Software, Inc., Cisco Systems, Inc., Compaq Computer Corporation, Intel Corporation, and Microsoft Corporation, based on standards being developed by the DMTF and the Internet Engineering Task Force (IETF), to provide a mechanism for managed components to specify the information that they can provide to management applications and to provide a mechanism that management applications can use to access the information.

Win32 Driver Model (WDM)
A driver model based on the Windows NT driver model that is designed to provide a common set of I/O services and binary-compatible device drivers for both Windows NT and future Windows operating systems for specific driver classes. These driver classes include USB and IEEE 1394 buses, audio, still-image capture, video capture, and HID-compliant devices such as USB mice, keyboards, and joysticks.

Windows
Refers to the Microsoft Windows 95 operating system, including any add-on capabilities and any later versions of the operating system.

Windows Management Instrumentation (WMI)
Extensions to WDM being developed for Windows NT 5.0 and Windows 98 to provide an operating system interface through which instrumented components can provide information and notifications.

Windows NT
Refers to the Microsoft Windows NT 4.0 operating system, including any add-on capabilities and any later versions of the operating system, unless specific design issues are defined that relate to version 5.0. In this case, the version number is specifically cited.

Wired for Management (WfM)
An initiative aimed at increasing the manageability of desktop PCs and servers and improving the management software for these systems.

Zero Administration Windows
A Microsoft initiative that focuses on improving Windows and Windows NT for maximum automation of administrative tasks with centralized control and maximum flexibility.

Network PC Hardware Requirements

This section presents the system hardware requirements for Net PC design.

General System Requirements

This section presents the basic system requirements for Net PC design.

Any requirements that are defined as specific architecture or hardware implementations are stated in this way because there are no common industry-accepted benchmark tests for system performance.

1. Minimum CPU: 133-MHz Intel Pentium processor or compatible processor with similar performance, or Windows NT–compatible RISC-based processor
Required
This minimum computational capability is required to ensure that the customer is purchasing a product optimized to run Windows-based applications.

The requirement for Windows NT–compatible RISC-based systems includes Digital Alpha 21064 (EV4) or higher processors.

2. Level 2 cache with 256K minimum, for systems with Pentium or compatible processors
Required
This minimum L2 cache is required for performance on Net PC systems that use Pentium or compatible processors. This requirement does not apply for a Net PC

system with a Pentium Pro or compatible processor with a built-in L2 cache, or processors whose architecture permits equivalent performance without an L2 cache.

3. Minimum RAM: 16 MB
Required
Recommended: 32 MB

This minimum memory is required to ensure that the customer is purchasing a product optimized to run Windows-based applications.

4. Upgrade capabilities for RAM and CPU
Optional
If the capability for memory and CPU upgrade is provided, this capability must not be end-user accessible.

BIOS and Remote New System Setup

A Net PC system must be capable of remote configuration and booting, even when the operating system is not loaded. This section defines the requirements that support this capability, which specify baseline capabilities required to support remote boot and remote diagnostics.

5. Limit user access in preboot modes
Required
For a Net PC system, the operating system provides the capability for centrally enabling or disabling capabilities on the system. To ensure TCO objectives, security to protect enable/disable capabilities for hardware components must also be provided before the operating system boots. The purpose of this feature is to prevent end users from accidentally or purposefully circumventing operating-system level security as applied by an administrator.

6. System BIOS support for boot devices, for Intel architecture
Required
For network adapters, the system BIOS must comply with the requirements defined in Sections 3 and 4 (as they apply to Plug and Play devices) of the Compaq, Phoenix, Intel BIOS Boot Specification, version 1.01 or higher, which describes the requirements for Initial Program Load (IPL) devices.

If a CD-ROM device is provided as a boot device in the system, the system must support No Emulation mode in “El Torito—Bootable CD-ROM Format Specification, Version 1.0” by Compaq, Intel, and Phoenix or an equivalent method that supports the Windows NT CD-ROM installation process. For information about requirements for remote management capabilities for CD-ROM, see the “Platform Management Information Requirements” section and related attachments later in this document.

7. Support Int 13h Extensions in system BIOS and option ROMs, for Intel architecture

Required

The Int 13h Extensions ensure correct support for high-capacity drives. Support for the fixed-disk access subset of Int 13h Extensions must be provided in the system BIOS and in any option ROMs for storage devices that include BIOS support.

8. BIOS boot support for USB keyboard, if USB is the only keyboard

Required

For any system that uses the Intel architecture and has a USB keyboard as the only keyboard in the system, the system BIOS must provide boot support for USB keyboards. The specification for this support is defined in *Universal Serial Bus PC Legacy Compatibility Specification, Version 1.0* or higher, available from http://www.teleport.com/~usb/data/usb_le9.pdf.

9. Remote new system setup and service boot supported using DHCP and TFTP as defined in Attachment A

Required

Dynamic Host Configuration Protocol (DHCP) is an open, industry standard designed to reduce complexity of TCP/IP network administration. DHCP provides methods for dynamic configuration of computers on TCP/IP networks. DHCP is specified by IETF RFCs 1533, 1534, 1541, and 1542. Trivial File Transfer Protocol (TFTP, Revision 2) to support boot image download is implemented under IETF RFC 1350.

The required implementation for remote new system setup is defined in "Attachment A: DHCP Extensions for New System Setup" in these guidelines.

10. Preboot execution environment

Required

The execution environment provided by the client for the downloaded code described in the previous requirement must conform to the description given in "Attachment B: Preboot Execution Environment" in these guidelines.

11. Remote BIOS update and revision support

Required

Recommended: Implement a mechanism to validate that the program arrived intact after download.

BIOS ROMs must be implemented to allow them to be upgraded to a new image through OEM-provided programs using either: 1) the remote new system setup mechanism that will be downloaded and executed at boot time, or 2) normal file access and execution methods when the system is fully booted into the normal operating system environment.

If they are provided, option ROMs must also be capable of being upgraded.

Power Management Requirements

Net PC systems are power-managed systems based on solutions provided under the OnNow design initiative, so that the platform enters a low-power state when not in use. This section describes the requirements that ensure the Net PC system is power managed.

12. ACPI support meets PC 97 requirements

Required

The system board must support the Advanced Configuration and Power Interface (ACPI) Specification, version 1.0 or higher. This requirement ensures that the system correctly supports the ACPI-based Plug and Play and power management functionality.

For complete information about requirements for ACPI support, see item #4 in the "Basic PC 97" chapter of *PC 97 Hardware Design Guide*. For information about clarifications for implementation of ACPI, see http://www.microsoft.com/hwdev/desguid/pc97faq.htm.

13. Hardware support for the OnNow initiative

Required

Elements of the OnNow design initiative ensure that the operating system and device drivers control the state of individual devices and the system board. For complete design information and related requirements, see item #5 in the "Basic PC 97" chapter of *PC 97 Hardware Design Guide*.

14. BIOS support for the OnNow initiative, for Intel architecture

Required

This requirement applies only to systems that use the Intel architecture. For complete design information and related requirements, see item #6 in the "Basic PC 97" chapter of *PC 97 Hardware Design Guide*.

15. Wakeup on LAN supported

Recommended until 1/1/98; Required as of 1/1/98

Until January 1, 1998, for Net PCs with Ethernet or token ring network adapters, it is recommended that the system should be capable of being awakened from a lower power state for services and management. Magic Packet capability is a possible implementation. After January 1, 1998, wakeup capabilities are required for Net PCs and must be based on matching patterns specified by the local networking software, as described in "Network Wake-up Frames" and "Network Wake-up Frame Details" in *Network Device Class Power Management Reference Specification, Version 1.0* or higher.

Pattern matching-based wakeup enables any standard Windows network TCP/IP access, such as connections to shared drives and WinSock connections, as well as focused service and management applications, to "wake up" machines from lower power states. Microsoft operating system support for wake on LAN capabilities

will be compliant with this pattern matching solution. There is no guarantee of functionality with these operating system services for wake on LAN solutions that are not based on *Network Device Class Power Management Reference Specification* pattern matching.

This requirement applies specifically to Ethernet and token ring adapters. *Network Device Class Power Management Reference Specification* does not support ATM and ISDN adapters.

For additional implementation guidelines, see items #34–35 in the "Network Communications" chapter of *PC 97 Hardware Design Guide*. Implementation details are described in "Network Wake-up Frames" and "Network Wake-up Frame Details" in *Network Device Class Power Management Reference Specification, Version 1.0* or higher.

Platform Management Information Requirements

Each Net PC system must be able to provide consistent and dependable platform information for use by any management application. The management solution must ensure that the Net PC is manageable in heterogeneous networking environments and that there is a basic set of management information that is guaranteed to be available for management applications. This section defines the requirements to support these capabilities.

Overview of Platform Management Information Technologies

This section briefly describes the platform management information technologies referenced in these guidelines and gives references to full definitions and descriptions of these technologies. To provide useful guidelines for products prior to the availability of technologies under development, these guidelines include specifications for both the currently available and the forthcoming technologies for providing platform management information.

To understand the relationships of the technologies referenced in these guidelines, note that platform management information technologies generally have three key elements:

- Component Instrumentation—an interface by which information is supplied by manageable platform components.
- Management Information Provider—an interface used by applications to access platform management information.
- Management Information Schema—the logical structure of the information handled by the component instrumentation and the management information provider.

The platform management information technologies referenced in these guidelines are the following, defined in alphabetical order:

- **Common Information Model.** CIM is the management information schema for WBEM. It is an object-oriented schema that is being defined by a subcommittee of the DMTF. CIM is designed to be extended for each operating environment in which it is used; as an example of this extensibility, Windows operating systems use the CIM and have added Win32 extensions.

  For CIM specifications, see http://www.dmtf.org/work/cim.html.

- **Desktop Management Interface.** DMI is a platform management information framework created by the DMTF. DMTF specifications define industry-standard interfaces for component instrumentation and management applications.

  For specifications on DMI Management Information Interfaces, see *Desktop Management Interface Specification, Version 2.00*. For specification of DMI Component Interfaces, see *Desktop Management Interface Specification, Version 2.00*. Compliance of Net PC platforms to the DMI specifications referenced in these guidelines is measured by *DMI Compliance Guidelines, Version 1.0*.

- **CIM Object Manager.** CIMOM is a software component that interacts with the CIM schema and its extensions and in turn serves as a access point for management data providers to acquire data from the schema. Microsoft will implement CIMOM within Windows NT 5.0 and Windows 98 as well as other Win32 platforms, and will create a portable, platform-independent reference implementation. Versions of CIMOM for other operating systems may be available from other vendors. CIMOM handles both schema interactions and requests from management data providers.

  An example of use of CIMOM is its capability to integrate and associate management data from different sources, as in the case where one would associate a particular desktop application with its required network services, which would in turn be associated with a particular network card and a particular path through the network to the application running on a remote server. For further explanation of this example, please see the conceptual demonstration on http://wbem.freerange.com.

- **Simple Network Management Protocol.** SNMP is used widely throughout the industry as the standard under which servers, routers, hubs, and other network-based devices are managed. Enterprise-level management applications have long used SNMP as their manageability protocol because of its stability, flexibility, and wide-spread adoption.

- **Web-Based Enterprise Management.** WBEM is a set of platform management information technologies originally proposed by BMC Software, Inc., Cisco Systems, Inc., Compaq Computer Corporation, Intel Corporation,

and Microsoft Corporation based on standards being developed in a number of industry bodies, including the DMTF and the IETF. WBEM is being designed to provide uniform access for management applications to management information from a variety of sources, such as DMI, SNMP, and operating system–specific component instrumentation.

For specifications on WBEM, see http://wbem.freerange.com.

- **Win32 Extensions Schema.** This term is another name for the Win32 extensions to the CIM schema for Windows operating systems. For specifications on the Win32 Extensions Schema, see http://www.microsoft.com/management/wbem/.
- **Windows Management Interface.** WMI is an extension to WDM and is a new component instrumentation approach for Microsoft operating systems. WMI drivers have their schema built into the driver image as a resource, which enables simple, dynamic "import" of specific driver schema data into the CIM schema.

  WDM and WMI will be available in Windows NT 5.0 and Windows 98. For specifications on WMI, see http://www.microsoft.com/management/wbem/.

In the following sections, platform management information requirements are divided into two categories:

- Component Instrumentation Requirements. Describes interfaces by which instrumentation is supplied by manageable platform components.
- Management Information Providers. Describes interfaces used by applications to access platform management information.

Component Instrumentation Requirements

To ensure a basic level of manageability, a baseline set of platform management information for each Net PC needs to be available to management applications. This section addresses how system components make this information available and which information must be made available.

16. Baseline platform management information capabilities
Required

The solution for Net PC component instrumentation for WMI-capable systems is use of the WMI extensions to WDM that use the CIM and Win32 extensions schema.

The solution for advancing manageability and deploying manageable platforms with operating systems that are not WMI-capable is the DMI 2.0-based instrumentation solution. Intel and Microsoft are working together to ensure that there will be no loss of functionality as Net PCs are upgraded to use WMI-enabled operating systems.

WMI Driver Instrumentation

This section summarizes the requirements related to implementing instrumentation for WMI-capable systems using the WMI extensions.

17. Support WMI/CIM and Win32 extensions schema objects and data

Required

For Net PCs running WMI-capable operating systems, the CIM and Win32 extensions schema classes and associations listed in "Attachment H: WMI/CIM and Win32 Extensions Instrumentation Details" must be supported through WMI. Subsequent revisions of *Network PC System Design Guidelines* will incorporate additions to this set of classes and associations.

18. Support WMI alert generation for required events

Optional

None of the classes defined in the required schema have required events. Therefore, support for WMI alert generation is not required at this time, but might become required in subsequent revisions of *Network PC System Design Guidelines*.

19. Compliant with WMI alert model for WMI alerts

Required

WMI alerts that fit within the size limitation of the alert buffer can be sent directly with the alert packet. Larger alerts are required to send a global unique ID (GUID) in the alert, which is then later queried to obtain the full alert data.

All WMI alerts must supply a WMI alert severity level as recognized by the WMI alert model.

20. WMI instrumentation interface meets device-specific requirements

Required

WMI-based component instrumentation must comply with the device-specific requirements that will be described in the Microsoft Windows Device Driver Kit (DDK) WMI supplement.

DMI Component Instrumentation

This section summarizes the requirements if a DMI-based instrumentation solution is implemented.

21. DMI standard groups instrumented and deployed

Required

The standard groups listed in "Attachment I: DMI Instrumentation Details" must be instrumented and deployed on DMI-instrumented Net PCs.

22. Components compliant with DMI Component Interface

Required

DMI-instrumented components on Net PCs must comply with the Component Interface (CI) as specified for DMI version 2.0. This interface includes compliance

with the Service Provider API for Components and the Component Provider API. The required DMI instrumentation must be deployed with each DMI-instrumented platform and installed with the operating system. See the DMTF compliance guidelines regarding backward compatibility for existing instrumentation implemented using the DMI version 1.x block interface.

23. DMI event generation for DMI events in required groups
Optional
Some of the required standard groups specified in "Attachment I: MI Instrumentation Details" are associated with event generation groups. Event generation for these groups is optional.

24. Compliant with DMI event model for DMI events generated
Required
If DMI-based instrumentation generates events associated with a required group, event generation must be compliant with the event model specification defined in *Desktop Management Interface Specification, Version 2.00*.

Management Information Providers

To enable management applications to access the Net PC, each Net PC needs at least one management information provider that makes management information available to management applications using common mechanisms.

25. At least one management information provider enabled
Required
When available, WBEM-based management information providers will provide uniform access to management information from a variety of sources, including platform component instrumentation. Net PCs for which WBEM-based management information providers are available will employ those providers.

Net PCs for which WBEM-based management information providers are not available will employ DMI-based management information providers.

Note that these guidelines define the minimum management information provider functionality for a Net PC. Deployment of additional management information providers on Net PCs, such as SNMP agents, might provide significant value in certain environments.

WBEM-based Management Information Provider

Having a WBEM-based management information provider allows a Net PC to be managed by applications that can access the platform through industry-standard WBEM protocols and/or interfaces.

Microsoft will implement HMMP (a protocol for which standardization is currently being proposed to the IETF) for local and remote access to CIMOM and will also implement Common Object Model (COM) for local CIMOM access in Windows

NT 5.0 and Windows 98. Highest performance local accesses are achievable using the COM interface; greatest portability is achieved using HMMP.

Furthermore, newly developed applications for managing WBEM-capable Net PC platforms should be written to access those platforms through industry-standard WBEM protocols and/or interfaces.

26. WBEM-based service provider enabled on system
Required
The CIMOM protocol and the CIM and Win32 extensions schema will be provided automatically with WMI-instrumented Windows operating systems. For Net PC systems that are pre-installed with these operating systems, these providers must be enabled when the systems are shipped.

DMI-based Management Information Provider

Having a DMI-based management information provider allows a Net PC to be managed by applications that can access the platform through the DMI Management Interface (MI).

To be guaranteed to be able to manage DMI-instrumented Net PCs compliant with these guidelines, management applications must comply with the procedural version of the MI as specified for DMI version 2.0. This includes compliance with *Service Provider API for Management Applications* and *Management Provider API*. The new procedural interface introduced with DMI version 2.0 is an interface that supports remote access using Remote Procedure Call (RPC).

27. DMI service provider present and configured in system
Required
On each DMI-instrumented Net PC, a DMI 2.0 service provider must be present and configured to run whenever the operating system is running. This supports the ability of any compliant DMI management application to access and manage a DMI-instrumented Net PC as soon as the system is up and running.

Simple Network Management Protocol

Managing a Net PC in a corporate enterprise might require that these devices integrate with current enterprise management applications. For this reason, SNMP support can be provided in addition to DMI and WBEM providers.

28. SNMP support in addition to DMI or WBEM providers
Optional
It is recommended that if SNMP support is provided, the same information available through DMI or WMI interfaces should also be available through SNMP.

Industrial Design Requirements

This section summarizes physical design requirements for Net PC systems. Actual form factors are OEM design choices.

29. Minimum set of user indicators
Required

The minimum indicator is a power light. Indicators for hard-disk activity and LAN activity are optional and can be placed for access by service personnel only, with no requirement for end users to be able to view indicators.

30. End user can easily control power through switches and software
Required

The power control buttons must be implemented to meet PC 97 requirements for hardware support for the OnNow design initiative, as defined in item #5 in the "Basic PC 97" chapter of *PC 97 Hardware Design Guide*, plus changes and clarifications published on http://www.microsoft.com/hwdev/desguid/pc97faq.htm.

31. Minimal footprint
Recommended

In general, the form factor chosen for the Net PC should be small and should accommodate the full processor range.

32. Lockable or sealed-case design with no end-user accessible internal expansion capabilities
Required

Net PC systems include no internal expansion slots that are accessible to end users. The Net PC goals do not allow end users to add devices using traditional internal expansion capabilities. The meaning of this requirement is the equivalent of "no user-serviceable parts inside" for consumer electronics or appliances. It does not preclude providing internal expansion slots to allow the OEM to provide additional devices in manufacturing or to allow qualified service personnel to install internal devices.

See also the requirements and recommendations for external expansion buses in the "System Buses" section later in this document.

33. Thermal sensor for monitoring temperature in the chassis
Optional

This solution must be implemented under the ACPI specification, version 1.0 or higher. If implemented in any Net PC system, this must provide support for automatic shutdown in overtemp conditions.

For information about requirements for remote management capabilities, see the "Platform Management Information Requirements" section earlier in this document and the related attachments.

General Device Requirements

This section summarizes requirements for the devices and peripherals provided with Net PC systems.

34. Device driver and installation meet PC 97 requirements for Windows operating systems

Required

Each device supplied with the system must meet PC 97 requirements for that device class, and its drivers must be tested by Windows Hardware Quality Labs (WHQL). The manufacturer does not need to supply a driver if a driver provided with the operating system can be used. If the manufacturer supplies drivers, the requirements for installation include:

- All devices and drivers must pass Microsoft WHQL testing.
- All configuration settings are stored in the registry. The driver must not use INI files for configuration settings.
- The correct minidriver or any other manufacturer-supplied files specified in the device's INF must be installed in the correct locations. For manufacturer-supplied files, the vendor must not be identified as Microsoft and all other copyright and version information must be correct for the manufacturer.
- Driver installation and removal uses the Windows-based methods defined in the appropriate Windows or Windows NT DDK. However, any software applications included with the device can be installed using an alternative Windows-based installation method.
- Driver files provided by the vendor must not use the same file names as used by files included in Microsoft operating systems unless specifically agreed upon by Microsoft.
- It must be possible for the device's driver support to be installed without the user being present, with required parameters supplied by way of a script or other mechanism for predefining settings.

For systems that come pre-installed with Windows NT:

- Only 32-bit protected-mode system-level components are installed. No real-mode or 16-bit protected-mode system-level components are provided.

For systems that come pre-installed with either Windows 98 or Windows NT 5.0, the following requirements apply for drivers:

- For any device for which WDM-based support is provided in the operating system, the driver supplied by the manufacturer must be a WDM minidriver.
- Every WDM driver (or minidriver) must support Plug and Play IRPs.

For systems that come pre-installed with Windows 98, the following requirements apply for drivers:

- Every VxD must support Plug and Play messages.
- Where power management support is provided in the operating system, the VxD must implement device power management as defined in the DDK information provided for Windows 98.

For complete details about installation requirements for drivers, see item #22 in the "Basic PC 97" chapter of *PC 97 Hardware Design Guide*.

35. Each device complies with current Plug and Play specifications
Required
Each device provided in a Net PC system must meet the current Plug and Play specifications related to its class, including requirements defined in the ACPI specification and clarifications published for some Plug and Play specifications. This includes requirements for automatic device configuration, resource allocation, and dynamic disable capabilities.

Standard system devices, such as interrupt controllers, timers, keyboard controllers, RTC, DMA controllers 1 and 2 and page registers, and math co-processors, are exempt from this requirement.

For complete information about this requirement, see item #14 in the "Basic PC 97" chapter of *PC 97 Hardware Design Guide.*

For information about requirements for remote management capabilities, see the "Platform Management Information Requirements" section earlier in this document and the related attachments.

36. Unique Plug and Play device ID for each system device and add-on device
Required
Each device connected to an expansion bus must be able to supply its own unique Plug and Play identifier. Each type of bus contains different information for uniquely identifying devices on expansion cards, with guidelines and exceptions defined in *PC 97 Hardware Design Guide*. For complete requirements, see item #15 in the "Basic PC 97" chapter of *PC 97 Hardware Design Guide.*

Methods for asset-number assignment and serial-number assignment for system components are OEM specific, but must be reported using compliant instrumentation technology. For more information, see the "Platform Management Information Requirements" section earlier in this document and the related attachments.

37. Option ROMs meet Plug and Play requirements, for Intel architecture
Required

This requirement applies only for any devices that might use option ROMs on systems based on Intel architecture, whether the device is present on the system board or provided through an expansion card. Related option ROM requirements are also defined for specific bus classes and specific devices, such as SCSI and graphics adapters, respectively. For complete guidelines, see item #16 in the "Basic PC 97" chapter of *PC 97 Hardware Design Guide*.

38. All devices must support correct 16-bit decoding for I/O port addresses
Required

Each device must support a unique I/O port address in the 16-bit address range. This requirement means that, at a minimum, the upper address lines A10 – A15 can be used as device enable so the device does not respond to addresses outside of the 10-bit address range. For complete guidelines, see item #18 in the "Basic PC 97" chapter of *PC 97 Hardware Design Guide*. CardBus controllers and cards, if present, must meet the requirements defined in the "PC Card" chapter of *PC 97 Hardware Design Guide*.

39. Users are protected from connecting devices incorrectly
Required

This requirement is to help ensure that the end user or service support personnel can correctly make the physical connections required for adding a device to the system. For implementation guidelines, see item #20 in the "Basic PC 97" chapter of *PC 97 Hardware Design Guide*.

40. Minimal interaction required to install and configure devices
Required

After physically installing the device, qualified service personnel must not be required to perform any action other than to provide a pointer to a source that contains drivers and other files. For implementation guidelines, see item #21 in the "Basic PC 97" chapter of *PC 97 Hardware Design Guide*.

41. Multifunction add-on devices meet general device requirements for each device
Required

This requirement is provided as a guideline for OEM-installed devices in a Net PC system. When integrated into a Net PC system, multifunction add-on devices must meet the PC 97 requirements referenced earlier in this section for automated software-only settings for device configuration, device drivers, and Windows-based installation. For implementation guidelines, see item #23 in the "Basic PC 97" chapter of *PC 97 Hardware Design Guide*.

42. Standard system board devices use ISA-compatible addresses, for Intel architecture

Required

This includes devices with I/O port addresses within the reserved range 0h through 0ffh.

System Buses

This section defines requirements for buses provided in a Net PC system.

43. Each bus complies with written specifications and PC 97 requirements

Required

Each bus used in the system must meet all the requirements for that bus as defined in Part 3 of *PC 97 Hardware Design Guide*. This includes the requirement to meet the current Plug and Play specifications related to its class, requirements defined in the ACPI specification, and clarifications published for some Plug and Play specifications. This also includes requirements for automatic device configuration, resource allocation, and dynamic disable capabilities.

44. Universal Serial Bus with one USB port, minimum

Required

The USB implementation in the system must meet the requirements defined in USB specifications, plus the additional requirements for PC 97 as defined in the "USB" chapter of *PC 97 Hardware Design Guide*.

45. PCI bus meets PCI 2.1 specifications and PC 97 requirements

Required

If PCI is used in a Net PC system, the PCI bus must meet the requirements defined in PCI version 2.1 or higher, plus the additional requirements for PC 97 as defined in the "PCI" chapter of *PC 97 Hardware Design Guide*.

Exceptions for particular devices are noted in Parts 3 and 4 of *PC 97 Hardware Design Guide*. For example, add-on PCI IDE devices must comply with PCI 2.1 requirements and also must provide Subsystem IDs and Subsystem Vendor IDs, but PCI-to-PCI bridges and core chip sets do not have to provide Subsystem IDs and Subsystem Vendor IDs.

July 1, 1997, is the compliance date for PCI motherboard devices and for PCI add-on adapters for PCI 2.1 Subsystem IDs. For more information, see the guidelines on http://www.microsoft.com/hwdev/busbios/idpnp.htm.

46. Support for high-speed expansion buses meets PC 97 and Net PC requirements, if present

Required

For Net PC systems, internal expansion capabilities that are accessible by the end user are not allowed. However, if implemented in a Net PC system, all high-speed expansion buses and expansion devices must meet the requirements as specified in

Parts 3 and 4 of *PC 97 Hardware Design Guide*. This includes requirements for IEEE 1394 and CardBus, if implemented.

The following are specifically required for Net PC systems:

- Any bus that supports hot plugging or a device designed to use such a bus must support adding or removing devices while the system is fully powered.
- Any devices provided as expansion devices must be capable of being remotely disabled, including the capability to disable drives (CD-ROM, floppy drive, and so on), ensuring that control and TCO policies can be realized. For information about requirements for remote management capabilities, see the "Platform Management Information Requirements" section earlier in this document and the related attachments.

47. Device Bay-capable bay and peripherals

Recommended

If implemented in a Net PC system, Device Bay capabilities must meet the following requirements

- A Device Bay controller, compliant with *Device Bay Interface Specification, Version 1.0* and implemented as an ACPI device object on the system board.
- One USB controller and one IEEE 1394 controller to support all Device Bay-capable bays in the system.
- One USB port and one IEEE 1394 port for each Device Bay-capable bay in the system.

Any Device Bay peripherals provided with a Net PC system must meet the following requirements.

- Peripherals compliant with Device Bay Interface Specification, Version 1.0.
- Peripherals interface with either the USB or IEEE 1394 bus, or both.
- Support relevant USB device class specifications.

The following is specifically required for Net PC systems:

- Any devices provided as expansion devices must be capable of being remotely disabled, including the capability to disable drives (CD-ROM, floppy drive, and so on), ensuring that control and TCO policies can be realized. For information about requirements for remote management capabilities, see the "Platform Management Information Requirements" section earlier in this document and the related attachments.

48. ISA bus expansion slots must not be provided
Required
No ISA bus expansion slots are permitted in a Net PC system. The benefits of an ISA-free Net PC system for administrative support personnel include easier and more stable configuration, lower support costs, and improved performance.

This requirement does not preclude the inclusion of embedded ISA devices in a system board design. However, such devices, along with all other system board devices and system boards in general, must fully comply with all Net PC requirements for ACPI, OnNow, and system board device requirements. The end result must always be that all devices on a system board are fully and completely capable of software detection, configuration, and control.

I/O Devices

This section defines the general requirements for I/O devices.

49. Keyboard connection and keyboard
Required
Recommended: USB

The external connection requirements on any PC can also be met using a PS/2-style port or wireless capabilities in the system. A mobile or all-in-one system that has a built-in keyboard must also provide the capability for an external keyboard connection, which can be implemented using a port replicator or a single PS/2-style port with special cabling for both external keyboard and mouse. For complete requirements for keyboard ports and peripherals, see the “Input Components” chapter in *PC 97 Hardware Design Guide*.

50. Pointing-device connection and pointing device
Required
Recommended: USB or other external bus

The external connection requirements on any PC can also be met using a PS/2-style port or wireless capabilities in the system. A mobile or all-in-one system that has a built-in pointing device must also provide the capability for an external mouse connection, which can be implemented using a port replicator or a single PS/2-style port with special cabling for both external keyboard and mouse. A second serial port is not permitted as the external connection for a pointing device. For complete requirements for mouse ports and peripherals, see the “Input Components” chapter of *PC 97 Hardware Design Guide*.

51. Connection for external parallel devices
Optional
If a parallel port is present, it must be implemented as an Extended Capabilities Port (ECP) mode parallel port. For information about requirements for remote

management capabilities, see the “Platform Management Information Requirements” section earlier in this document and the related attachments.

52. Connection for external RS-232C or equivalent devices
Optional
If present, the RS-232C serial connection must be implemented using a 16550A serial port or equivalent. For information about requirements for remote management capabilities, see the “Platform Management Information Requirements” section earlier in this document and the related attachments.

53. Wireless capabilities meet PC 97 requirements, if present
Required
If wireless capabilities are included in the system, PC 97 requirements must be met as defined in the “Serial, Parallel, and Wireless Support” chapter of *PC 97 Hardware Design Guide*. For information about requirements for remote management capabilities, see the “Platform Management Information Requirements” section earlier in this document and the related attachments.

54. Network connectivity meets PC 97 requirements and supports remote new system setup
Required
The Net PC system must include I/O device support and system BIOS support for boot devices to allow installation of the operating system, as described in the “BIOS and Remote New System Setup” section earlier in this document. In addition, the manufacturer must ensure the ability to upgrade the network adapter’s option ROM using software, for forward compatibility with remote new system setup capabilities.

The network connectivity device must meet related PC 97 requirements for network communications and for the bus to which it is attached as defined in the “Network Communications” chapter and Part 3 of *PC 97 Hardware Design Guide.* The network connectivity device must be capable of remote configuration and control. For information about requirements for remote management capabilities, see the “Platform Management Information Requirements” section earlier in this document and the related attachments.

The PC 97 guidelines define requirements for NDIS 4.0 support. For Net PC systems that come pre-installed with a Microsoft operating system that supports the extensions in NDIS 5.0, the system must include an NDIS 5.0 network adapter driver. A MAC or NDIS 4.0 implementation is not allowed because NDIS 5.0 support is required to take advantage of new operating system capabilities.

55. Communications device meets PC 97 requirements, if present
Required
Modems or other communications devices such as ISDN cards implemented in the Net PC system must meet PC 97 requirements as defined in the “Modems” and “Network Communications” chapters of *PC 97 Hardware Design Guide*. For

information about requirements for remote management capabilities, see the "Platform Management Information Requirements" section earlier in this document and the related attachments.

Graphics Adapter and Multimedia Requirements

This section summarizes the Net PC requirements for the graphics adapter and monitor.

56. Display adapter meets PC 97 and Net PC minimum requirements
Required

Recommended: PCI 2.1 or AGP

A Net PC system must contain a graphics adapter that permits a color depth of 16 bits per pixel (bpp), minimum. The following are the minimum requirements:

- Minimum resolution: 800x600x16 bpp for Windows desktop and 640x480x16 bpp double buffered for Microsoft DirectDraw®–based full-screen applications.

  Display RAM requirements are tied directly to the minimum graphics resolution supported by the adapter. The requirement to support double buffering implies 1.5 MB of display RAM. However, PC 97 requirements do not specify minimum display RAM support; rather, the adapter designer can implement any solutions for supporting the minimum resolution requirements.

- Graphics adapters do not use legacy bus.

  For the graphics adapter, the video bus must not use ISA. A higher-performance solution is required to optimize performance of the packed-pixel frame buffer. Possible implementations that meet this requirement can include PCI 2.1 for all system types or the Intel Accelerated Graphics Port version 1.0 (AGP) interface for systems that have Pentium Pro processors. If the graphics adapter uses the PCI bus, it must comply with PCI 2.1 and additional requirements as defined in the "PCI" and "Graphics Adapters" chapters in *PC 97 Hardware Design Guide*.

- System operates normally with default VGA-mode driver.

  The default VGA driver is required for installing the operating system. The adapter must support 4-bit planar VGA mode as described in the Windows 95 DDK.

- Multimonitor/multiple-display adapter support meets PC 97 requirements, if dual adapter capabilities are possible in the Net PC system as provided by the system manufacturer.

For complete information, see items #1–16 in the "Graphics Adapters" chapter of *PC 97 Hardware Design Guide*.

57. Support for NTSC or PAL television output meets PC 97 requirements, if present

Required

Implementing this support is optional. If implemented, this functionality should meet PC 97 guidelines as defined in item #34 in the "Graphics Adapters" chapter of *PC 97 Hardware Design Guide*.

58. Monitor supports DDC 2.0 Level B, EDID, 800x600 minimum, and PC 97 requirements

Required

A monitor designed for or included with a Net PC system must be compliant with Display Data Channel Standard version 2.0, Level B (DDC2B), which defines the communication channel between the display and host system. In addition, the monitor must transmit an Extended Display Identification Data (EDID) structure containing unique ID Manufacturer Name and ID Product Code identifiers, and all required fields as described in Section 3 of EDID Standard 1.0, revision 1.0.

For implementation guidelines, see item #28 in the "Video Components" chapter of *PC 97 Hardware Design Guide* and the changes and clarifications on http://www.microsoft.com/hwdev/desguid/pc97faq.htm.

59. System supports MPEG-1 playback requirements for PC 97 if system has CD-ROM plus multimedia audio and video capabilities

Required

For Net PC systems that include device support for multimedia, operating system support is provided through Microsoft DirectShow™ (formerly ActiveMovie™). The minimum system requirements to support MPEG-1 playback include:

- Audio and video decode performance: 30 frames per second, minimum, as defined in item #13 in the "Video Components" chapter of *PC 97 Hardware Design Guide.*
- Graphics support for color space conversion and arithmetic stretching, including hardware arithmetic stretching and YUV off-screen surfaces for color space conversion, as defined in item #14 in the "Video Components" chapter of *PC 97 Hardware Design Guide.*

60. PC 97 DVD playback requirements, if system includes DVD-Video

Required

All Net PC systems that include DVD-Video support must provide PC 97 playback support for DVD content, including:

- Video support for decoding MPEG-2 Main Profile and Main Level video streams.
- Audio support for MPEG-2 and Dolby AC-3 decoding and DVD audio mixed with other PC audio streams.

- Synchronized audio and video, meeting general broadcast industry guidelines.
- Independent audio/video streams supported by the decode subsystem.
- WDM-based implementation for MPEG-2 acceleration.

For implementation guidelines, see item #15 in the "Video Components" chapter of *PC 97 Hardware Design Guide.*

61. Audio support meets PC 97 requirements, if present
Required
Recommended: USB or host-based digital audio

If audio capabilities are implemented in a Net PC system, audio must meet PC 97 requirements, whether implemented as baseline audio (items #3–#7) or advanced audio (items #8–#11), as defined in the "Audio Components" chapter of *PC 97 Hardware Design Guide*.

Storage and Related Components

This section presents the requirements and recommendations for storage and related peripheral devices for Net PC systems. See also system BIOS requirements to support high-capacity drives in the "BIOS and Remote New System Setup" section earlier in this guide.

62. Host controller meets PC 97 requirements
Required
Minimum requirements for SCSI, ATA and ATAPI, and IEEE 1394 are defined in related chapters in Part 3 of *PC 97 Hardware Design Guide*.

63. Primary host controller and devices support bus mastering
Required
The primary host controller must support bus mastering, whether using IDE, SCSI, or IEEE 1394. Bus-mastering support must also be enabled for IDE devices, including hard disks, CD-ROM, and tape drives. Bus-master capabilities must meet the related specification for the particular controller. For example, the programming register set for PCI IDE bus master DMA is defined in Small Form Factor (SFF) 8038i.

64. Hard drive meets PC 97 requirements
Required
A hard disk drive is required for the Net PC. The hard drive must meet the PC 97 requirements for hard drives and for the bus it uses, including:

- Drive spin-up time supports OnNow capabilities, as defined in item # 23 in the "Storage and Related Peripherals" chapter of *PC 97 Hardware Design Guide*.

- If IDE is used, each IDE drive must support Master, Slave, and Cable Select settings, as defined in item #24 in the "Storage and Related Peripherals" chapter of *PC 97 Hardware Design Guide*.

65. Hard drive is SMART-compliant
Required
SMART-compliant drive uses SMART IOCTL API, as defined in item #24 in the "ATA and ATAPI" chapter of *PC 97 Hardware Design Guide*.

66. CD-ROM meets PC 97 and Net PC requirements, if present
Required
Recommended: CD-ROM not included in system

If this device is present, the host controller must meet PC 97 requirements, as defined in the related chapter in Part 3 of *PC 97 Hardware Design Guide*. The CD-ROM drive must meet the PC 97 requirements defined in items #25–#30 in the "Storage and Related Components" chapter of *PC 97 Hardware Design Guide.*

For Net PC, the device must be capable of remote lockdown and boot device selection. For information about requirements for remote management capabilities, see the "Platform Management Information Requirements" section earlier in this document and the related attachments.

67. Media status notification support for ATAPI removable media, if present
Required
IDE and ATAPI removable devices must follow the Microsoft specification named Media Status Notification, version 1.03 or higher (included in SFF 8070i).

CD-ROM and DVD-ROM manufacturers must use the Media Status Notification specification contained within the Mt. Fuji specification, which will be provided as SFF 8090 by the SFF Committee.

68. Legacy floppy-disk controller
Optional
Recommended: Floppy disk capabilities not included in system

If a legacy FDC is included on a Net PC system, the drive and controller must meet PC 97 requirements, as defined in items #17–21 in the "Storage and Related Peripherals" chapter of *PC 97 Hardware Design Guide*.

For any implementation of a floppy drive on a Net PC system, the floppy drive must be capable of being remotely disabled (as a boot selection) and must make provisions for locking. For information about requirements for remote management capabilities, see the "Platform Management Information Requirements" section earlier in this document and the related attachments. For information about locking removable media, see the "Hardware Security Features" section later in this document.

Hardware Security Features

This section summarizes the system security requirements and recommendations for Net PC systems. For information about security for limiting user access in preboot modes, see the "BIOS and Remote New System Setup" section earlier in this document.

69. Smart card support meets interoperability specifications, if present
Required

Smart card readers and cards must be compatible with *Interoperability Specification for ICCs and Personal Computer Systems*, published by CP8 Transac, Hewlett-Packard, Microsoft, Schlumberger, and Siemens Nixdorf on http://www.smartcardsys.com.

In addition, smart card readers and device drivers must be Plug and Play compliant and adhere to the Win32 smart card specifications published in the Windows NT DDK. Smart card applications and service provider DLLs must adhere to the Win32 smart card specifications as published in the Win32 SDK.

70. Physical system security
Required

The following security features are required for Net PC hardware to prevent unauthorized access to hardware:

- External drive devices have security capabilities. Each removable media device on a Net PC system must be capable of being secured to prevent unauthorized access to data. This means that the device is rendered useless, either electronically or mechanically.
- PC case and switches have locking capabilities to prevent unauthorized internal access—an OEM-specific method can be implemented, either electronically or mechanically. Usability controls such as volume, brightness, and contrast that are usually configured by the end user are exempt from this requirement.
- Remote software management is supported for physical components. For information about requirements for remote management capabilities, see the "Platform Management Information Requirements" section earlier in this document and the related attachments.

References and Resources

The following presents information resources, services, and tools available to help build Net PC hardware. This section also lists technical references for the specifications cited in these requirements.

Information Resources

Intel developer information
http://developer.intel.com

Information on Net PC, Zero Administration Windows, and hardware development on Microsoft web sites
http://www.microsoft.com/windows/
http://www.microsoft.com/hwdev/

Windows Hardware Quality Labs (WHQL)
whqlinfo@microsoft.com
http://www.microsoft.com/hwtest/

Microsoft Developer Network (MSDN) Professional membership
Phone: (800) 759-5474
Outside North America: (510) 275-0763
Fax: (510) 275-0762
http://www.microsoft.com/msdn/

WBEM information
http://wbem.freerange.com
http://www.microsoft.com/management/wbem/

WMI information
http://www.microsoft.com/management/wbem/

Information on DMTF web site

General information	http://www.dmtf.org
Common Information Model (CIM)	http://www.dmtf.org/work/cim.html
Specifications	http://www.dmtf.org/tech/specs.html

Technical References

Advanced Configuration and Power Interface Specification, Version 1.0
http://www.teleport.com/~acpi/

ATA 2 [X3T9.2 948D]
ATA Packet Interface for CD-ROM, SFF 8070I and other specifications
Small Computer Interface (SCSI-2) [X3T9.2-375R]
Small Computer Interface (SCSI-3) Parallel Interface (SPI) [X3T9.2/91-10]
Global Engineering Documents
Fax: (303) 397-2740
Telephone: (800) 854-7179; Outside North America: (303) 792-2181

CIM Specifications
http://www.dmtf.org/work/cim.html

Desktop Management Interface Specification, Version 2.00
DMI Compliance Guidelines, Version 1.0
http://www.dmtf.org/tech/specs.html

Device Bay Interface Specification, Version 1.0
http://www.device-bay.org

Device Class Power Management Specifications
http://www.microsoft.com/hwdev/onnow.htm

El Torito—Bootable CD-ROM Format Specification, version 1.0
Compaq, Intel, Phoenix BIOS Boot Specification, version 1.01
http://www.ptltd.com/techs/specs.html

IBM Personal System/2 Common Interfaces, Part No. S84F-9809
IBM Personal System/2 Mouse Technical Reference, Part No. S68X-2229
International Business Machines Corporation
IBM Customer Publications Support: (800) 879-275
Or contact an IBM sales representative

IEEE 1394 Standards
Telephone: (800) 949-4333
Fax: (410) 259-5045
Released Standards: Global Engineering Documents

Intel/Duracell Smart Battery System Specification
http://developer.intel.com/ial/powermgm/specs.htm

International Color Consortium
ICC Profile Format Specification
http://www.color.org

Interoperability Specification for ICCs and Personal Computer Systems
http://www.smartcardsys.com

NLX Motherboard Specification, version 1.0
http://www.teleport.com/~nlx/

Media Status Notification, version 1.03 or higher (included in SFF 8070i)
Mt. Fuji Specification (SFF 8090)
Global Engineering Documents

Multi-session Compact Disc Specification
Enhanced Music CD Specification, version 1.0
Philips Consumer Electronics B.V.
Coordination Office Optical−Magnetic Media Systems
Building SWA-109, PO Box 80002
5600 JB Eindhoven, The Netherlands
Fax: (31) (40) 732113

PC 97 Hardware Design Guide
http://www.microsoft.com/hwdev/pc97.htm

PCI, Version 1.0
PCI Bus Power Management Interface Specification
http://www.pcisig.com

PCMCIA Standards
Personal Computer Memory Card International Association
2635 North First Street, Suite 209
San Jose, CA 95134 USA

Plug and Play specifications
http://www.microsoft.com/hwdev/specs/pnpspecs.htm

Universal Serial Bus, Version 1.0
http://www.usb.org

Universal Serial Bus PC Legacy Compatibility Specification, Version 1.0
USB device class specifications
http://www.teleport.com/~usb/data/usb_le9.pdf

WBEM Specifications
http://wbem.freerange.com

WMI Specifications and Win32 Extensions Schema
http://www.microsoft.com/management/wbem/

Checklist for Network PC Requirements

General System Requirements

1. Minimum CPU: 133-MHz Intel Pentium processor or compatible processor with similar performance, or Windows NT–compatible RISC-based processor
Required

2. Level 2 cache with 256K minimum, for systems with Pentium or compatible processors
Required

3. Minimum RAM: 16 MB
Required

4. Upgrade capabilities for RAM and CPU
Optional

BIOS and Remote New System Setup

5. Limit user access in preboot modes
Required

6. System BIOS support for boot devices, for Intel architecture
Required

7. Support Int 13h Extensions in system BIOS and option ROMs, for Intel architecture
Required

8. BIOS boot support for USB keyboard, if USB is the only keyboard
Required

9. Remote new system setup and service boot supported using DHCP and TFTP as defined in Attachment A
Required

10. Preboot execution environment
Required

11. Remote BIOS update and revision support
Required

Power Management Requirements

12. ACPI support meets PC 97 requirements
Required

13. Hardware support for OnNow initiative
Required

14. BIOS support for OnNow initiative, for Intel architecture
Required

15. Wakeup on LAN supported
Required

Component Instrumentation Requirements

16. Baseline platform management information capabilities
Required

WMI Driver Instrumentation

17. Support WMI/CIM and Win32 extensions schema objects and data
Required

18. Support WMI alert generation for required events
Optional

19. Compliant with WMI alert model for WMI alerts
Required

20. WMI instrumentation interface meets device-specific requirements
Required

DMI Component Instrumentation

21. DMI standard groups instrumented and deployed
Required

22. Components compliant with DMI Component Interface
Required

23. DMI event generation for DMI events in required groups
Optional

24. Compliant with DMI event model for DMI events generated
Required

Management Information Providers

25. At least one management information provider enabled
Required

26. WBEM-based service provider enabled on system
Required

27. DMI service provider present and configured in system
Required

28. SNMP support in addition to DMI or WBEM providers
Optional

Industrial Design Requirements

29. Minimum set of user indicators
Required

30. End user can easily control power through switches and software
Required

31. Minimal footprint
Recommended

32. Lockable or “sealed case” design with no end-user accessible internal expansion capabilities
Required

33. Thermal sensor for monitoring temperature in the chassis
Optional

General Device Requirements

34. Device driver and installation meet PC 97 requirements for Windows operating systems
Required

35. Each device complies with current Plug and Play specifications
Required

36. Unique Plug and Play device ID for each system device and add-on device
Required

37. Option ROMs meet Plug and Play requirements, for Intel architecture
Required

38. All devices must support correct 16-bit decoding for I/O port addresses
Required

39. Users are protected from incorrectly connecting devices
Required

40. Minimal interaction required to install and configure devices
Required

41. Multifunction add-on devices meet general device requirements for each device
Required

42. Standard system board devices use ISA-compatible addresses, for Intel architecture
Required

System Buses

43. Each bus complies with written specifications and PC 97 requirements
Required

44. Universal Serial Bus with one USB port, minimum
Required

45. PCI bus meets PCI 2.1 specifications and PC 97 requirements
Required

46. Support for high-speed expansion buses meets PC 97 and Net PC requirements, if present
Required

47. Device Bay-capable bay and peripherals
Recommended

48. ISA bus expansion slots must not be provided
Required

I/O Devices

49. Keyboard connection and keyboard
Required

50. Pointing-device connection and pointing device
Required

51. Connection for external parallel devices
Optional

52. Connection for external RS-232C devices
Optional

53. Wireless capabilities meet PC 97 requirements, if present
Required

54. Network connectivity meets PC 97 requirements and supports remote new system setup
Required

55. Communications device meets PC 97 requirements, if present
Required

Graphics Adapter and Multimedia Requirements

56. Display adapter meets PC 97 and Net PC minimum requirements
Required

57. Support for NTSC or PAL television output meets PC 97 requirements, if present
Required

58. Monitor supports DDC 2.0 Level B, EDID, 800x600 minimum, and PC 97 requirements
Required

59. System supports MPEG-1 playback requirements for PC 97, if system has CD-ROM plus multimedia audio and video capabilities
Required

60. PC 97 DVD playback requirements, if system includes DVD-Video
Required

61. Audio support meets PC 97 requirements, if present
Required

Storage and Related Components

62. Host controller meets PC 97 requirements
Required

63. Primary host controller and devices support bus mastering
Required

64. Hard drive meets PC 97 requirements
Required

65. Hard drive is SMART-compliant
Required

66. CD-ROM meets PC 97 and Net PC requirements, if present
Required

67. Media status notification support for ATAPI removable media, if present
Required

68. Legacy floppy-disk controller
Optional

Hardware Security Features

69. Smart card support meets Interoperability Specifications, if present
Required

70. Physical system security
Required

Attachment A: DHCP Extensions for New System Setup

This description assumes a knowledge of the standard DHCP/BOOTP protocols.

Protocol Overview

The protocol is a combination of a straightforward extension of DHCP (through the use of several new DHCP Option tags) and the definition of simple packet transactions which use the DHCP packet format and options to pass additional information between the client and server. This added complexity is introduced by the requirement to operate without disturbing existing DHCP services.

In this protocol, DHCP options fields are used to do the following:

- Distinguish DHCP request packets sent by a client as part of this extended protocol from other DHCP request packets that the installation server may receive.
- Distinguish DHCP reply packets sent by a server as part of this extended protocol from other DHCP reply packets that the client may receive.
- Convey client network adapter type.
- Convey client System ID.
- Convey client system architecture type.

Based on the client network adapter type and system architecture type, the server returns to the client the file name (on the server) of an appropriate executable. The client downloads the specified executable into memory and executes it. How this executable accomplishes the setup of the system is not specified by these guidelines.

This section presents an informal, step-by-step description of the remote new system setup protocol. A detailed description of packet formats and client and server actions appears later in this attachment.

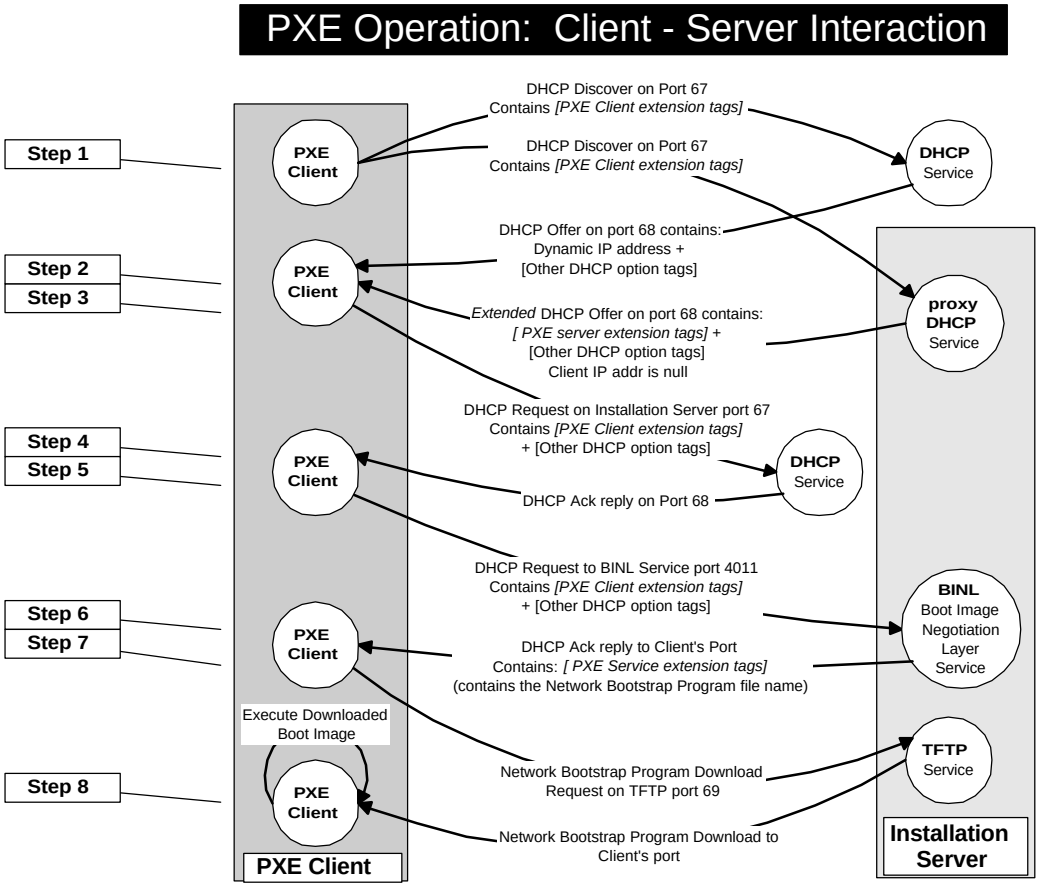


Figure 1 – PXE Operation – Client Server Interaction (updated in version 1.0b)

Step 1. The client broadcasts a DHCP discover message to the standard DHCP port (67). An option field in this packet contains the following:

- A tag for client identifier (if the client identifier is known).
- A tag for the client Network Interface Identifier.
- A tag for the client system architecture.

Step 2. The PXE PROXY DHCP Service responds by sending a PXE PROXY DHCPOFFER message to the client on the standard DHCP reply port (68). This packet contains the address of the PXE PROXY DHCP Service. The client IP address field is null.

At this point, other DHCP Services and BOOTP Services also respond with DHCP offers or BOOTP reply messages on port 68. Each message contains standard DHCP parameters: an IP address for the client and any other parameters that the

administrator might have configured on the Service. If the Installation Server is also functioning as a standard DHCP Service, then the DHCP Service reply from the Installation Server will also contain standard DHCP parameters (in particular, an IP address for the client)

The timeout for a reply from a DHCP server is standard. The timeout for re-broadcasting to receive a DHCPOFFER with PXE extensions, or a PROXY DHCPOFFER is based on the standard DHCP timeout, but is substantially shorter to allow reasonable operation of the client in standard BOOTP or DHCP environments that do not provide a OFFER with PXE extensions. The PXE timeout for rebroadcast is:

4, 8, 16 seconds, yielding three broadcasts and a timeout after 28 seconds.

The PXE timeout for rebroadcast is 4 seconds after receiving an OFFER without PXE extensions but with a valid “bootfile name” option.

Step 3. From the DHCPOFFER(s) that it receives, the client records the following:

- The Client IP address (and other parameters) offered by a standard DHCP or BOOTP Service.
- The Server IP address of the BINL (Boot Image Negotiation Layer) Service from the “siaddr” field in the PXE proxy DHCP offer.

Step 4. If the client selects an IP address offered by a DHCP Service, then it must complete the standard DHCP protocol by sending a request for the address back to the Service and then waiting for an acknowledgment from the Service. If the client selects an IP address from a BOOTP reply, it can simply go ahead and use the address.

Step 5. The client sends a DHCP Request packet to the BINL Service on port 4011. This packet is exactly the same as the initial DHCP Discover in Step 1, except that it is coded as a DHCP Request and now contains the following:

- Contains the IP address assigned to the client from a DHCP Service.
- Contains all the PXE options fields received from the selected DHCP Offer which contained the PXE options.

Step 6. The BINL Service on the Installation Server sends a DHCP Acknowledge packet back to the client, also on port 4011. This reply packet contains:

- Boot file name and location.
- The Client UUID/GUID option in the PXE proxy DHCP offer.

- MTFTP[1] configuration parameters.

Step 7. The client downloads the executable file using either standard TFTP or MTFTP. The file downloaded and the placement of the downloaded code in memory is dependent on the client's CPU architecture. (For Intel architecture Net PCs, see Attachment B.)

Step 8. Finally, the PXE Client initiates execution of the downloaded code. The way in which this is done is dependent on the client's CPU type. For Intel architecture Net PC systems, the client code executes a far call to the first location in the code.

Relationship to the Standard DHCP Protocol

The initial phase of this protocol piggybacks on a subset of the DHCP protocol messages to enable the client to discover an installation server, that is, one that delivers executables for new system setup. The client *can* use the opportunity to obtain an IP address, which is the expected behavior, but it is not required. Clients that do obtain an IP address using DHCP or BOOTP must implement the protocol as specified in RFC 1541, even though not all possible messages and states of that protocol are described or mentioned in this protocol specification. The points at which this protocol piggybacks or otherwise interacts with the standard DHCP protocol are also noted.

The second phase of this protocol takes place between the client and an installation server, and uses the DHCP message format simply as a convenient format for communication. This second phase of the protocol is otherwise unrelated to the standard DHCP services.

[1] Multicast Trivial File Transfer Protocol, as defined by this document through the use of DHCP encapsulated vendor options.

PXE DHCP Options				
Tag Name	*Tag Number*	*Length*	*Type*	*Data Field*
Client UUID/GUID	97	17	20	*per Attachment K*
Client Network Interface Identifier	94	3-9	1 = UNDI 2 = PCI 3 = PNP	Type 1 = Major ver(1), Minor Ver(1) Type 2 = Vendor ID(2), Device ID(2), Class Code(3), Rev(1) Type 3 = EISA Device ID(4), Class Code(3)
Tag Name	*Tag Number*	*Length Field*		*Data Field*
Client System Architecture	93	2		0 = Intel Architecture PC 1 = NEC/PC98 2 = etc.
Class Identifier	60	9		"PXEClient"

Encapsulated Vendor Options (DHCP Option #43)			
Tag Name	*Tag Number*	*Length Field*	*Data Field*
DHCP_VENDOR	43	varies	Encapsulated options below
(Multiple DHCP_VENDOR options can be used)			

"PXEClient" Encapsulated Options for DHCP Option #43			
Tag Name	*Tag Number*	*Length Field*	*Data Field*
PXE_PAD	0	None	None
MTFP IP Addr	1	4	a0, a1, a2, a3
MTFTP Client UDP	2	2	Port Number
MTFTP Server UDP	3	2	Port Number
MTFTP Start Delay	4	1	
MTFTP Timeout Delay	5	1	
Reserved	6-63	1	
Loader Options	64-127	1	(vendor specific)
Vendor Options	128-254	1	(vendor specific)
PXE_END	255	None	

The *Client UUID/GUID* field specifies a globally unique ID (GUID), retrieved from the client system. Client UUID/GUID must be generated per Attachment K. If the Client does not have a GUID, the DHCP service (or Installation Service) may supply one by returning the option with a valid value. The client must store this value if it can, and if so, must use it in all subsequent DHCP transactions.

The *Client Network Interface Identifier* specifies either the physical Network Interface Adapter or indicates the presence of an API (UNDI, described below) that will support a universal boot loader. The UNDI interface should be supported.

The UNDI type field must have a major version of 2 and a minor version of 0 for this version of the protocol. (Future versions may recognize more tags based on this version number.)

If neither PCI nor PNP information is available then the UNDI interface should be supported. Otherwise, the vendor must create an ad hoc PNP or PCI entry and assume the responsibility of distributing the appropriate NIC driver to PXE Servers.

The *Client System Architecture* identifier specifies the system architecture of the client.

The *Class Identifier* (Option 60) of "PXEClient" is required to assure unambiguous identification of clients meeting this specification.

Encapsulated Options (Option 43) are provided to allow configuration for MTFTP boot file transfers. MTFTP should be implemented in the Client. If provided by the DHCP, proxyDHCP, or BINL service, these options should be used.

MTFTP IP Addr is the multicast IP address the client must use to receive the image file.

MTFTP Client UDP is the port the client must listen on to receive the image file.

MTFTP Server UDP is the port the client must use to communicate with the MTFTP service. The client binds to the MTFTP UDP port and waits for the duration of the MTFTP transmission start delay to receive packets.

MTFTP Start Delay is the timeout to begin receiving image file packets before attempting to become the MTFTP acknowledging client (master client) upon initial connection to the MTFTP service.

MTFTP Timeout Delay is the delay, multiplied by the percentage of the file received, the client must wait before attempting to become the MTFTP acknowledging client (master client) upon cessation of packet transmissions during an ongoing MTFTP transfer.

Client Behavior

This section summarizes client behavior for initiation, discovery reply, installation service request, installation service reply, and executable download and execution.

Sending a PXE (Preboot eXecution Environment) Client message requires the use of DHCP Option fields. All PXE Client packets provide the same extended DHCP information in these options. This includes DHCP Request messages used to communicate with the server to which the PXE Client has been redirected. Other fields and options may be different between the packets, based on the standard DHCP protocol.

Initiation

To initiate the interchange between the client and server, the client broadcasts a DHCPDISCOVER packet to the standard DHCP server UDP port (67). The contents of this message must be as described in RFC 1541 for a

DHCPDISCOVER message, with the addition of PXE Client option fields: The format of these options is specified below:

DHCP Header		
Field (length)	**Value**	**Comment**
op (1)	1	Code for BOOTP BOOTREQUEST
htype (1)	*	
hlen (1)	*	
hops (1)	*	
xid (4)	*	
secs (2)	*	
flags (2)	*	
ciaddr (4)	0.0.0.0	PXE client always sets this value to 0.0.0.0
yiaddr (4)	blank	Client's IP address. Provided by server
siaddr (4)	*	Next bootstrap server IP address
giaddr (4)	*	
chaddr (16)	xx-xx-xx-xx-xx-xx-xx-xx	Client's MAC address
sname (64)	*	Can be overloaded if using Opt 66
bootfile (128)	*	Can be overloaded if using Opt 67
99.130.83.99		

DHCP Options				
Tag Name	*Tag Number*	*Length*	*Type*	*Data Field*
Client UUID/GUID	97	17	20	*per Attachment K*
Client Network Device Interface Type	94	3-9	1 = UNDI 2 = PCI 3 = PNP	Type 1 = Major ver(1), Minor Ver(1) Type 2 = Vendor ID(2), Device ID(2), Class Code(3), Rev(1) Type 3 = EISA Device ID(4), Class Code(3)

Tag Name	*Tag Number*	*Length Field*	*Data Field*
DHCP Message Type	53	1	1 = DHCPDISCOVER
Class Identifier	60	9	"PXEClient"
Client System Architecture	93	2	0 = Intel Architecture PC, 1 = NEC/PC98

After sending the DHCPDISCOVER message, the client must be prepared to receive replies as described in the following section.

Discovery Reply

In this state, the client is prepared to receive one or more *extended* DHCPOFFER replies from servers on the standard DHCP client UDP port (68). Sending a PXE Server message requires the use of DHCP Options. The format of these options are as follows:

DHCP Header		
Field (length)	**Value**	**Comment**
op (1)	2	Code for BOOTP REPLY
htype (1)	*	
hlen (1)	*	
hops (1)	*	
xid (4)	*	
secs (2)	*	
flags (2)	*	
ciaddr (4)	0.0.0.0	PXE client always sets this value to 0.0.0.0
yiaddr (4)	a0, a1, a2, a3	Client's IP address. Provided by server
siaddr (4)	a0, a1, a2, a3	Next bootstrap server IP address
giaddr (4)	*	
chaddr (16)	*	Client's MAC address
sname (64)	*	Can be overloaded if using Opt 66
bootfile (128)	*	Can be overloaded if using Opt 67
99.130.83.99		

DHCP Options				
Tag Name	*Tag Number*	*Length*	*Type*	*Data Field*
Client UUID/GUID	97	17	20	*per Attachment K*

Tag Name	*Tag Number*	*Length Field*	*Data Field*
DHCP Message Type	53	1	2 = DHCPOFFER
Server Identifier	54	4	a1, a2, a3, a4
Class Identifier	60	9	"PXEClient"

Encapsulated Vendor Options (DHCP Option #43)			
Tag Name	*Tag Number*	*Length Field*	*Data Field*
DHCP_VENDOR	43	varies	Encapsulated options below
(Multiple DHCP_VENDOR options can be used)			

"PXEClient" Encapsulated Options for DHCP Option #43			
Tag Name	*Tag Number*	*Length Field*	*Data Field*
PXE_PAD	0	None	None
MTFP IP Addr	1	4	a0, a1, a2, a3
MTFTP Client UDP	2	2	Port Number
MTFTP Server UDP	3	2	Port Number
MTFTP Start Delay	4	1	
MTFTP Timeout Delay	5	1	
PXE_END	255	None	

If the responding server does not have an installation capability, it will provide a valid address in *siaddr* to redirect the client to an installation server.

In this state, the client must also be prepared to receive one or more *standard* DHCPOFFER messages from servers. Each of these messages will contain configuration information as specified in RFC 1541. Each *extended* DHCPOFFER message can also contain configuration information as specified in RFC 1541. The presence of such information in an *extended* DHCPOFFER message is indicated by a nonzero value in the client IP address field. Which, if any, of these configurations is used by the client is not defined by this specification. If the client decides to accept one of the configurations offered, then it must engage in further communications with the server as specified in RFC 1541.

To move to the installation server request state, the client must have received at least one *extended* DHCPOFFER message. Beyond this, the criteria for the client exiting this state are not defined by this specification.

Installation Service Request

To enter this state, the client must have an IP address. Also, the client must have received one or more *extended* DHCPOFFER messages and therefore know the IP address of one or more installation servers. The client selects one of these installation servers and sends a DHCPREQUEST message to the server on port 4011. Otherwise the format of this message is the same as an *extended* DHCPDISCOVER. The following table lists the required values in the fields of this message; fields marked with an asterisk contain unspecified values.

DHCP Header		
Field (length)	**Value**	**Comment**
op (1)	1	Code for BOOTP BOOTREQUEST
htype (1)	*	
hlen (1)	*	
hops (1)	*	
xid (4)	*	
secs (2)	*	
flags (2)	*	
ciaddr (4)	0.0.0.0	PXE client always sets this value to 0.0.0.0
yiaddr (4)	a0, a1, a2, a3	Client's IP address. Provided by DHCP server
siaddr (4)	a0, a1, a2, a3	server IP address
giaddr (4)	0.0.0.0	
chaddr (16)	*xx-xx-xx-xx-xx-xx-xx-xx	Client's MAC address
sname (64)	*	Can be overloaded if using Opt 66
bootfile (128)	*	Can be overloaded if using Opt 67
99.130.83.99		

DHCP Options				
Tag Name	*Tag Number*	*Length*	*Type*	*Data Field*
Client UUID/GUID	97	17	20	*per Attachment K*
Client Network Device Interface Type	94	3-9	1 = UNDI 2 = PCI 3 = PNP	Type 1 = Major ver(1), Minor Ver(1) Type 2 = Vendor ID(2), Device ID(2), Class Code(3), Rev(1) Type 3 = EISA Device ID(4), Class Code(3)

Tag Name	*Tag Number*	*Length Field*	*Data Field*
DHCP Message Type	53	1	3 = DHCPREQUEST
Server Identifier	54	4	a1, a2, a3, a4
Class Identifier	60	9	"PXEClient"
Client System Architecture	93	2	0 = Intel Architecture PC 1 = NEC/PC98.

Installation Service Reply

In this state, the client must be prepared to receive an *extended* DHCPACKNOWLEDGE message from the installation server. The following table lists the required values in the fields of this message:

DHCP Header		
Field (length)	**Value**	**Comment**
op (1)	2	Code for BOOTP REPLY
htype (1)	*	
hlen (1)	*	
hops (1)	*	
xid (4)	*	
secs (2)	*	
flags (2)	*	
ciaddr (4)	0.0.0.0	PXE client always sets this value to 0.0.0.0
yiaddr (4)	a0, a1, a2, a3	Client's IP address. Provided by server
siaddr (4)	a0, a1, a2, a3	Next bootstrap server IP address
giaddr (4)	*	
chaddr (16)	*	Client's MAC address
sname (64)	*	Can be overloaded if using Opt 66
bootfile (128)	*	Can be overloaded if using Opt 67
99.130.83.99		
DHCP Options		

Tag Name	*Tag Number*	*Length*	*Type*	*Data Field*
Client UUID/GUID	97	17	20	*per Attachment K*

Tag Name	*Tag Number*	*Length Field*	*Data Field*
DHCP Message Type	53	1	4 = DHCPACKNOWLEDGE
Server Identifier	54	4	a1, a2, a3, a4

Encapsulated Vendor Options (DHCP Option #43)

Tag Name	*Tag Number*	*Length Field*	*Data Field*
DHCP_VENDOR	43	varies	Encapsulated options below
(Multiple DHCP_VENDOR options can be used)			

"PXEClient" Encapsulated Options for DHCP Option #43

Tag Name	*Tag Number*	*Length Field*	*Data Field*
PXE_PAD	0	None	None
MTFP IP Addr	1	4	a0, a1, a2, a3
MTFTP Client UDP	2	2	Port Number
MTFTP Server UDP	3	2	Port Number
MTFTP Start Delay	4	1	
MTFTP Timeout Delay	5	1	
PXE_END	255	None	

The options fields in this message must include the following:

- The Installation Server must direct the client to a TFTP server by responding with *siaddr* filled in. (Usually, the TFTP Server resides on the same machine, so *siaddr* would be set to null.)
- Client UUID/GUID if received from the client

- Server Identifier (address of the responding Installation server)
- Bootfile Name if Option 52 (Option Overload) is used.
- *System architecture* indicating architecture the *bootfile* supports. *System architecture* value must be the same as received by the client to insure proper operation of the *bootfile*.

After receiving this message, the client moves to the executable download state.

Executable Download and Execution

In this state, the client is to download all or some portion of the executable file using the standard TFTP. The portion of the file downloaded and the placement of the downloaded code in memory is dependent on the client's CPU architecture.

For Net PC systems based on Intel Architecture, the entire bootstrap image (up to 32K in size) is downloaded into the client PC starting at location 07C00h. The TFTP/MTFTP session that was used to download the bootstrap image is terminated and the logical network connection to the TFTP server is closed.

After the bootstrap image is downloaded, the TFTP connection is closed and control is passed to the bootstrap image. The way in which this is done is dependent on the client's CPU type. For Net PC systems based on Intel Architecture, the boot ROM code is to execute a far call to location 0:7C00h.

MTFTP Operation

Implementation of MTFTP in the client is strongly recommended. If the server sends MTFTP parameters, then the client should proceed as described in this section. In this case the client goes through three phases: an open, a receive and a close, with an error recovery phase that can be entered at any point.

MTFTP open

1. The network client acquires at least the following information from the BINL reply:

   - Client bootfilename
   - MTFTP Server UDP port number
   - MTFTP Client UDP port number
   - MTFTP multicast IP address
   - MTFTP transmission start delay
   - MTFTP transmission time-out delay

2. Client binds to the MTFTP UDP port and waits for the MTFTP transmission start delay to receive packets. No network traffic is generated.

3. If there is a response, MTFTP packets are collected from the network. The client keeps track of received packets in an internal list.

If no packet is received, the client initiates an MTFTP open to the server.

MTFTP receive

1. In order to find out if a client needs to acknowledge or not, the server sends a unicast TFTP packet to that client. The first packet of a MTFTP transmission is always sent both as unicast and multicast UDP/IP. This instructs the network client that it is the acnowledging client.
2. A server always transmits the complete file. Therefore, clients that start listening to a conversation part way through can wait and then get the rest on the next MTFTP transmission to make up for what was missed the first time.
3. The acknowledging client must ACK all packets even if the client has received the entire file.

MTFTP close

1. An MTFTP transmission is finished when the acknowledging client has received all packets and disconnects from the network. Clients who did not receive all packets can initiate a new transmission, if one has not already started.
2. Before a new transmission is started there is a calculated delay. The default delay is modified by an algorithm based on the number of packets received. Clients who received fewer packets will wait for a shorter time than those who received more. This algorithm ensures that:

    - Slow clients define the transmission speed.
    - Clients with a large number of received packets can disconnect from MTFTP after they received all missing packets.
    - Clients who hook into an ongoing MTFTP transmission and therefore only receive the tail of the transmission can disconnect from MTFTP after they received the missing head of the transmission.
    - Clients with a small number of received packets are more likely to become the acknowledging client.

Server Behavior

The server behavior needed for the extended protocol comprises two pieces of functionality: a redirection service, and an installation service.

- The redirection service receives extended DHCPDISCOVER messages (generated by the client Initiation step) on the standard DHCP server port (67)

and responds with DHCPOFFER messages containing the location (IP address) of the installation service.

- The installation service receives extended DHCPREQUEST messages (generated by the client Installation Service Request step) on UDP port 4011 and responds with DHCPACK messages containing the location (IP address) of the TFTP service and the file name of a new system setup executable appropriate to the client.

A standard DHCP service may be extended to include the functionality of either the redirection service or the installation service. In this case, this extended DHCP service must implement all behaviors specified for the service included.

Redirection Service Behavior

This section summarizes the behavior of the redirection service to the DHCPDISCOVER message and other DHCP messages.

Response to DHCPDISCOVER

The redirection service will always be prepared to receive on UDP port 67, an extended DHCPDISCOVER message with contents as described earlier in the "Initiation" section. The redirection service will only respond to messages which include DHCP Option 60 with the value of "PXEClient".

If the redirection service responds to a message, it will respond by sending to the initiating client a DHCPOFFER message containing options as described earlier in the "Discovery Reply" section:

The "siaddr" field in the reply, if filled in, will be the location of an installation service. If the "siaddr" field is not filled in then the installation service is at the same address as the redirection service.

The client IP address field of the message will be 0.0.0.0.

If the redirection service is also a standard DHCP configuration service, then the DHCPOFFER message sent to the client will be as specified in RFC 1541.

Installation Service Behavior

This section summarizes the behavior of the installation service to the DHCPREQUEST message and TFTP service messages.

Response to DHCPREQUEST

The installation service will always be prepared to receive a DHCPREQUEST message with contents as described earlier in the "Installation Service Request" section. The installation service will respond by sending to the initiating client a DHCPACKNOWLEDGE message as described earlier in the "Installation Service Reply" section. The file name in this message will be the complete path name of a

new system setup executable appropriate to the client that is accessible using TFTP from the installation server's IP address.

TFTP Service

The server running the installation service will provide TFTP service, as described in the previous section.

Attachment B: Preboot Execution Environment

To enable the interoperability of clients and downloaded bootstrap programs, the client preboot code must provide a set of services for use by a downloaded bootstrap. It also must ensure certain aspects of the client state at the point in time when the bootstrap begins executing. The services provided by the client for use by the bootstrap are as follows:

- **Preboot Services API.** Contains several global control and information functions.
- **TFTP API.** Enables opening and closing of TFTP connections, and reading packets from and writing packets to a TFTP connection.
- **UDP API.** Enables opening and closing UDP connections, and reading packets from and writing packets to a UDP connection.
- **Universal NIC Driver Interface (UNDI) API.** Enables basic control of and I/O through the client's network interface device.

The aspects of the client's state to be ensured by the client preboot code at the point in time that execution of the downloaded bootstrap is initiated are as follows:

- The use of certain portions of the client's main memory
- The settings of certain portions of the client's interrupt vector
- The settings of certain of the client's CPU registers

Note: The descriptions in subsequent sections are specific to Intel-architecture PCs. A processor architecture-independent description of these interface and state specifications is probably possible, but has not been attempted.

Client State at Bootstrap Execution Time

This section describes the client state, including information about the bootstrap calling convention, memory usage, and interrupt vector table.

Bootstrap Calling Convention

The entire bootstrap image is downloaded into memory starting at location 07C00h. The preboot code transfers control to the bootstrap by executing a far call to the beginning of the bootstrap code. At this point the following must be true:

- CS:IP is to contain the value 0:7C00h

- ES:BX is to contain the address of the PXENV Entry Point structure described in the “Preboot API Entry Point and Installation Check” section later in this attachment
- EDX is to contain the physical address of the PXENV Entry Point structure
- SS:SP is to contain the address of the beginning of the unused portion of the preboot services stack

Note that the bootstrap code can determine how much free stack space is available by examining the contents of SP and by having knowledge of the memory usage conventions described in the following section.

Caution: A bootstrap should not exceed 32 KB in length. The memory between 07C00h and 10000h is free for use by the bootstrap.

Memory Usage

The following table describes the usage of the first megabyte of the client’s main memory when execution of the downloaded bootstrap is initiated.

Memory Usage During Execution of Downloaded Bootstrap

Address	Status	Preboot services usage	Conventional usage
0 3FF	RESERVED, except for Vector 1Ah at 0:68h.	Vector 1Ah is used to export the preboot services API.	Interrupt Vector Table
400 4FF	RESERVED, except for the 16-bit word at 40:13h	The 16-bit word at 40:13h is the size of free base memory in KB (SFBM/400h).	BIOS Data Area
500 6FF	RESERVED		DOS Data Area
700 7BFF	RESERVED		IO.SYS Load Area
7C00 10000		Downloaded Bootstrap	
10000 10000+SFBM-1		Free base memory	
10000+SFBM (SS:SP)		Preboot Services CPU Stack (unused)	
(SS:SP)+1 9FFFF	RESERVED	Preboot Services CPU Stack (used by Preboot Services) Preboot Services Code and Data Extended BIOS Data Area (possibly)	
A0000 BFFFF	RESERVED		Video Memory
C0000 C7FFF	RESERVED		Video BIOS
C8000 DFFFF	RESERVED		Other BIOS / Upper Memory
E0000 EFFFF	RESERVED	Contains a unique system ID structure.	Other BIOS / Upper Memory / System BIOS
F0000 FFFFF	RESERVED	Contains a unique system ID structure.	System BIOS

Free Memory Size (Bios Data Area). When execution of the downloaded bootstrap begins, the 16-bit word at memory address 40:13h must contain the amount of free base memory in KB.

Preboot Services Stack. When execution of the downloaded bootstrap is begun, SS:SP is to contain the address of the top of the unused portion of the preboot services CPU stack. The downloaded image should not modify the used portion of the preboot services CPU stack prior to the time in the boot sequence when it is certain that the preboot services will not be needed again.

Preboot Services Code and Data. This memory area is reserved for the code and data that implement the preboot services. These locations should not be modified by the downloaded image prior to the time in the boot sequence, when it is certain that the preboot services will not be needed again.

Extended BIOS Data. If EBDA has been allocated, the downloaded image should not modify memory in the EBDA.

PXENV Unique System ID (SYSID Bios Area). When execution of the download bootstrap begins, the client's main memory must contain a PXENV unique system ID structure. This structure must meet the following conditions:

- Entry Point Structure - This will be found in the 000E0000h to 000FFFFFh physical address area of Memory/RAM. *The Entry Point Structure is PARAGRAPH Aligned.*

Entry Point Structure

Element	Length	Description
Header/Type	7 Bytes	_SYSID_
Checksum	1 Byte	Checksum of SYSID BIOS Entry Point Structure
Length	2 Bytes	Total length of SYSID BIOS Structure Table (Set to 011h).
SYSID BIOS Structure Table Address	4 Bytes	32 bit physical address of beginning of SYSID BIOS Structure Table. *This value is BYTE Aligned.*
Number of SYSID BIOS Structures	2 Bytes	Total number of structures within the SYSID BIOS Structure Table.
SYSID BIOS Revision	1 Byte	Revision of the SYSID BIOS Extensions (Set to 00h).

UUID Structure Format

Element	Length	Description
Header/Type	6 Bytes	_UUID_
Checksum	1 Byte	Checksum of UUID BIOS Structure
Length	2 Bytes	Total length of UUID BIOS Structure (Set to 0019h).
Variable Data Portion	16 Bytes	Actual UUID data (Initially set all bytes to 0FFh).

1. Header/Type - This is a fixed size for all SYSID BIOS Structure Types. It will always be 6 bytes long. The first and last byte will always be the underscore ascii characters. The middle four bytes are the ASCII characters of UUID.
2. Checksum - This value is a two's complement based checksum which will cause the addition of all bytes defined for this table interface to be equal to 00h. Please note that this is a 8-bit addition calculation (byte wide addition).
3. Length - This value is a Total length of the entire UUID BIOS Structure type. In other words, this value is the addition of all the bytes in this structure from the first byte of the Header/Type field to the last byte in the Variable Data Portion field. The value for this field (for 16 bytes in the Variable Data Portion) is 019h.
4. Variable Data Portion - This value is the 16 byte long (10h) UUID.

Interrupt Vector Table

When execution of the downloaded bootstrap begins, interrupt 1Ah is chained to export the preboot services, TFTP, UDP, and UNDI APIs.

Preboot API Entry Point and Installation Check

Procedures for finding the preboot API entry point structure are architecture dependent. The methods described in this section work for PC/AT x86 clients. In general, the API entry point can be discovered using either of two methods. The first method is to use the installation check interrupt, Int 1Ah. The second is to scan base memory for the preboot API entry point structure. In addition, as described earlier in the "Bootstrap Calling Convention" section, certain registers contain the address of the entry point structure when the downloaded bootstrap is executed.

The preboot API supports only 16-bit real-mode or virtual-86 mode calls. Application programs must make far calls (CALL xxxxh:yyyyh) to the functions in the preboot APIs.

Int 1Ah Function 5650h (Preboot API Installation Check)

Enter:

AX := 5650h (VP)

Exit:

AX := 564Eh (VN)

ES := 16-bit real-mode segment of the preboot API entry point structure.

BX := 16-bit real-mode offset of the preboot API entry point structure.

EDX := 32-bit physical address of the preboot API entry point structure.

All other register contents are preserved.

CF is cleared.

IF is preserved.

All other flags are undefined.

Preboot API Entry Point Structure

The preboot API entry point structure will be paragraph aligned and placed between the top of free base memory and A0000h (640k). The top of free base memory can be calculated using the size of free base memory word. This word is located in the BIOS data segment at 40:13h.

```
typedef struct s_PXENV_ENTRY {
UINT8 signature[6]; /* "PXENV+" */
UINT16 version;     /* MSB=major, LSB=minor */
UINT8 length;       /* sizeof(struct s_PXENV_ENTRY) */
UINT8 checksum;     /* 8-bit checksum off structure, */
                /* including this bytes should be 0. */
UINT16 rm_entry_off;/* 16-bit real-mode offset and segment */
UINT16 rm_entry_seg;/* of the PXENV API entry point. */
UINT16 pm_entry_off;/* 16-bit protected-mode offset and */
UINT32 pm_entry_base;   /* segment base address of the */
                /* PXENV API entry point. */
/* The PROM stack, base code and data segment selectors are only */
     /* required until the base code (TFTP API) layer is removed from */
     /* memory (this can only be done in real mode). */
        UINT16 stack_sel;         /* PROM stack segment.  Will be set */
        UINT16 stack_size;        /* to 0 when removed from memory. */
        UINT16 base_cs_sel;       /* Base code segment.  Will be set */
        UINT16 base_cs_size;      /* to 0 when removed from memory. */
        UINT16 base_ds_sel;       /* Base data segment.  Will be set */
        UINT16 base_ds_size;      /* to 0 when removed from memory. */
     /* The MLID code and data segment selectors are always required */
     /* when running the boot PROM in protected mode. */
        UINT16 mlid_ds_sel;       /* MLID data segment. */
        UINT16 mlid_ds_size;
        UINT16 mlid_cs_sel;       /* MLID code segment. */
        UINT16 mlid_cs_size;
} t_PXENV_ENTRY;
```

Register Usage for Preboot APIs

All API services use the following register settings:

Enter:

BX := PXENV function number

ES := Segment or selector of parameter structure

DI := Offset of parameter structure

Exit:

AX := EXIT_SUCCESS or EXIT_FAILURE

All other register contents are preserved.

CF := Cleared on success, set on error

IF is preserved.

All other flags are undefined.

Preboot Services API

All the fields in the preboot services API parameter structures are to be stored in little endian (Intel) format unless otherwise specified.

UNLOAD PREBOOT STACK

Op-Code: PXENV_UNLOAD_STACK

Input: ES:DI points to a t_PXENV_UNLOAD_STACK parameter structure.

Output: PXENV_EXIT_SUCCESS or PXENV_EXIT_FAILURE will be returned in AX, with the CF set accordingly. The status field in the parameter structure will be set to one of the values represented by the PXENV_STATUS_xxx constants. If successful, the address of the preboot entry point structure will also be filled in.

Description: The preboot services implementation, except for the Universal NIC Driver, will be removed from base memory. UNDI API calls will still be available.

Note: Service cannot be used in protected mode.

Warning! The contents of the preboot entry point structure will be changed by this service. The old preboot entry point structure and contents are invalid and should no longer be used. The CPU stack used by the preboot services will be discarded. The caller must switch to a local CPU stack before making this call. This service should not be used after transferring control to a downloaded OS image.

GET BINL INFO

Op-Code: `PXENV_GET_BINL_INFO`

Input: ES:DI points to a `t_PXENV_GET_BINL_INFO` parameter structure that has been initialized by the caller.

Output: `PXENV_EXIT_SUCCESS` or `PXENV_EXIT_FAILURE` will be returned in AX, with the CF set accordingly. The status field in the parameter structure will be set to one of the values represented by the `PXENV_STATUS_xxx` constants. The buffer specified in the parameter structure will be filled with the requested information.

Description: This service will return one of three buffers:

- The client's DHCPDISCOVER packet
- The DHCP server's DHCPACK packet
- The DHCPOFFER packet, which contains Option 60 set to "PXEClient" and a valid bootfile name.

In the downloaded image, the information that is returned by this service is used to configure client INI and CFG files. These files are then used to complete a valid network connection back to the configuration server.

RESTART DHCP

Op-Code: `PXENV_RESTART_DHCP`

Input: ES:DI points to a `t_PXENV_RESTART_DHCP` parameter.

Output: If DHCP cannot be restarted, `PXENV_EXIT_FAILURE` will be returned in AX, with the CF set accordingly. The status field in the parameter structure will be set to one of the values represented by the `PXENV_STATUS_xxx` constants. If DHCP is restarted, control is never returned to the caller.

Description: This service will try to establish a new DHCP connection with the server and try to start a download of a new image. The image to be downloaded will be determined by the server.

Note: It is the responsibility of the caller to make sure the network connection is in a valid state before trying to restart DHCP. Any existing network connection should be closed, and the network adapter must be shutdown using the UNDI API service `PXENV_UNDI_SHUTDOWN`.

Service cannot be used in protected mode.

RESTART TFTP

Op-Code: PXENV_RESTART_TFTP

Input: ES:DI points to a t_PXENV_RESTART_TFTP parameter structure that has been initialized by the caller. The t_PXENV_RESTART_TFTP parameter structure is identical to the t_PXENV_TFTP_OPEN parameter structure.

Output: If TFTP cannot be restarted, PXENV_EXIT_FAILURE will be returned and CF will be set. The status field in the parameter structure will be set to one of the values represented by the PXENV_STATUS_xxx constants. If TFTP is restarted, control is never returned to the caller.

Description: This service will try to establish a new TFTP connection with the server and to start a download of a new image. The image to be downloaded will be determined by the previously downloaded image.

Note: It is the responsibility of the caller to make sure the network connection is in a valid state before trying to restart TFTP. The existing network connection with the server needs to be maintained or restored. The existing TFTP connection needs to be closed.

Service cannot be used in protected mode.

MODE SWITCH

Op-Code: `PXENV_MODE_SWITCH`

Input: ES:DI points to a `t_PXENV_MODE_SWITCH` parameter structure that has been initialized by the caller.

Output: The status field in the parameter structure will be set to one of the values represented by the `PXENV_STATUS_xxx` constants.

Description: This service *must* be used when changing the processor between real mode and protected mode. The caller *must* initialize the stack, base code, base data, MLID code, and MLID data selectors and recompute the structure checksum before running this service.

Note: This service can only be called from real mode (before entering, and after leaving, protected mode.) Interrupts need to be disabled when changing the PXENV entry point structure and when calling this service.

Warning! This service can only be used with the default base code interrupt call backs.

TFTP API Service Descriptions

All the fields in the TFTP API parameter structures are to be stored in little endian (Intel) format unless otherwise specified.

TFTP OPEN

Op-Code: `PXENV_TFTP_OPEN`

Input: ES:DI points to a `t_PXENV_TFTP_OPEN` parameter structure that has been initialized by the caller. The IP addresses and port numbers in this structure are to be stored in big endian (Motorola) format.

Output: `PXENV_EXIT_SUCCESS` or `PXENV_EXIT_FAILURE` will be returned in AX, with the CF set accordingly. The `status` field in the parameter structure will be set to one of the values represented by the `PXENV_STATUS_xxx` constants.

Description: Opens a TFTP connection for reading/writing. At any one time there can be only one open connection. The connection must be closed before another can be opened.

TFTP CLOSE

Op-Code: PXENV_TFTP_CLOSE

Input: ES:DI points to a t_PXENV_TFTP_CLOSE parameter structure that has been initialized by the caller.

Output: PXENV_EXIT_SUCCESS or PXENV_EXIT_FAILURE will be returned in AX, with the CF set accordingly. The status field in the parameter structure will be set to one of the values represented by the PXENV_STATUS_xxx constants.

Description: Closes the previously opened TFTP connection.

TFTP READ

Op-Code: PXENV_TFTP_READ

Input: ES:DI points to a t_PXENV_TFTP_READ parameter structure that has been initialized by the caller.

Output: PXENV_EXIT_SUCCESS or PXENV_EXIT_FAILURE will be returned in AX, with the CF set accordingly. The status field in the parameter structure will be set to one of the values represented by the PXENV_STATUS_xxx constants. When a read is successful, the PacketNumber and PacketLength fields will also be filled in.

Description: Reads one packet from the open TFTP connection.

TFTP/MTFTP READ FILE

Op-Code: PXENV_TFTP_READ_FILE

Input: ES:DI points to a t_PXENV_TFTP_READ_FILE parameter structure that has been initialized by the caller.

Output: PXENV_EXIT_SUCCESS or PXENV_EXIT_FAILURE will be returned in AX, with the CF set accordingly. The status field in the parameter structure will be set to one of the values represented by the PXENV_STATUS_xxx constants. When a read is successful, the PacketCount and PacketLength fields will also be filled in.

Description: This service will open a TFTP, or MTFTP, connection, download the entire file and close the connection. It is up to the caller to make sure that there is enough free memory to download the file into.

For example, you cannot download a 2 MB file into base memory (below 640K).

Note: UDP open must be called before UDP read or write can be used after transferring

a file with this service.

This service cannot be call while in protected mode.

PROTECTED-MODE TFTP/MTFTP READ FILE

Op-Code: `PXENV_TFTP_READ_FILE_PMODE`

Input: ES:DI points to a `t_PXENV_TFTP_READ_FILE_PMODE` parameter structure that has been initialized by the caller.

Output: `PXENV_EXIT_SUCCESS` or `PXENV_EXIT_FAILURE` will be returned in AX, with the CF set accordingly. The status field in the parameter structure will be set to one of the values represented by the `PXENV_STATUS_xxx` constants. When a read is successful, the `PacketCount` and `PacketLength` fields will also be filled in.

Description: This service will open a TFTP or MTFTP connection, download the entire file, and close the connection. It is up to the caller to make sure that there is enough free memory to download the file into.

For example, you cannot download a 2-MB file into base memory (below 640K).

Note: UDP open must be called before UDP read or write can be used after transferring a file with this service.

This service cannot be called while in real mode.

TFTP_GET_FILE_SIZE

Op-Code: PXENV_TFTP_GET_FSIZE

Input: ES:DI points to a `t_PXENV_TFTP_Get_FSIZE` parameter structure that has been initialized by the caller.

Output: `PXENV_EXIT_SUCCESS` or `PXENV_EXIT_FAILURE` will be returned in AX, with the CF set accordingly. The status field in the parameter structure will be set to one of the values represented by the `PXENV_STATUS_xxx` constants. When the call is successful, the `FileSize` field will be filled in.

Description This service will query the server for the size of the given file using tftp option extension protocol. This service will not and hence must not be used to open a tftp connection for the given file.

Note: This service must not be called when there is an outstanding open tftp connection on the file

UDP API Service Descriptions

UDP OPEN

Op-Code: PXENV_UDP_OPEN

Input: ES:DI points to a t_PXENV_UDP_OPEN parameter.

Output: PXENV_EXIT_SUCCESS or PXENV_EXIT_FAILURE will be returned in AX, with the CF set accordingly. The status field in the parameter structure will be set to one of the values represented by the PXENV_STATUS_xxx constants.

Description: Opens a UDP connection for reading and writing. There can only be one open connection at a time.

UDP CLOSE

Op-Code: PXENV_UDP_CLOSE

Input: ES:DI points to a t_PXENV_UDP_CLOSE parameter.

Output: PXENV_EXIT_SUCCESS or PXENV_EXIT_FAILURE will be returned in AX, with the CF set accordingly. The status field in the parameter structure will be set to one of the values represented by the PXENV_STATUS_xxx constants.

Description: Closes the previously opened UDP connection.

UDP WRITE

Op-Code: PXENV_UDP_WRITE

Input: ES:DI points to a t_PXENV_UDP_WRITE parameter structure that has been initialized by the caller.

Output: PXENV_EXIT_SUCCESS or PXENV_EXIT_FAILURE will be returned in AX, with the CF set accordingly. The status field in the parameter structure will be set to one of the values represented by the PXENV_STATUS_xxx constants.

Description: Writes one packet to the specified IP address on the open UDP connection.

UDP READ

Op-Code: `PXENV_UDP_READ`

Input: ES:DI points to a `t_PXENV_UDP_READ` parameter structure that has been initialized by the caller.

Output: `PXENV_EXIT_SUCCESS` or `PXENV_EXIT_FAILURE` will be returned in AX, with the CF set accordingly. The status field in the parameter structure will be set to one of the values represented by the `PXENV_STATUS_xxx` constants.

Description: Reads one packet from the opened UDP connection.

UNDI API Service Descriptions

UNDI STARTUP

Op-Code: `PXENV_UNDI_STARTUP`

Input: ES:DI points to a `t_PXENV_UNDI_STARTUP` parameter structure that has been initialized by the caller.

Output: `PXENV_EXIT_SUCCESS` or `PXENV_EXIT_FAILURE` will be returned in AX, with the CF set accordingly. The status field in the parameter structure will be set to one of the values represented by the `PXENV_STATUS_xxx` constants.

Description: This call provides the Universal NIC Driver with necessary startup parameters, such as the data segment and network adapter identification variables. This call hooks Interrupt 1Ah to export the UNDI API. The rest of the API will not be available until this call has been completed. The data segment must be zero-filled before this API call is made.

Note: The entry point of the UNDI API must be at offset 0 of the UNDI code segment. The preboot code will install the UNDI API by making a far call to the API entry point, with ES:DI and BX setup for UNDI STARTUP.

This service cannot be used in protected mode.

UNDI CLEANUP

Op-Code: PXENV_UNDI_CLEANUP

Input: ES:DI points to a t_PXENV_UNDI_CLEANUP parameter structure.

Output: PXENV_EXIT_SUCCESS or PXENV_EXIT_FAILURE will be returned in AX, with the CF set accordingly. The status field in the parameter structure will be set to one of the values represented by the PXENV_STATUS_xxx constants.

Description: This call will uninstall the Interrupt 1Ah hook and will prepare the network adapter driver to be unloaded from memory. This call must be made just before unloading the Universal NIC Driver. The rest of the API will not be available after this call executes.

This service cannot be used in protected mode.

UNDI INITIALIZE

Op-Code: `PXENV_UNDI_INITIALIZE`

Input: ES:DI points to a `t_PXENV_UNDI_INITIALIZE` parameter structure that has been initialized by the caller.

Output: `PXENV_EXIT_SUCCESS` or `PXENV_EXIT_FAILURE` will be returned in AX, with the CF set accordingly. The status field in the parameter structure will be set to one of the values represented by the `PXENV_STATUS_xxx` constants.

Description: This call resets the adapter and programs it with default parameters. The default parameters used are those supplied to the most recent UNDI_STARTUP call. This routine does not enable the receive and transmit units of the network adapter to readily receive or transmit packets. The application must call `PXENV_UNDI_OPEN` to logically connect the network adapter to the network.

This call must be made by an application to establish an interface to the network adapter driver. The parameter block to this call contains the pointer to the call-back routines that will be called when a packet is received or when any other interrupt occurs.

Note: When a receive interrupt occurs, the network adapter driver queues the packet and calls the application's callback receive routine with a pointer to the packet received. Then, the callback routine can either copy the packet into its buffer or decide to delay the copy to a later time. The callback receive routine always gets the pointer to the first packet in the receive queue and not to the currently received packet that generated the interrupt.

If the call-back routine decides not to copy the data from the buffer at this time, the packet will remain in the receive queue and, as a result, the later packets might be dropped when the receive queue is full. At a later time, when the application wants to copy the packet, it can call the `PXENV_UNDI_FORCE_INTERRUPT` routine to simulate the receive interrupt.

When the preboot code makes this call to initialize the network adapter, it passes a NULL pointer for the `ProtocolIni` field in the parameter structure.

UNDI RESET ADAPTER

Op-Code: PXENV_UNDI_RESET_ADAPTER

Input: ES:DI points to a t_PXENV_UNDI_RESET_ADAPTER parameter structure that has been initialized by the caller.

Output: PXENV_EXIT_SUCCESS or PXENV_EXIT_FAILURE will be returned in AX, with the CF set accordingly. The status field in the parameter structure will be set to one of the values represented by the PXENV_STATUS_xxx constants.

Description: This call resets and reinitializes the network adapter with the same set of parameters supplied to Initialize Routine. Unlike Initialize, this call opens the adapter, that is, it connects logically to the network. This routine cannot be used to replace Initialize or Shutdown calls.

UNDI SHUTDOWN

Op-Code: PXENV_UNDI_SHUTDOWN

Input: ES:DI points to a t_PXENV_UNDI_SHUTDOWN parameter.

Output: PXENV_EXIT_SUCCESS or PXENV_EXIT_FAILURE will be returned in AX, with the CF set accordingly. The status field in the parameter structure will be set to one of the values represented by the PXENV_STATUS_xxx constants.

Description: This call resets the network adapter and leaves it in a safe state for another driver to program it.

Note: The contents of the PXENV_UNDI_STARTUP parameter structure need to be saved by the Universal NIC Driver in case PXENV_UNDI_INITIALIZE is called again.

UNDI OPEN

Op-Code: PXENV_UNDI_OPEN

Input: ES:DI points to a t_PXENV_UNDI_OPEN parameter structure that has been initialized by the caller.

Output: PXENV_EXIT_SUCCESS or PXENV_EXIT_FAILURE will be returned in AX, with the CF set accordingly. The status field in the parameter structure will be set to one of the values represented by the PXENV_STATUS_xxx constants.

Description: This call activates the adapter's network connection and sets the adapter ready to accept packets for transmit and receive.

UNDI CLOSE

Op-Code: PXENV_UNDI_CLOSE

Input: ES:DI points to a t_PXENV_UNDI_CLOSE parameter.

Output: PXENV_EXIT_SUCCESS or PXENV_EXIT_FAILURE will be returned in AX, with the CF set accordingly. The status field in the parameter structure will be set to one of the values represented by the PXENV_STATUS_xxx constants.

Description: This call disconnects the network adapter from the network. Packets cannot be transmitted or received until the network adapter is open again.

UNDI TRANSMIT PACKET

Op-Code: PXENV_UNDI_TRANSMIT

Input: ES:DI points to a t_PXENV_UNDI_TRANSMIT parameter structure that has been initialized by the caller.

Output: PXENV_EXIT_SUCCESS or the error code will be returned in AX, with the CF set accordingly. The error code will be set to one of the values represented by the PXENV_STATUS_xxx constants.

Description: This call transmits a buffer to the network. The media header for the packet can be filled by the calling protocol, but it might not be. The network adapter driver will fill it if required by the values in the parameter block. The transmission is always synchronous and blocks until the network adapter has placed the packet on the network.

UNDI SET MULTICAST ADDRESS

Op-Code: PXENV_UNDI_SET_MCAST_ADDRESS

Input: ES:DI points to a t_PXENV_TFTP_SET_MCAST_ADDRESS parameter structure that has been initialized by the caller.

Output: PXENV_EXIT_SUCCESS or PXENV_EXIT_FAILURE will be returned in AX, with the CF set accordingly. The status field in the parameter structure will be set to one of the values represented by the PXENV_STATUS_xxx constants.

Description: This call changes the current list of multicast addresses to the input list and resets the network adapter to accept it. If the number of multicast addresses is zero, multicast is disabled.

UNDI SET STATION ADDRESS

Op-Code: PXENV_UNDI_SET_STATION_ADDRESS

Input: ES:DI points to a t_PXENV_UNDI_SET_STATION_ADDRESS parameter structure that has been initialized by the caller.

Output: PXENV_EXIT_SUCCESS or PXENV_EXIT_FAILURE will be returned in AX, with the CF set accordingly. The status field in the parameter structure will be set to one of the values represented by the PXENV_STATUS_xxx constants.

Description: This call sets the MAC address to be the input value and is called before opening the network adapter. Later, the open call uses this variable as a temporary MAC address to program the adapter's individual address registers.

UNDI SET PACKET FILTER

Op-Code: PXENV_UNDI_SET_PACKET_FILTER

Input: ES:DI points to a t_PXENV_UNDI_SET_PACKET_FILTER parameter structure that has been initialized by the caller.

Output: PXENV_EXIT_SUCCESS or PXENV_EXIT_FAILURE will be returned in AX, with the CF set accordingly. The status field in the parameter structure will be set to one of the values represented by the PXENV_STATUS_xxx constants.

Description: This call resets the adapter's receive unit to accept a new filter, different from the one provided with the open call.

UNDI GET INFORMATION

Op-Code: PXENV_UNDI_GET_INFORMATION

Input: ES:DI points to a t_PXENV_UNDI_GET_INFORMATION parameter structure that has been initialized by the caller.

Output: PXENV_EXIT_SUCCESS or PXENV_EXIT_FAILURE will be returned in AX, with the CF set accordingly. The status field in the parameter structure will be set to one of the values represented by the PXENV_STATUS_xxx constants.

Description: This call copies the network adapter variables, including the MAC address, into the input buffer.

UNDI GET STATISTICS

Op-Code: PXENV_UNDI_GET_STATISTICS

Input: ES:DI points to a t_PXENV_UNDI_GET_STATISTICS parameter structure that has been initialized by the caller.

Output: PXENV_EXIT_SUCCESS or PXENV_EXIT_FAILURE will be returned in AX, with the CF set accordingly. The status field in the parameter structure will be set to one of the values represented by the PXENV_STATUS_xxx constants.

Description: This call reads statistical information from the network adapter, and returns.

UNDI CLEAR STATISTICS

Op-Code: PXENV_UNDI_CLEAR_STATISTICS

Input: ES:DI points to a t_PXENV_UNDI_CLEAR_STATISTICS parameter.

Output: PXENV_EXIT_SUCCESS or PXENV_EXIT_FAILURE will be returned in AX, with the CF set accordingly. The status field in the parameter structure will be set to one of the values represented by the PXENV_STATUS_xxx constants.

Description: This call clears the statistical information from the network adapter.

UNDI INITIATE DIAGS

Op-Code: PXENV_UNDI_INITIATE_DIAGS

Input: ES:DI points to a t_PXENV_UNDI_INITIATE_DIAGS parameter.

Output: PXENV_EXIT_SUCCESS or PXENV_EXIT_FAILURE will be returned in AX, with the CF set accordingly. The status field in the parameter structure will be set to one of the values represented by the PXENV_STATUS_xxx constants.

Description: This call can be used to initiate the run-time diagnostics. It causes the network adapter to run hardware diagnostics and to update its status information.

UNDI FORCE INTERRUPT

Op-Code: PXENV_UNDI_FORCE_INTERRUPT

Input: ES:DI points to a t_PXENV_UNDI_FORCE_INTERRUPT parameter structure that has been initialized by the caller.

Output: PXENV_EXIT_SUCCESS or PXENV_EXIT_FAILURE will be returned in AX, with the CF set accordingly. The status field in the parameter structure will be set to one of the values represented by the PXENV_STATUS_xxx constants.

Description: This call forces the network adapter to generate an interrupt. When a receive interrupt occurs, the network adapter driver usually queues the packet and calls the application's callback receive routine with a pointer to the packet received. Then, the callback routine either can copy the packet to its buffer or can decide to delay the copy to a later time. If the packet is not immediately copied, the network adapter driver does not remove it from the input queue. When the application wants to copy the packet, it can call the PXENV_UNDI_FORCE_INTERRUPT routine to simulate the receive interrupt.

UNDI GET MULTICAST ADDRESS

Op-Code: PXENV_UNDI_GET_MCAST_ADDRESS

Input: ES:DI points to a t_PXENV_GET_MCAST_ADDRESS parameter structure that has been initialized by the caller.

Output: PXENV_EXIT_SUCCESS or PXENV_EXIT_FAILURE will be returned in AX, with the CF set accordingly. The status field in the parameter structure will be set to one of the values represented by the PXENV_STATUS_xxx constants.

Description: This call converts the given IP multicast address to a hardware multicast address.

UNDI GET NIC TYPE

Op-Code: PXENV_UNDI_GET_NIC_TYPE

Input: ES:DI points to a t_ PXENV_UNDI_GET_NIC_TYPE parameter structure that has been initialized by the caller.

Output: PXENV_EXIT_SUCCESS or PXENV_EXIT_FAILURE will be returned in AX, with the Carry Flag set accordingly. The status field in the parameter structure will be set to one of the values represented by the PXENV_STATUS_xxx constants. If the PXENV_EXIT_SUCCESS is returned the parameter structure will contain the NIC information.

Description: This call, if successful, provides the NIC specific information necessary to identify

the network adapter that is used to boot the system.

Note: The application first gets the DHCP discover packet using GET_BINL_INFO and checks if the UNDI is supported before making this call. If the UNDI is not supported, the NIC specific information can be obtained from the DHCP discover packet itself.

Attachment C: Preboot API Common Type Definitions

Important: The code provided in this attachment is provided for informational purposes only.

```
/*
 *
* Copyright(c) 1997 by Intel Corporation.  All Rights Reserved.
 *
 */

#ifndef _PXENV_CMN_H
#define _PXENV_CMN_H

/* = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
= = */
/* PXENV.H - PXENV/TFTP/UNDI API common, Version 2.x, 97-Jan-17
 *
 * Constant and type definitions used in other PXENV API header files.
 */


/* = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
= = */
/* Parameter/Result structure storage types.
 */
typedef unsigned char UINT8;
typedef unsigned short UINT16;
typedef unsigned long UINT32;
typedef signed char INT8;
typedef signed short INT16;
typedef signed long INT32;


/* = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
= = */
/* Result codes returned in AX by a PXENV API service.
 */
#define PXENV_EXIT_SUCCESS   0x0000
#define PXENV_EXIT_FAILURE   0x0001
#define PXENV_EXIT_CHAIN     0xFFFF  /* used internally */


/* = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
= = */
/* CPU types
 */
#define PXENV_CPU_X86        0
#define PXENV_CPU_ALPHA      1
```

```
#define PXENV_CPU_PPC       2


/* = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
= = */
/* Bus types
 */
#define PXENV_BUS_ISA       0
#define PXENV_BUS_EISA      1
#define PXENV_BUS_MCA       2
#define PXENV_BUS_PCI       3
#define PXENV_BUS_VESA      4
#define PXENV_BUS_PCMCIA    5


/* = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = */
/* Status codes returned in the status word of PXENV API parameter
structures.
 */

/* Generic API errors that are reported by the loader*/
#define PXENV_STATUS_SUCCESS                              0x00
#define PXENV_STATUS_FAILURE                              0x01
/* General failure. */
#define PXENV_STATUS_BAD_FUNC                             0x02
/* Invalid function number. */
#define PXENV_STATUS_UNSUPPORTED                          0x03
/* Function is not yet supported. */
#define PXENV_STATUS_1A_HOOKED                            0x04
/* Int 1Ah cannot be unhooked. */

/* ARP errors (0x10 to 0x1F) */
#define PXENV_STATUS_ARP_CANCELED_BY_KEYSTROKE            0x10
#define PXENV_STATUS_ARP_TIMEOUT                          0x11

/* BIOS/system errors (0x20 to 0x2F) */
#define PXENV_STATUS_MCOPY_PROBLEM                        0x20

/* TFTP errors (0x30 to 0x3F) */
#define PXENV_STATUS_TFTP_CANNOT_ARP_ADDRESS              0x30
#define PXENV_STATUS_TFTP_OPEN_CANCELED_BY_KEYSTROKE      0x31
#define PXENV_STATUS_TFTP_OPEN_TIMEOUT                    0x32
#define PXENV_STATUS_TFTP_UNKNOWN_OPCODE                  0x33
#define PXENV_STATUS_TFTP_ERROR_OPCODE                    0x34
#define PXENV_STATUS_TFTP_READ_TIMEOUT                    0x35
#define PXENV_STATUS_TFTP_ERROR_OPCODE                    0x36
#define PXENV_STATUS_TFTP_CANNOT_OPEN_CONNECTION          0x38
#define PXENV_STATUS_TFTP_CANNOT_READ_FROM_CONNECTION     0x39
#define PXENV_STATUS_TFTP_TOO_MANY_PACKAGES               0x3A
#define PXENV_STATUS_TFTP_FILE_NOT_FOUND                  0x3B
```

```
#define PXENV_STATUS_TFTP_ACCESS_VIOLATION              0x3C
#define PXENV_STATUS_TFTP_NO_MCAST_ADDRESS              0x3D

/* BOOTP errors 0x40 to 0x4F) */
#define PXENV_STATUS_BOOTP_CANCELED_BY_KEYSTROKE        0x40
#define PXENV_STATUS_BOOTP_TIMEOUT                      0x41
#define PXENV_STATUS_BOOTP_NO_BOOTFILE_NAME             0x43

/* DHCP errors (0x50 to 0x5F) */
#define PXENV_STATUS_DHCP_CANCELED_BY_KEYSTROKE         0x50
#define PXENV_STATUS_DHCP_TIMEOUT                       0x51
#define PXENV_STATUS_DHCP_NO_IP_ADDRESS                 0x52
#define PXENV_STATUS_DHCP_NO_BOOTFILE_NAME              0x53

/* Driver errors (0x60 to 0x6F) */
/* These errors are for UNDI compatible NIC drivers. */
#define PXENV_STATUS_UNDI_MEDIATEST_FAILED              0x61
#define PXENV_STATUS_UNDI_CANNOT_INIT_NIC_FOR_MCAST     0x62

/* Bootstrap (.1) errors (0x70 to 0x7F) */
/* These errors are for the LSA/LCM bootstrap layer. */

/* Environment (.2) errors (0x80 to 0x8F) */
/* These errors are for LSA/LCM environment layers. */

/* MTFTP errors */
#define PXENV_STATUS_MTFTP_OPEN_CANCELED_BY_KEYSTROKE   0x91
#define PXENV_STATUS_MTFTP_OPEN_TIMEOUT                 0x92
#define PXENV_STATUS_MTFTP_UNKNOWN_OPCODE               0x93
#define PXENV_STATUS_MTFTP_READ_CANCELED_BY_KEYSTROKE   0x94
#define PXENV_STATUS_MTFTP_READ_TIMEOUT                 0x95
#define PXENV_STATUS_MTFTP_ERROR_OPCODE                 0x96
#define PXENV_STATUS_MTFTP_CANNOT_OPEN_CONNECTION       0x98
#define PXENV_STATUS_MTFTP_CANNOT_READ_FROM_CONNECTION  0x99
#define PXENV_STATUS_MTFTP_TOO_MANY_PACKAGES            0x9A

/* Misc errors (0xA0 to 0xAF) */
#define PXENV_STATUS_BINL_CANCELED_BY_KEYSTROKE         0xA0
#define PXENV_STATUS_BINL_NO_PXE_SERVER                 0xA1
#define PXENV_STATUS_NOT_AVAILABLE_IN_PMODE             0xA2
#define PXENV_STATUS_NOT_AVAILABLE_IN_RMODE             0xA3
/* Reserved errors (0xB0 to 0xCF) */

/* Vendor errors (0xD0 to 0xFF) */

#endif /* _PXENV_CMN_H */

/* EOF - $Workfile:   pxe_cmn.h  $ */
```

Attachment D: Preboot API Parameter Structure and Type Definitions

Important: The code provided in this attachment is provided for informational purposes only.

```
/*
 *
* Copyright(c) 1997 by Intel Corporation.  All Rights Reserved.
 *
 */

#ifndef _PXENV_API_H
#define _PXENV_API_H

/* = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
= = */
/*  Parameter structure and type definitions for PXENV API version 2.x
 *
 *  PXENV.H needs to be #included before this file.
 *
 *  None of the PXENV API services are available after the stack
 *  has been unloaded.
 */

#include "bootp.h"          /* Defines BOOTPLAYER */


/* = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
= = */
/* Format of PXENV entry point structure.
 */
typedef struct s_PXENV_ENTRY {
    UINT8 signature[6];      /* 'PXENV+' */
    UINT16 version;          /* MSB=major, LSB=minor */
    UINT8 length;            /* sizeof(struct s_PXENV_ENTRY) */
    UINT8 checksum;          /* 8-bit checksum off structure, */
                    /* including this bytes should */
                    /* be 0. */
    UINT16 rm_entry_off;     /* 16-bit real-mode offset and */
    UINT16 rm_entry_seg;     /* segment of the PXENV API entry */
                    /* point. */
    UINT16 pm_entry_off;     /* 16-bit protected-mode offset */
    UINT32 pm_entry_seg;     /* and segment base address of */
                             /* the PXENV API entry point. */
    UINT16 stack_sel;        /* PROM stack segment.  Will be set */
    UINT16 stack_size;       /* to 0 when removed from memory. */
```

```
    UINT16 base_cs_sel;     /* Base code segment.  Will be set */
    UINT16 base_cs_size;    /* to 0 when removed from memory. */

    UINT16 base_ds_sel;     /* Base data segment.  Will be set */
    UINT16 base_ds_size;    /* to 0 when removed from memory. */

    /* The MLID code and data segment selectors are always required */
    /* when running the boot PROM in protected mode. */

    UINT16 mlid_ds_sel;     /* MLID data segment. */
    UINT16 mlid_ds_size;

    UINT16 mlid_cs_sel;     /* MLID code segment. */
    UINT16 mlid_cs_size;

} t_PXENV_ENTRY;

#define PXENV_ENTRY_SIG         "PXENV+"


/* = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
= = */
/* One of the following command op-codes needs to be loaded into the
 * op-code register (BX) before making a call a PXENV API service.
 */
#define PXENV_UNLOAD_STACK      0x70
#define PXENV_GET_BINL_INFO     0x71
#define PXENV_RESTART_DHCP      0x72
#define PXENV_RESTART_TFTP      0x73
#define PXENV_MODE_SWITCH       0x74


/* = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
= = */
/* PXENV API parameter structure typedefs.
 */

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - */
typedef struct s_PXENV_UNLOAD_STACK {
    UINT16 status;          /* Out: See PXENV_STATUS_xxx */
                    /*      constants. */
    UINT16 rm_entry_off;    /* Out: 16-bit real-mode segment and */
    UINT16 rm_entry_seg;    /*      offset of PXENV Entry Point */
                    /*      structure. */
    UINT16 pm_entry_off;    /* Out: 16-bit protected-mode offset */
    UINT32 pm_entry_base;       /*      and segment base address of */
                    /*      PXENV Entry Point structure. */
} t_PXENV_UNLOAD_STACK;
```

```
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - */
/* Packet types that can be requested in the s_PXENV_GET_BINL_INFO
structure. */
#define PXENV_PACKET_TYPE_DHCP_DISCOVER  1
#define PXENV_PACKET_TYPE_DHCP_ACK       2
#define PXENV_PACKET_TYPE_BINL_REPLY     3

/* Three packets are preserved and available through this interface: 1)
The
 * DHCP Discover packet sent by the client, 2) the DHCP acknowledgement
 * packet returned by the DHCP server, and 3) the reply packet from the
BINL
 * server.  If the DHCP server provided the image bootfile name, the
 * DHCP_ACK and BINL_REPLY packets will identical.
 */

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - */
typedef struct s_PXENV_GET_BINL_INFO {
    UINT16 status;          /* Out: See PXENV_STATUS_xxx */
                    /*      constants. */
    UINT16 packet_type;     /* In: See PXENV_PACKET_TYPE_xxx */
                    /*     constants */
    UINT16 buffer_size;     /* In: Size of the buffer in */
                    /*     bytes.  Specifies the maximum */
                    /*     amount of data that will be */
                    /*     copied by the service.  A size */
                    /*     of zero is valid. */
                    /* Out: Amount of BINL data, in */
                    /*      bytes, that was copied into */
                    /*      the buffer.  For an input */
                    /*      size of zero, no data will be */
                    /*      copied and buffer_size will */
                    /*      be set to the maximum amount */
                    /*      of data available to be */
                    /*      copied. */
    UINT16 buffer_offset;       /* In: 16-bit offset and segment */
    UINT16 buffer_segment;      /*     selector of a buffer where the */
                    /*     requested packet will be */
                    /*     copied. */
                    /*Out: If buffer_size, buffer_offset and */
                    /*     buffer_segment are all zero;  */
                    /*     buffer_offset and buffer_segment */
                    /*     will be changed to point at the */
                    /*     packet buffers in the base code. */
} t_PXENV_GET_BINL_INFO;
```

```
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - */
typedef struct s_PXENV_RESTART_DHCP {
    UINT16 status;          /* Out: See PXENV_STATUS_xxx */
                    /*      constants. */
} t_PXENV_RESTART_DHCP;


/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - */
#define s_PXENV_RESTART_TFTP s_PXENV_TFTP_READ_FILE
#define t_PXENV_RESTART_TFTP t_PXENV_TFTP_READ_FILE


typedef struct s_PXENV_MODE_SWITCH {
    UINT16 status;  /* Out: See PXENV_STATUS_xxx constants*/
    UINT16 pxenv_entry_off;   /* In: Offset of PXENV entry point */
                              /*     structure. */
    UINT16 pxenv_entry_seg;   /* In: Real-mode segment or protected- */
                              /*     mode selector of the PXENV */
                              /*     entry point structure. */

    /* Protected-mode status call-back API is documented below. */

    UINT16 pmode_status_off;      /* In: Offset of 16-bit protected */
                                  /*     mode status call-back. */
    UINT16 pmode_status_sel;      /* In: Selector of 16-bit protected */
                                  /*     mode status call-back. */
} t_PXENV_MODE_SWITCH;

/*
 * The protected-mode call back will be used by the base code when the
 * client PC is in protected-mode and pmode_status_sel is non-zero.
 *
 * The base code will call the status call-back
 * with the following registers:
 *      AX = 0 (Inside a time-out loop.)
 *      AX = 1 - n (Packet number of received TFTP packet.)
 *      All other registers and flags are undefined.
 *
 * The call-back will return a continue/cancel flag
 * in the following registers:
 *      AX = 0 (continue)
 *      AX = 1 (cancel)
 * All other AX values are undefined, and will be treated as cancel.
 * All other registers and flags must be unchanged.
 */} t_PXENV_MODE_SWITCH;

#endif /* _PXENV_API_H */
```

```
/* EOF - $Workfile:   pxe_api.h  $ */
```

Attachment E: TFTP API Parameter Structure and Type Definitions

Important: The code provided in this attachment is provided for informational purposes only.

```
/*
* Copyright(c) 1997 by Intel Corporation.  All Rights Reserved.
 *
 */

/* TFTP_API.H
 *  Parameter structure and type definitions for TFTP API version 2.x
 *
 *  PXENV.H needs to be #included before this file.
 *
 *  None of the TFTP API services are available after the stack
 *  has been unloaded.
 */

#ifndef _TFTP_API_H
#define _TFTP_API_H


#include "pxe_cmn.h"


/* = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
= = */
/* #defines and constants
 */

/* One of the following command op-codes needs to be loaded into the
 * op-code register (BX) before making a call a TFTP API service.
 */
#define PXENV_TFTP_OPEN             0x20
#define PXENV_TFTP_CLOSE            0x21
#define PXENV_TFTP_READ             0x22
#define PXENV_TFTP_READ_FILE        0x23
#define PXENV_TFTP_READ_FILE_PMODE  0x24
#define PXENV_TFTP_GET_FSIZE         0x25


/* Definitions of TFTP API parameter structures.
 */

typedef struct s_PXENV_TFTP_OPEN {
```

```
    UINT16 Status;          /* Out: See PXENV_STATUS_xxx */
                            /*      constants. */
    UINT8 ServerIPAddress[4];   /* In: 32-bit server IP */
                                /*     address. Big-endian. */
    UINT8 GatewayIPAddress[4];  /* In: 32-bit gateway IP */
                                /*     address. Big-endian. */
    UINT8 FileName[128];
    UINT16 TFTPPort;        /* In: Socket endpoint at */
                            /*     which the TFTP server is */
                            /*     listening to requests. */
                            /*     Big-endian. */
} t_PXENV_TFTP_OPEN;


typedef struct s_PXENV_TFTP_GET_FSIZE {
    UINT16 Status;              /* Out: See PXENV_STATUS_xxx */
                                /*      constants. */
    UINT8 ServerIPAddress[4];   /* In: 32-bit server IP */
                                /*     address. Big-endian. */
    UINT8 GatewayIPAddress[4];  /* In: 32-bit gateway IP */
                                /*     address. Big-endian. */
    UINT8 FileName[128];
    UINT32 FileSize;            /* Out: File Size */
} t_PXENV_TFTP_GET_FSIZE;


typedef struct s_PXENV_TFTP_CLOSE {
    UINT16 Status;          /* Out: See PXENV_STATUS_xxx */
                    /*      constants. */
} t_PXENV_TFTP_CLOSE;


typedef struct s_PXENV_TFTP_READ {
    UINT16 Status;          /* Out: See PXENV_STATUS_xxx */
                    /*      constants. */
    UINT16 PacketNumber;    /* Out: 16-bit packet number. */
    UINT16 BufferSize;      /* In: Size of the receive */
                    /*     buffer in bytes. */
                    /* Out: Size of the packet */
                    /*      written into the buffer. */
    UINT16 BufferOffset;    /* In: Segment/Selector and */
    UINT16 BufferSegment;       /*     offset of the receive buffer. */
                    /* Out: Unchanged */
} t_PXENV_TFTP_READ;


typedef struct s_PXENV_TFTP_READ_FILE {
    UINT16 Status;          /* Out: See PXENV_STATUS_xxx */
                    /*      constants. */
    UINT8 FileName[128];    /* In: file to be read */
```

```
    UINT32 BufferSize;      /* In: Size of the receive */
                    /*     buffer in bytes. */
                    /* Out: Size of the file */
                    /*      written into the buffer. */
    UINT32 BufferOffset;    /* In: 32-bit physical address of the */
                    /*     buffer to load the file into. */
    UINT8 ServerIPAddress[4];   /* In: 32-bit server IP */
                    /*     address. Big-endian. */
    UINT8 GatewayIPAddress[4];  /* In: 32-bit gateway IP */
                    /*     address. Big-endian. */
    UINT8 McastIPAddress[4];    /* In: 32-bit multicast IP address */
                    /*     on which file can be received */
                    /*     can be null for unicast */
    UINT16 TFTPClntPort;        /* In: Socket endpoint on the Client */
                    /*     at which the file can be */
                    /*     received in case of Multicast */
    UINT16 TFTPSrvPort;     /* In: Socket endpoint at which */
                    /*     server listens for requests. */
    UINT16 TFTPOpenTimeOut;     /* In: Timeout value in seconds to be */
                    /*     used for receiving data or ACK */
                    /*     packets.  If zero, default */
                    /*     TFTP-timeout is used. */
    UINT16 TFTPReopenDelay;     /* In: wait time in seconds to delay */
                    /*     a reopen request in case of */
                    /*     multicast. */
} t_PXENV_TFTP_READ_FILE;

                        typedef struct
                        s_PXENV_TFTP_READ_FILE_PMODE {
                               UINT16 Status; /* Out: See
                        PXENV_STATUS_xxx */
                                       /*      constants. */
                               UINT8 FileName[128];  /* In: file
                        to be read */
                               UINT32 BufferSize;    /* In: Size
                        of the receive */
                                       /*     buffer in bytes. */
                                       /* Out: Size of the file */
                                       /*      written into the
                        buffer. */
                               UINT32 BufferOffset;  /* In: 32-
                        bit physical address of the */
                                       /*     buffer to load the
                        file into. */
                          UINT16 BufferSelector;    /* In: This
                        field must be set to 0 in */
                                                     /*     real-
                        mode, and to a valid data */
                                                     /*
                        selector in protected-mode. */
```

```
        UINT8 ServerIPAddress[4];     /*
In: 32-bit server IP */
                /*     address. Big-endian.
*/
        UINT8 GatewayIPAddress[4]; /* In:
32-bit gateway IP */
                /*     address. Big-endian.
*/
        UINT8 McastIPAddress[4];      /*
In: 32-bit multicast IP address */
                /*     on which file can be
received */
                /*     can be null for
unicast */
        UINT16 TFTPClntPort;   /* In:
Socket endpoint on the Client */
                /*     at which the file
can be */
                /*     received in case of
Multicast */
        UINT16 TFTPSrvPort;    /* In:
Socket endpoint at which */
                /*     server listens for
requests. */
        UINT16 TFTPOpenTimeOut;       /*
In: Timeout value in seconds to be */
                /*     used for receiving
data or ACK */
                /*     packets.  If zero,
default */
                /*     TFTP-timeout is
used. */
        UINT16 TFTPReopenDelay;       /*
In: wait time in seconds to delay */
                /*     a reopen request in
case of */
                /*     multicast. */
} t_PXENV_TFTP_READ_FILE_PMODE;

/* Note:
    If the McastIPAddress specifies a non-zero value, the TFTP_ReadFile
call
    tries to listen for multicast packets on the TFTPClntPort before
    opening a TFTP/MTFTP connection to the server.
    If it receives any packets (and not all) or if does not receive any,
    it waits for specified time and tries to reopen a multicast
connection
    to the server.
    If the server supports multicast, it notifies the acknowledging
client
```

```
    with a unicast and starts sending (multicast) the file.
    If the multicast open request times out, the client tries to connect
to
    the server at TFTP server port for a unicast transfer.
*/


#endif /* _TFTP_API_H */

/* EOF - $Workfile:   tftp_api.h  $ */
```

Attachment F: UDP API Constant and Type Definitions

Important: The code provided in this attachment is provided for informational purposes only.

```
/*
 *
* Copyright(c) 1997 by Intel Corporation.  All Rights Reserved.
 *
 */

#ifndef _UDP_API_H
#define _UDP_API_H


#include "pxe_cmn.h"


/* = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
= = */
/* #defines and constants
 */

#define PXENV_UDP_OPEN  0x30
#define PXENV_UDP_CLOSE 0x31
#define PXENV_UDP_READ  0x32
#define PXENV_UDP_WRITE 0x33


/* = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
= = */
/* Typedefs
 */

typedef struct s_PXENV_UDP_OPEN {
    UINT16 status;          /* Out: See PXENV_STATUS_xxx #defines. */
    UINT8 src_ip[4];    /* Out: 32-bit IP address of this station */

} t_PXENV_UDP_OPEN;


typedef struct s_PXENV_UDP_CLOSE {
    UINT16 status;          /* Out: See PXENV_STATUS_xxx #defines. */
} t_PXENV_UDP_CLOSE;


typedef struct s_PXENV_UDP_READ {
    UINT16 status;          /* Out: See PXENV_STATUS_xxx #defines. */
```

```
    UINT8 src_ip[4];    /* Out: See description below */
    UINT8 dest_ip[4];       /* In/Out: See description below */
    UINT16 s_port;          /* Out: See description below */
    UINT16 d_port;          /* In/Out: See description below */
    UINT16 buffer_size;     /* In: Size of receive buffer. */
                        /* Out: Length of packet written into */
                        /*     receive buffer. */
    UINT16 buffer_off;      /* In: Segment/Selector and offset */
    UINT16 buffer_seg;      /*     of receive buffer. */
} t_PXENV_UDP_READ;

/*
src_ip: (Output)
======
UDP_READ fills this value on return with the 32-bit IP address
of the sender.

dest_ip: (Input/Output)
=======
If this field is non-zero then UDP_READ will filter the incoming
packets and accept those that are sent to this IP address.

If this field is zero then UDP_READ will accept any incoming
packet and return it's destination IP address in this field.

s_port: (Output)
=======
UDP_READ fills this value on return with the UDP port number
of the sender.

d_port: (Input/Output)
=======
If this field is non-zero then UDP_READ will filter the incoming
packets and accept those that are sent to this UDP port.

If this field is zero then UDP_READ will accept any incoming
packet and return it's destination UDP port in this field.

*/

#define UDP_READ_ANY_IP     0x0000  /* Accept packets sent to any IP. */
#define UDP_READ_CHECK_IP   0x0001  /* Only accept packets sent to a */
                    /* specific IP address. */


typedef struct s_PXENV_UDP_WRITE {
    UINT16 status;          /* Out: See PXENV_STATUS_xxx #defines. */
    UINT8 ip[4];        /* In: 32-bit destination IP address. */
    UINT8 gw[4];        /* In: 32-bit Gateway IP address. */
    UINT16 src_port;/* In: Source UDP port, assigned 2069 if given 0 */
```

```
    UINT16 dst port;        /* In: Destination UDP port */
    UINT16 buffer_size;     /* In: Length of packet in buffer. */
    UINT16 buffer_off;  /* In: Segment/Selector and offset */
    UINT16 buffer_seg;  /*     of transmit buffer. */
} t_PXENV_UDP_WRITE;


#endif /* _UDP_API_H */

/* EOF - $Workfile:   udp_api.h  $ */
```

Attachment G: UNDI API Constant and Type Definitions

Important: The code provided in this attachment is provided for informational purposes only.

```
/*
 *
* Copyright(c) 1997 by Intel Corporation.  All Rights Reserved.
 *
 */

#ifndef _UNDI_API_H
#define _UNDI_API_H

/* = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
= = */
/* UNDI_API.H
 *  Parameter structure and type definitions for TFTP API version 2.x
 *
 *  PXENV.H needs to be #included before this file.
 *
 *  All of the UNDI API services are still available after the stack
 *  has been unloaded.
 */


/* One of the following command op-codes needs to be loaded into the
 * op-code register (BX) before making a call a TFTP API service.
 */

#include "pxe_cmn.h"

#define PXENV_UNDI_STARTUP       0x0001
#define PXENV_UNDI_CLEANUP       0x0002
#define PXENV_UNDI_INITIALIZE        0x0003
#define PXENV_UNDI_RESET_NIC     0x0004
#define PXENV_UNDI_SHUTDOWN      0x0005
#define PXENV_UNDI_OPEN          0x0006
#define PXENV_UNDI_CLOSE         0x0007
#define PXENV_UNDI_TRANSMIT      0x0008
#define PXENV_UNDI_SET_MCAST_ADDR        0x0009
#define PXENV_UNDI_SET_STATION_ADDR  0x000A
#define PXENV_UNDI_SET_PACKET_FILTER 0x000B
#define PXENV_UNDI_GET_INFORMATION   0x000C
#define PXENV_UNDI_GET_STATISTICS        0x000D
#define PXENV_UNDI_CLEAR_STATISTICS  0x000E
#define PXENV_UNDI_INITIATE_DIAGS        0x000F
#define PXENV_UNDI_FORCE_INTERRUPT   0x0010
```

```
#define PXENV_UNDI_GET_MCAST_ADDR       0x0011

#define ADDR_LEN        16
#define MAXNUM_MCADDR           8


/* Definitions of TFTP API parameter structures.
 */

typedef struct s_PXENV_UNDI_MCAST_ADDR {
    UINT16 MCastAddrCount;      /* In: Number of multi-cast */
                    /* addresses. */
    UINT8 MCastAddr[MAXNUM_MCADDR][ADDR_LEN]; /* In: */
                    /* list of multi-cast addresses. */
                    /* Each address can take up to */
                    /* ADDR_LEN bytes and a maximum */
                    /* of MAXNUM_MCADDR address can */
                    /* be provided*/
} t_PXENV_UNDI_MCAST_ADDR;


typedef struct s_PXENV_UNDI_STARTUP {
    UINT16 Status;          /* Out: See PXENV_STATUS_xxx constants. */
    UINT8 BusType;          /* In: NIC bus type. */
    UINT8 AddrType;         /* 0 means DataSeg contains segment */
                    /* addess for DS; 1 means DataSegAddr */
                    /* contains 32-bit physical addr for */
                    /* the data segment. */
    UINT16 DataSeg;         /* Segment address for DS */
    UINT32 DataSegAddr;     /* In: 32-bit physical address */
                    /* of Universal NIC Driver */
                    /* data segment. */
    UINT16 DataSegSize;     /* In: Size of data segment in bytes. */
    UINT16 CodeSegSize;  /* In: Size of Code segment in bytes. */
    struct {
        UINT16 BusDevFunc; /* In: Bus, device and function numbers */
                        /* of this NIC. -1 if not PCI NIC */
        UINT16 PCI_ds_off;      /* Far pointer to PCI data structure */
        UINT16 PCI_ds_seg;
    } pci;
    struct {
        UINT16 CardSelNum;  /* In: Card select number.  */
                        /* -1 for non-PnP BBS device */
        UINT16 PnP_eh_off;  /* Far pointer to PnP expansion header */
        UINT16 PnP_eh_seg;
    } pnp;
} t_PXENV_UNDI_STARTUP;


typedef struct s_PXENV_UNDI_CLEANUP {
```

```
    UINT16 Status;          /* Out: See PXENV_STATUS_xxx constants. */
} t_PXENV_UNDI_CLEANUP;


typedef struct s_PXENV_UNDI_INITIALIZE {
    UINT16 Status;          /* Out: See PXENV_STATUS_xxx constants. */
    UINT32 ProtocolIni;     /* In: See description below */
    UINT16 ReceiveOffset;       /* In: See description below */
    UINT16 ReceiveSegment;      /* In: See description below */
    UINT16 GeneralIntOffset;/* In: See description below */
    UINT16 GeneralIntSegment;   /* In: See description below */
} t_PXENV_UNDI_INITIALIZE;


/* ProtocolIni :

    This is an input parameter and is a 32-bit physical address of
    a memory  copy of the  driver module in  the protocol.ini file
    obtained from the  Protocol Manager  driver(refer to  NDIS 2.0
    specifications).   This parameter  is basically  supported for
    the universal NDIS driver to pass the information contained in
    protocol.ini   file   to   the   NIC   driver   for   any   specific
    configuration of   the   NIC.      (Note   that   the   module
    identification in the  protocol.ini  file  was  done  by  NDIS
    itself.)  This value can be NULL for any other application
    interfacing to the Universal NIC Driver.

    ReceiveOffset,  ReceiveSegment:

    This is a pointer to the receive call-back routine and must be
    a non NULL pointer.   This  routine  will  be  called  in  the
    context  of  the  receive  interrupt  after  switching  to  an
    interrupt stack.  The parameters for the routine are passed in
    the registers which  are -  pointer to  the receive  buffer in
    ES:DI and the length of data in CX.  AX contains the length of
    the media  header starting at ES:DI,  BL contains the protocol
    id (0-unknown,  1-IP,  2-ARP,  3-RARP  and  4-others)  and  BH
    contains receive flag (0-directed/promiscuous, 1-broadcast and
    2-multicast).  It is the call-back routine's responsibility to
    initialize it's own  data segment  before starting  to execute
    and to preserve the contents of all the registers except AX.

    The  call-back can either  process the packet  or postpone the
    processing  to a later time.   It must return  a SUCCESS if it
    either copied the  packet into  its own  buffer or  decided to
    reject the  packet after  examining the  packet contents.   In
    this  case the NIC driver removes  the packet from the receive
    queue and recycles the buffer.  If the call-back does not want
    to look at the packet at this time it can return DELAY and the
    NIC  driver keeps the packet in the queue and will always give
```

```
    the  first packet's pointer to the call-back in the subsequent
    interrupts.  This  delay  may  however  cause  the  subsequent
    packets  to be dropped if  the receive queue is  full.

    If the application  decides  to  process  the  packet  it  had
    delayed  it can force the NIC driver to start the call-back by
    calling ForceInterrupt routine.

    GeneralIntOffset, GeneralIntSegment:

    This is also a pointer to a call back routine and will also be
    called  in  the  context  of  an  interrupt.    However,  this
    interrupt is  not  a  receive  interrupt  and  may  be  for  a
    1)transmit  complete, 2)post processing for a previous receive
    interrupt after releasing the interrupt stack or 3)it may be a
    software  interrupt.   The AX  register contains  the function
    code 1,  2 or 3 accordingly.  If  this routine is called for a
    transmit  complete indication, CX register contains the length
    of the packet transmitted and BX register contains the type of
    transmission  0, 1 or 2 according to 0)if the transmit was for
    a directed packet  (i.e.    neither  a  broadcast  and  nor  a
    multicast),  1)if  it  was  a  broadcast  or  2)if  it  was  a
    multicast.


    Note: This call-back  pointer  must  not  be  NULL.    If  the
    application  does not want to process any of these interrupts,
    a pointer to the routine which just returns the status must be
    provided.
*/


typedef struct s_PXENV_UNDI_SHUTDOWN {
    UINT16 Status;          /* Out: See PXENV_STATUS_xxx constants. */
} t_PXENV_UNDI_SHUTDOWN;


typedef struct s_PXENV_UNDI_RESET {
    UINT16 Status;          /* Out: See PXENV_STATUS_xxx constants. */
    t_PXENV_UNDI_MCAST_ADDR R_Mcast_Buf; /* multicast address list */
                    /* see note below  */
} t_PXENV_UNDI_RESET;

  /* Note: The  NIC  driver  does  not  remember  the  multicast
    addresses provided in any  call.    So  the  application  must
    provide the multicast address  list with all  the calls that
    reset the receive unit of the adapter.
  */
```

```
typedef struct s_PXENV_UNDI_OPEN {
    UINT16 Status;          /* Out: See PXENV_STATUS_xxx constants. */
    UINT16 OpenFlag;    /* In: See description below */
    UINT16 PktFilter;       /* In: Filter for receiving */
                    /* packet. It takes the following */
                    /* values, multiple values can be */
                    /* ORed together. */
#define FLTR_DIRECTED   0x0001  /* directed/multicast */
#define FLTR_BRDCST 0x0002  /* broadcast packets */
#define FLTR_PRMSCS 0x0004  /* any packet on LAN */
#define FLTR_SRC_RTG 0x0008  /* source routing packet */

    t_PXENV_UNDI_MCAST_ADDR McastBuffer; /* In: */
                    /* See t_PXENV_UNDI_MCAST_ADDR. */
} t_PXENV_UNDI_OPEN;

/*  OpenFlag:
    This is  an input parameter and is  adapter specific.  This is
    supported  for Universal NDIS 2.0 driver to pass down the Open
    flags  provided  by   the  protocol   driver  (See   NDIS  2.0
    specifications).  This can be zero.
*/


typedef struct s_PXENV_UNDI_CLOSE {
    UINT16 Status;      /* Out: See PXENV_STATUS_xxx constants. */
} t_PXENV_UNDI_CLOSE;


#define MAX_DATA_BLKS       8

typedef struct s_PXENV_UNDI_TBD{
    UINT16 ImmedLength;     /* In: Data buffer length in */
                    /* bytes. */
    UINT16 XmitOffset;      /* 16-bit segment & offset of the */
    UINT16 XmitSegment;     /* immediate data buffer. */
    UINT16 DataBlkCount;    /* In: Number of data blocks. */
    struct DataBlk {
        UINT8 TDPtrType;/*  /* 0 => 32 bit Phys pointer in TDDataPtr
                            /*  not supported in this version of LSA */
                            /*  1 => seg:offser in TDDataPtr which can
                            /*  be a real mode or 16-bit protected mode
                            /*  pointer */
        UINT8 TDRsvdByte;   /*   Reserved, must be zero. */
        UINT16 TDDataLen;   /*   Data block length in bytes. */
        UINT32 TDDataPtr;   /*   Far pointer to data buffer. */
    } DataBlock[MAX_DATA_BLKS];
} t_PXENV_UNDI_TBD;

typedef struct s_PXENV_UNDI_TRANSMIT {
```

```
    UINT16 Status;          /* Out: See PXENV_STATUS_xxx constants. */
    UINT8   Protocol;       /* See description below */
#define P_UNKNOWN   0
#define P_IP    1
#define P_ARP       2
#define P_RARP      3

    UINT8   XmitFlag;       /* See description below */
#define XMT_DESTADDR 0x0000      /* destination address given */
#define XMT_BROADCAST   0x0001      /* use broadcast address */
    UINT16 DestAddrOffset;      /* 16-bit segment & offset of the */
    UINT16 DestAddrSegment;     /* destination media address */
                    /*  See description below */
    UINT16 TBDOffset;       /* 16-bit segment & offset of the */
    UINT16 TBDSegment;      /* transmit buffer descriptor of type */
                    /* XmitBufferDesc   */
    UINT32 Reserved[2];     /* for future use   */
} t_PXENV_UNDI_TRANSMIT;

/*
    Protocol:

    This is the protocol  of  the  upper  layer  that  is  calling
    NICTransmit call.   If the  upper layer  has filled  the media
    header this field must be 0.

    XmitFlag:

    If  this flag is  0, the NIC  driver expects a  pointer to the
    destination media  address in the field  DestMediaAddr.  If 1,
    the   NIC  driver   fills  the   broadcast  address   for  the
    destination.

    DestAddrOffset & DestAddrSegment:

    This  is a pointer to the  hardware address of the destination
    media.  It  can be  null if  the destination  is not  known in
    which case the XmitFlag contains 1 for broadcast.  Destination
    media address  must be  obtained by  the upper  level protocol
    (with  Address Resolution Protocol) and NIC driver does not do
    any address resolution.

*/

typedef struct s_PXENV_UNDI_SET_MCAST_ADDR {
    UINT16 Status;          /* Out: See PXENV_STATUS_xxx constants. */
    t_PXENV_UNDI_MCAST_ADDR McastBuffer; /* In: */
                    /* See t_PXENV_UNDI_MCAST_ADDR. */
} t_PXENV_UNDI_SET_MCAST_ADDR;
```

```
typedef struct s_PXENV_UNDI_SET_STATION_ADDR {
    UINT16 Status;          /* Out: See PXENV_STATUS_xxx constants. */
    UINT8  StationAddress[ADDR_LEN]; /* new address to be set */
} t_PXENV_UNDI_SET_STATION_ADDR;


typedef struct s_PXENV_UNDI_SET_PACKET_FILTER {
    UINT16 Status;          /* Out: See PXENV_STATUS_xxx constants. */
    UINT8 filter;           /* In: Receive filter value. */
                    /* see t_PXENV_UNDI_OPEN for values */
} t_PXENV_UNDI_SET_PACKET_FILTER;


typedef struct s_PXENV_UNDI_GET_INFORMATION {
    UINT16 Status;          /* Out: See PXENV_STATUS_xxx constants. */
    UINT16 BaseIo;          /* Out: Adapter's Base IO */
    UINT16 IntNumber;       /* Out: IRQ number */
    UINT16 MaxTranUnit;     /* Out: MTU */
    UINT16  HwType;         /* Out: type of protocol at hardware level
*/

#define ETHER_TYPE 1
#define EXP_ETHER_TYPE 2
#define IEEE_TYPE   6
#define ARCNET_TYPE    7

    /*  other numbers can  be obtained from  rfc1010 for "Assigned
    Numbers".  This number may not be validated by the application
    and hence adding new numbers to the list should be fine at any
    time.  */

    UINT16 HwAddrLen;       /* Out: actual length of hardware address */
    UINT8 CurrentNodeAddress[ADDR_LEN]; /* Out: Current hardware
address*/
    UINT8 PermNodeAddress[ADDR_LEN]; /* Out: Permanent hardware
address*/
    UINT16 ROMAddress;      /* Out: ROM address */
    UINT16 RxBufCt;         /* Out: receive Queue length */
    UINT16 TxBufCt;         /* Out: Transmit Queue length */
} t_PXENV_UNDI_GET_INFORMATION;


typedef struct s_PXENV_UNDI_GET_STATISTICS {
    UINT16 Status;          /* Out: See PXENV_STATUS_xxx constants. */
    UINT32 XmtGoodFrames;       /* Out: No. of successful
transmissions*/
    UINT32 RcvGoodFrames;       /* Out: No. of good frames received */
    UINT32 RcvCRCErrors;        /* Out: No. of frames with CRC error */
    UINT32 RcvResourceErrors;   /* Out: no. of frames discarded - */
```

```
                    /* Out: receive Queue full */
} t_PXENV_UNDI_GET_STATISTICS;


typedef struct s_PXENV_UNDI_CLEAR_STATISTICS {
    UINT16 Status;          /* Out: See PXENV_STATUS_xxx constants. */
} t_PXENV_UNDI_CLEAR_STATISTICS;


typedef struct s_PXENV_UNDI_INITIATE_DIAGS {
    UINT16 Status;          /* Out: See PXENV_STATUS_xxx constants. */
} t_PXENV_UNDI_INITIATE_DIAGS;


typedef struct s_PXENV_UNDI_FORCE_INTERRUPT {
    UINT16 Status;          /* Out: See PXENV_STATUS_xxx constants. */
} t_PXENV_UNDI_FORCE_INTERRUPT;


typedef struct s_PXENV_UNDI_GET_MCAST_ADDR {
    UINT16 Status;          /* Out: See PXENV_STATUS_xxx constants. */
    UINT32 InetAddr;    /* In: IP Multicast Address */
    UINT8  MediaAddr[ADDR_LEN]; /* Out: corresponding hardware */
                    /*      multicast address */
} t_PXENV_UNDI_GET_MCAST_ADDR;

#define PXENV_UNDI_GET_NIC_TYPE  0x12

typedef s_PXENV_UNDI_GET_NIC_TYPE{
UINT16 Status;  /* OUT: See PXENV_STATUS_xxx constants */
UINT8 NicType;  /* OUT: 2=PCI, 3=PnP */
union{
    struct{
       UINT16 Vendor_ID;   /* OUT:  */
       UINT16 Dev_ID;      /* OUT:  */
       UINT8 Base_Class;   /* OUT: */
       UINT8 Sub_Class;    /* OUT: */
       UINT8 Prog_Intf;    /* OUT: program interface */
       UINT8 Rev;          /* OUT: Revision number */
       UINT16 BusDevFunc;  /* OUT: Bus, Device */
                           /* & Function numbers */
    }pci;
    struct{
       UINT32 EISA_Dev_ID; /* Out:  */
       UINT8 Base_Class;   /* OUT: */
       UINT8 Sub_Class;    /* OUT: */
       UINT8 Prog_Intf;    /* OUT: program interface */
       UINT16 CardSelNum;  /* OUT: Card Selector Number */
       }pnp;
    }pci_pnp_info;
}t_PXENV_UNDI_GET_NIC_TYPE;

#endif /* _UNDI_API_H */

/* EOF - $Workfile:   undi_api.h  $ */
```

Attachment H: WMI/CIM and Win32 Extensions Instrumentation Details

The WMI/CIM required schema is a set of base classes that provide the minimal set of information supported by Net PC platforms deployed with WMI instrumented operating system. The minimal set of classes and associations are listed here and must be deployed on all Net PCs designed for shipment with WMI-instrumented and WBEM-instrumented operating systems.

Over time, the set of base classes can be extended to accommodate additional requirements. CIM incorporates an extension process whereby additional classes, properties, and associations can be introduced. These will initially be introduced as non-standard extensions, typically subclasses of a standard class or new classes associated with a standard class. Over time, the model is intended to evolve as new extensions become widely used and accepted (clearly, the better the design, the better the chance of an extension being accepted).

It is the intention of the standard set defined here to establish a baseline for Net PC management, not a comprehensive solution to all possible management scenarios. Individual OEMs can extend the schema as required to accommodate special capabilities. If these extensions are made under the class structure defined here, management applications can be expected to benefit from them without any changes being required.

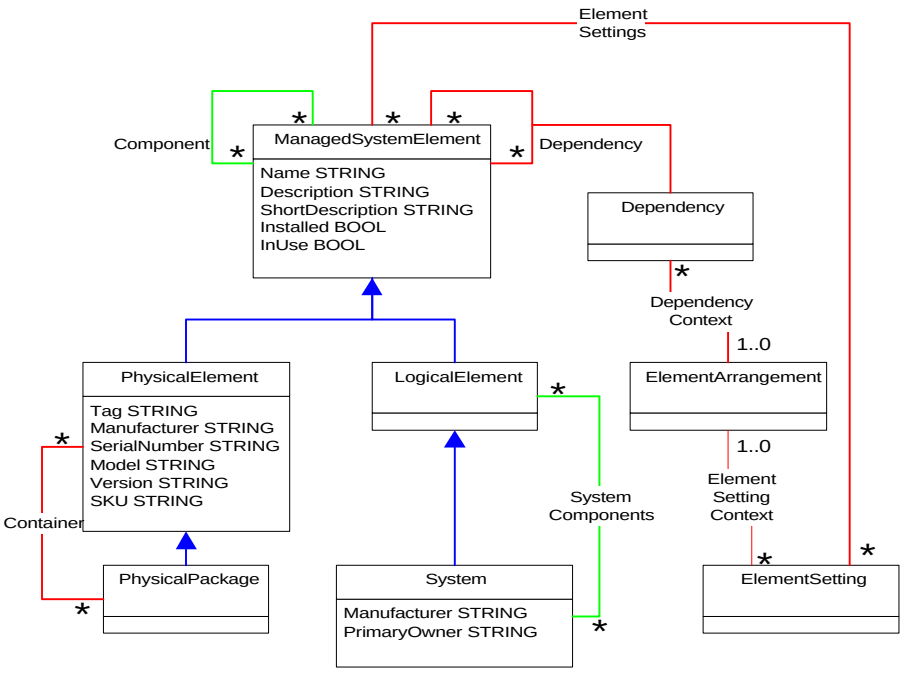
The class structure defined here is a subset of the overall CIM schema. Any extensions are required to be consistent with this broader schema. Extensions outside the logical framework of the CIM schema will require changes to management applications to take full advantage of them. As these extensions are standardized, management applications can be expected to accommodate the new capabilities with any required changes to associated algorithms and interfaces. Generic browsers, of course, can always display a new class or property without requiring any specialized extensions.

Management applications can be expected to take advantage of the CIM schema to provide, for example, different views of a system (a physical component view, a services and drivers view, a running processes view, and so on.)

An example of the extension of a standard class is the addition of a new type of service as an extension of the standard service class. Any management application that uses the standard service class will pick up instances of the new service class (as a result of inheritance), even though it will not be aware of the extension. If someone adds a new class that is not a subclass of the standard service class, yet the instances of the new class are services, then management applications will not be aware of the new class without some exception handling. The management application will have to be specially coded to go and look for the class and to present it along with the other information about services.

A similar but more complex argument applies to the use of associations.

The individual elements that make up a system can be enumerated using a number of different strategies. There are several key classes and associations involved; most are present in the diagram that follows. The strategies available for enumerating the components of a system provide a view of the schema from the perspective of a system considered as an aggregation hierarchy (as opposed to the schema as a classification hierarchy).



A function that lists the components of a system will start with a system object. The system components association relates the system to its components. The listing function must select the components to be listed based on the type of picture of the system to be presented. There is any number of alternative views depending on the circumstances at hand:

- If a list of the physical components is required, the function will list all the components that are physical elements.
- If the top-most physical elements are required, the function will list the physical elements that are not contained in anything.
- If a configuration view is required, the objects representing the physical configuration of the system will be accessed. Its dependency and context associations will be traversed to obtain the physical configuration.

- If the logical components of the system are required, the components of type logical element will be selected.
- Selecting logical elements that have nothing dependent on them would return top-level logical objects.
- Low-level elements — typically, a device-level view — could be constructed by selecting logical elements that either have no realization or have a physical element realization.
- Dependency or configuration trees could be constructed by pursuing suitable associations.

Name	SuperClass	Description
ManagedSystemElement		The base class for all system component objects. Any managed object that is a component of a system is a descendent of this class. These objects include: software components, such as files; devices, such as disk drives and controllers; and physical components, such as chips and cards.
PhysicalElement	ManagedSystemElement	Any component of a system that has a distinct physical identity and that can be defined in terms of labels that can be physically attached to the object is a member of this class. All processes, files, records, and devices are considered not to be physical elements. For example, it is not possible to attach a label to a modem. It is only possible to attach a label to the card that implements the modem. The same card could also implement a LAN adapter. These are tangible managed system elements—usually actual hardware items—that have a physical manifestation of some sort. A managed system element is not necessarily a discrete component. For example, it is possible for a single card—which is a type of physical element—to host more than one logical device. The card would be represented by a single physical element associated with multiple logical devices.
LogicalElement	ManagedSystemElement	A base class for all the components of the system that represent abstract system components, such as profiles, processes, or system capabilities in the form of logical devices.
System	LogicalElement	A grouping of other logical elements. Because systems are logical elements, a system can be composed of other systems.
Protocol	LogicalElement	Represents a protocol, which is a set of rules and algorithms that govern the interaction between two or among more than two interfaces.
SoftwareComponent	LogicalElement	Represents any software component, which can be either an individual file, such as an executable, or a collection of files, such as packages or operating systems. Software components can have additional associated information, such as the installation date.
Process	LogicalElement	A sequence of states defined by the interaction of one or more processors or interpreters, some executable code and a set of inputs.
Thread	LogicalElement	Represents a thread, which is a unit of execution; that is, an address space. Threads are owned by processes.
SystemService	LogicalElement	Represents a system service, which is a definition of a

Name	SuperClass	Description
		process owned by the system; rather than some specific user that provides an interface to some aspect of the functionality supported by the system.
Job	LogicalElement	Represents a unit of work, such as a print job.
JobDestination	LogicalElement	A process or service able to process one or more jobs.
FileSystem	LogicalElement	This object represents a set of conventions used for arranging data on a storage medium.
DiskPartition	LogicalElement	A structure used to manage the physical surface of a physical disk. There may be a level of indirection between the physical disk and the actual hardware as, for example, in the case of RAID devices.
Device	LogicalElement	A unit of functionality associated with providing the basic capabilities of a system such as input, output, or storage management. Devices may be directly expressed by a physical component as, for example, in the case of a keyboard. However, almost any device can be virtualized either by simulation (for example, simulating a modem using main CPU cycles), by allocation of a single device to multiple physical components, or by allocation of more than one device to a single physical unit. For example, a modem and LAN adapter may share the same PCMCIA card.
StorageDevice	Device	A source or destination for a file. Processors and end users typically see data in the system in terms of files, which in turn are allocated to data sources. The data source is a named unit of storage that may correspond to a variety of implementations, including memory, CDROM, and network.
Modem	Device	A device that translates binary data into wave modulations — typically, sound for transmission over telephone lines.
Processor	Device	A device capable of interpreting a sequence of machine instructions. Typically, the processor has a close correspondence to a physical chip, but it may be provided by an interpreter that is itself a process running on a processor of some kind.
Keyboard	Device	A device for entering data through keystrokes.
LogicalConnector	Device	A device capable of connecting two or more other devices.
InterfaceDevice	Device	Represents an interface device; any interface device is a descendant of this class. These are devices that act as an interface between a device and the rest of the system — for example, disk controllers, serial ports, parallel ports, and so on.

Name	SuperClass	Description
Display	Device	The device used to visually display output from the system.
MemoryModule	Device	A device capable of storing information for fast retrieval.
PointingDevice	Device	A device used to point to regions on the display.
Printer	Device	A device capable of reproducing a visual image on a medium of some kind, usually paper. Printers are a common example of a device that is also a system. The system aspect of the printer must be represented in this model as a discrete object that is a descendent of the system class.
ActualStorageDevice	Device	A device that is primarily intended to describe the organization of a physical unit used to store data.
Bus	Device	A device that provides high-bandwidth communication between different components of the system.
SCSIIInterface	InterfaceDevice	Represents a SCSII Interface device and its properties.
NetworkDrive	StorageDevice	Represents a logical drive that has been mapped to a network resource.
LogicalDiskDrive	StorageDevice	A data source that resolves to a local ActualStorageDevice.
Driver	SoftwareComponent	Represents an executable or set of executables that provide an interface either to another driver or to a logical device.
OperatingSystem	SoftwareComponent	Describes general information about operating systems installed on this system.
NetworkProtocol	Protocol	Provides information about a protocol that has been installed on the system.
ComputerSystem	System	A system that is capable of running programs, processing inputs, and displaying or otherwise returning outputs.
PhysicalPackage	PhysicalElement	Defines the characteristics of system components that physically contain other system components, such as the system enclosure, which would be a type of cabinet.
PhysicalLink	PhysicalElement	Contains any physical object used to link other objects together, which can include wires, wireless connections (radio frequencies and infrared), and so on.
PhysicalConnector	PhysicalElement	A physical element that is used to connect other Physical Elements, such as slots and plugs. This object has properties, such as the type (male or female) and the number of pins.
Slot	PhysicalConnector	Defines the attributes for the different expansion slots supported by this system.
PortConnector	PhysicalConnector	Defines the network connection points provided by the

Name	SuperClass	Description
		system.
Card	PhysicalPackage	A type of physical container that can be plugged into another card or board.
ElementSetting		ElementSetting are operational parameters that vary from time to time.
PartitionConfiguration	ElementSetting	An arrangement of partitions used to provide a basis for one or more logical disks.
Dependency		An association class that is the base class for all associations that define any dependency between managed system elements.
Component		Descendents of this association class define part of the relationships between managed system elements. For example, the system components association defines the parts of a system.
Location		The base class for all location objects.
ElementSettings		Relates an ElementSetting object to the system element it provides settings for.

Attachment I: DMI Instrumentation Details

The following standard groups from the *System Standards Group Definition, Approved Version 1.0*, must be instrumented and deployed on DMI-instrumented systems compliant with these guidelines:

- DMTF|ComponentID|001
- DMTF|Disk Mapping Table|001
- DMTF|Disks|002
- DMTF|General Information|001
- DMTF|Keyboard|003
- DMTF|Mouse|003
- DMTF|Operating System|001
- DMTF|Partition|001
- DMTF|Physical Container Global Table|001
- DMTF|Processor|003
- DMTF|System BIOS|001
- DMTF|System Cache|002
- DMTF|System Slots|003
- DMTF|Video BIOS|001
- DMTF|Video|002

The following table contains standard groups related to system resource management from the *System Standards Group Definition, Approved Version 1.0*. All these groups are valid standard groups, but the groups designated as "Replacement" groups are designed to replace the two groups marked "Original." The DMTF recommends that instrumentation migrate to the Replacement groups.

To be compliant with these guidelines, a DMI-instrumented system provides either all of the Original groups or all of the corresponding Replacement groups. It is highly recommended that the Replacement groups be selected for newly implemented instrumentation.

DMI Standard Group	Implementation Guidelines
DMTF|System Resource 2|001	Replacement, recommended for new instrumentation
DMTF|System Resource Device Info|001	Replacement, recommended for new instrumentation
DMTF|System Resource DMA Info|001	Replacement, recommended for new instrumentation
DMTF|System Resource I/O Info|001	Replacement, recommended for new instrumentation
DMTF|System Resource IRQ Info|001	Replacement, recommended for new instrumentation
DMTF|System Resource Memory Info|001	Replacement, recommended for new instrumentation
DMTF|System Resources|001	Original, recommended for legacy instrumentation only
DMTF|System Resources Description|001	Original, recommended for legacy instrumentation only

The following table contains standard groups related to physical memory management from the *System Standards Group Definition, Approved Version 1.0*. All of these groups are valid standard groups, but the groups designated as "Replacement" groups are designed to replace the group marked "Original." The DMTF recommends that instrumentation migrate to the Replacement groups. To be compliant with these guidelines, a DMI-instrumented system should be instrumented with either the Original group or the corresponding Replacement groups. It is highly recommended that the Replacement groups be selected for newly implemented instrumentation.

DMI Standard Group	Implementation Guidelines
DMTF|Memory Device|001	Replacement, recommended for new instrumentation
DMTF|Memory Array Mapped Addresses|001	Replacement, recommended for new instrumentation
DMTF|Memory Device Mapped Addresses|001	Replacement, recommended for new instrumentation
DMTF|Physical Memory Array|001	Replacement, recommended for new instrumentation
DMTF|Physical Memory|002	Original, recommended for legacy instrumentation only

The following standard groups from the *LAN Adapter Standard Groups Definition, Release Version 1.0*, must be instrumented and deployed on DMI-instrumented systems compliant with these guidelines:

- DMTF|Network Adapter 802 Port|001
- DMTF|Network Adapter Driver|001

Attachment J: Possible DMI/CIM Mappings

The following table is provided as a planning tool and will change as the DMTF completes its work on mapping DMI 2.0 groups into CIM classes. It maps the DMTF groups supported by Net PC platforms to the CIM classes supported by Net PC platforms. Note that there is generally not a one-to-one correspondence, as typically the CIM classes are more normalized than the groups, implying that properties will be spread across more than one class.

DMTF Group	CIM Equivalent
DMTF|ComponentID|001	System and associated objects
DMTF|Disk Mapping Table|001	Association between disks and partitions
DMTF|Disks|002	Elements of logical disk and physical disk
DMTF|General Information|001	System
DMTF|Keyboard|003	Keyboard and keyboard settings
DMTF|Mouse|003	Pointing device and settings
DMTF|Operating System|001	Operating system
DMTF|Partition|001	Disk partition
DMTF|Physical Container Global Table|001	System and associated physical elements
DMTF|Processor|003	Processor
DMTF|System BIOS|001	Win32BIOS and SystemROM
DMTF|System Cache|002	System cache
DMTF|System Slots|003	Logical connector and slot
DMTF|Video BIOS|001	Win32BOS
DMTF|Video|002	Display and display controller

Attachment K: UUIDs and GUIDs

Network Working Group — Paul J. Leach, Microsoft
INTERNET-DRAFT — Rich Salz, Open Group
<draft-leach-uuids-guids-00.txt>
Category: Informational
Expires August 24, 1997 — February 24, 1997

UUIDs and GUIDs

STATUS OF THIS MEMO

This document is an Internet-Draft. Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress".

To learn the current status of any Internet-Draft, please check the "1id-abstracts.txt" listing contained in the Internet-Drafts Shadow Directories on ftp.is.co.za (Africa), nic.nordu.net (Europe), munnari.oz.au (Pacific Rim), ds.internic.net (US East Coast), or ftp.isi.edu (US West Coast).

Distribution of this document is unlimited. Please send comments to the authors or the CIFS mailing list at cifs@listserv.msn.com. Discussions of the mailing list are archived at http://microsoft.ease.lsoft.com/archives/CIFS.html.

ABSTRACT

This specification defines the format of UUIDs (Universally Unique IDentifier), also known as GUIDs (Globally Unique IDentifier). A UUID is 128 bits long, and if generated according to the one of the mechanisms in this document, is either guaranteed to be different from all other UUIDs/GUIDs generated until 3400 A.D. or extremely likely to be different (depending on the mechanism chosen). UUIDs were originally used in the Network Computing System (NCS) [1] and later in the Open Software Foundation’s (OSF) Distributed Computing Environment [2].

This specification is derived from the latter specification with the kind permission of the OSF.

Table of Contents

Introduction

This specification defines the format of UUIDs (Universally Unique IDentifiers), also known as GUIDs (Globally Unique IDentifiers). A UUID is 128 bits long, and if generated according to the one of the mechanisms in this document, is either guaranteed to be different from all other UUIDs/GUIDs generated until 3400 A.D. or extremely likely to be different (depending on the mechanism chosen).

Motivation

One of the main reasons for using UUIDs is that no centralized authority is required to administer them (beyond the one that allocates IEEE 802.1 node identifiers). As a result, generation on demand can be completely automated, and they can be used for a wide variety of purposes. The UUID generation algorithm described here supports very high allocation rates: 10 million per second per machine if you need it, so that they could even be used as transaction IDs.

UUIDs are fixed-size (128-bits) which is reasonably small relative to other alternatives. This fixed, relatively small size lends itself well to sorting, ordering, and hashing of all sorts, storing in databases, simple allocation, and ease of programming in general.

Specification

A UUID is an identifier that is unique across both space and time, with respect to the space of all UUIDs. To be precise, the UUID consists of a finite bit space. Thus the time value used for constructing a UUID

is limited and will roll over in the future (approximately at A.D. 3400, based on the specified algorithm). A UUID can be used for multiple purposes, from tagging objects with an extremely short lifetime, to reliably identifying very persistent objects across a network.

The generation of UUIDs does not require that a registration authority be contacted for each identifier. Instead, it requires a unique value over space for each UUID generator. This spatially unique value is specified as an IEEE 802 address, which is usually already available to network-connected systems. This 48-bit address can be assigned based on an address block obtained through the IEEE registration authority. This section of the UUID specification assumes the availability of an IEEE 802 address to a system desiring to generate a UUID, but if one is not available section 4 specifies a way to generate a probabilistically unique one that can not conflict with any properly assigned IEEE 802 address.

Format

The following table gives the format of a UUID. The UUID consists of a record of 16 octets. The fields are in order of significance for comparison purposes, with "time_low" the most significant, and "node" the least significant.

Field	Data Type	Octet #	Note
time_low	unsigned 32 bit integer	0-3	The low field of the timestamp.
time_mid	unsigned 16 bit integer	4-5	The middle field of the timestamp.
time_hi_and_version	unsigned 16 bit integer	6-7	The high field of the timestamp multiplexed with the version number.
clock_seq_hi_and_reserved	unsigned 8 bit integer	8	The high field of the clock sequence multiplexed with the variant.
clock_seq_low	unsigned 8 bit integer	9	The low field of the clock sequence.
node	unsigned 48 bit integer	10-15	The spatially unique node identifier.

To minimize confusion about bit assignments within octets, the UUID record definition is defined only in terms of fields that are integral numbers of octets. The version number is in the most significant 4 bits of the time stamp (*time_hi*), and the variant field is in the most significant 3 bits of the clock sequence (*clock_seq_high*).

The timestamp is a 60 bit value. For UUID version 1, this is represented by Coordinated Universal Time (UTC) as a count of 100-nanosecond intervals since 00:00:00.00, 15 October 1582 (the date of Gregorian reform to the Christian calendar).

The following table lists currently defined versions of the UUID.

Msb0	**Msb1**	Msb2	Msb3	Version	Description
0	0	0	1	1	The version specified in this document.
0	0	1	0	2	Reserved for DCE Security version, with embedded POSIX UIDs.

The variant field determines the layout of the UUID. The structure of UUIDs is fixed across different versions within a variant, but not across variants; hence, other UUID variants may not interoperate with the UUID variant specified in this document. Interoperability of UUIDs is defined as the applicability of operations such as string conversion, comparison, and lexical ordering across different systems. The *variant* field consists of a variable number of the msbs of the *clock_seq_hi_and_reserved* field.

The following table lists the contents of the variant field.

Msb0	Msb1	Msb2	Description
0	-	-	Reserved, NCS backward compatibility.
1	0	-	The variant specified in this document.
1	1	0	Reserved, Microsoft Corporation GUID.
1	1	1	Reserved for future definition.

The clock sequence is required to detect potential losses of monotonicity of the clock. Thus, this value marks discontinuities and prevents duplicates. An algorithm for generating this value is outlined in the "Clock Sequence" section below.

The clock sequence is encoded in the 6 least significant bits of the *clock_seq_hi_and_reserved* field and in the *clock_seq_low* field.

The *node* field consists of the IEEE address, usually the host address. For systems with multiple IEEE 802 nodes, any available node address can be used. The lowest addressed octet (octet number 10) contains the global/local bit and the unicast/multicast bit, and is the first octet of the address transmitted on an 802.3 LAN.

Depending on the network data representation, the multi-octet unsigned integer fields are subject to byte swapping when communicated between different endian machines.

The nil UUID is special form of UUID that is specified to have all 128 bits set to 0 (zero).

Algorithms for Creating a UUID

Various aspects of the algorithm for creating a UUID are discussed in the following sections. UUID generation requires a guarantee of uniqueness within the node ID for a given variant and version. Interoperability is provided by complying with the specified data structure. To prevent possible UUID collisions, which could be caused by different implementations on the same node, compliance with the algorithm specified here is required.

Clock Sequence

The clock sequence value must be changed whenever:

- the UUID generator detects that the local value of UTC has gone backward.

- the UUID generator has lost its state of the last value of UTC used, indicating that time *may* have gone backward; this is typically the case on reboot.

While a node is operational, the UUID service always saves the last UTC used to create a UUID. Each time a new UUID is created, the current *UTC* is compared to the saved value and if either the current value is less (the non-monotonic clock case) or the saved value was lost, then the *clock sequence* is incremented modulo 16,384, thus avoiding production of duplicate UUIDs.

The *clock sequence* must be initialized to a random number to minimize the correlation across systems. This provides maximum protection against *node* identifiers that may move or switch from system to system rapidly. The initial value MUST NOT be correlated to the node identifier.

The rule of initializing the *clock sequence* to a random value is waived if, and only if all of the following are true:

- The *clock sequence* value is stored in non-volatile storage.

- The system is manufactured such that the IEEE address ROM is designed to be inseparable from the system by either the user or field service, so that it cannot be moved to another system.

- The manufacturing process guarantees that only new IEEE address ROMs are used.

- Any field service, remanufacturing or rebuilding process that could change the value of the clock sequence must reinitialise it to a random value.

In other words, the system constraints prevent duplicates caused by possible migration of the IEEE address, while the operational system itself can protect against non-monotonic clocks, except in the case of field service intervention. At manufacturing time, such a system may initialise the clock sequence to any convenient value.

System Reboot

There are two possibilities when rebooting a system:

- the UUID generator state - the last UTC, adjustment, and clock sequence - of the UUID service has been restored from non-volatile store

- the state of the last UTC or adjustment has been lost.

If the state variables have been restored, the UUID generator just continues as normal. Alternatively, if the state variables cannot be restored, they are reinitialised, and the clock sequence is changed.

If the clock sequence is stored in non-volatile store, it is incremented; otherwise, it is reinitialised to a new random value.

Clock Adjustment

UUIDs may be created at a rate greater than the system clock resolution. Therefore, the system must also maintain an adjustment value to be added to the lower-order bits of the time. Logically, each time the

system clock ticks, the adjustment value is cleared. Every time a UUID is generated, the current adjustment value is read and incremented atomically, then added to the UTC time field of the UUID.

Clock Overrun

The 100 nanosecond granularity of time should prove sufficient even for bursts of UUID creation in high-performance multiprocessors. If a system overruns the clock adjustment by requesting too many UUIDs within a single system clock tick, the UUID service may raise an exception, handled in a system or process-dependent manner either by:

- terminating the requester

- reissuing the request until it succeeds

- stalling the UUID generator until the system clock catches up.

If the processors overrun the UUID generation frequently, additional node identifiers and clocks may need to be added.

UUID Generation

UUIDs are generated according to the following algorithm:

- Determine the values for the UTC-based timestamp and clock sequence to be used in the UUID, as described above.

- For the purposes of this algorithm, consider the timestamp to be a 60-bit unsigned integer and the clock sequence to be a 14-bit unsigned integer. Sequentially number the bits in a field, starting from 0 (zero) for the least significant bit.

- Set the *time_low* field equal to the least significant 32-bits (bits numbered 0 to 31 inclusive) of the time stamp in the same order of significance.

- Set the *time_mid* field equal to the bits numbered 32 to 47 inclusive of the time stamp in the same order of significance.

- Set the 12 least significant bits (bits numbered 0 to 11 inclusive) of the *time_hi_and_version* field equal to the bits numbered 48 to 59 inclusive of the time stamp in the same order of significance.

- Set the 4 most significant bits (bits numbered 12 to 15 inclusive) of the *time_hi_and_version* field to the 4-bit version number corresponding to the UUID version being created, as shown in the table above.

- Set the *clock_seq_low* field to the 8 least significant bits (bits numbered 0 to 7 inclusive) of the *clock sequence* in the same order of significance.

- Set the 6 least significant bits (bits numbered 0 to 5 inclusive) of the *clock_seq_hi_and_reserved* field to the 6 most significant bits (bits numbered 8 to 13 inclusive) of the *clock sequence* in the same order of significance.

- Set the 2 most significant bits (bits numbered 6 and 7) of the *clock_seq_hi_and_reserved* to 0 and 1, respectively.

- Set the *node* field to the 48-bit IEEE address in the same order of significance as the address.

String Representation of UUIDs

For use in human readable text, a UUID string representation is specified as a sequence of fields, some of which are separated by single dashes.

Each field is treated as an integer and has its value printed as a zero-filled hexadecimal digit string with the most significant digit first. The hexadecimal values a to f inclusive are output as lower case characters, and are case insensitive on input. The sequence is the same as the UUID constructed type.

The formal definition of the UUID string representation is provided by the following extended BNF:

```
UUID                   = <time_low> "-" <time_mid> "-"
                         <time_high_and_version> "-"
                         <clock_seq_and_reserved>
                         <clock_seq_low> "-" <node>
time_low               = 4*<hexOctet>
time_mid               = 2*<hexOctet>
time_high_and_version  = 2*<hexOctet>
clock_seq_and_reserved = <hexOctet>
clock_seq_low          = <hexOctet>
node                   = 6*<hexOctet
hexOctet               = <hexDigit> <hexDigit>
hexDigit =
        "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
        | "a" | "b" | "c" | "d" | "e" | "f"
        | "A" | "B" | "C" | "D" | "E" | "F"
```

The following is an example of the string representation of a UUID:

```
f81d4fae-7dec-11d0-a765-00a0c91e6bf6
```

Comparing UUIDs

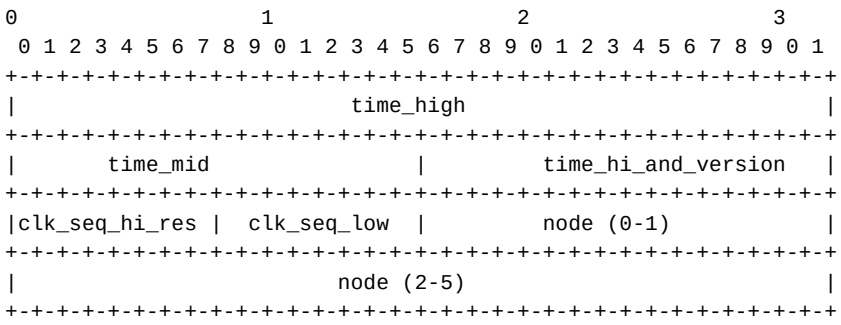
Consider each field of the UUID to be an unsigned integer as shown in the table in section 3.1. Then, to compare a pair of UUIDs, arithmetically compare the corresponding fields from each UUID in order of significance and according to their data type. Two UUIDs are equal if and only if all the corresponding fields are equal. The first of two UUIDs follows the second if the most significant field in which the UUIDs differ is greater for the first UUID. The first of a pair of UUIDs precedes the second if the most significant field in which the UUIDs differ is greater for the second UUID.

Byte order of UUIDs

UUIDs may be transmitted in many different forms, some of which may be dependent on the presentation or application protocol where the UUID may be used. In such cases, the order, sizes and byte orders of the UUIDs fields on the wire will depend on the relevant presentation or application protocol. However, it

is strongly RECOMMENDED that the order of the fields conform with ordering set out in section 3.1 above. Furthermore, the payload size of each field in the application or presentation protocol MUST be large enough that no information lost in the process of encoding them for transmission.

In the absence of explicit application or presentation protocol specification to the contrary, a UUID is encoded as a 128-bit object, as follows: the fields are encoded as 16 octets, with the sizes and order of the fields defined in section 3.1, and with each field encoded with the Most Significant Byte first (also known as network byte order).



Node IDs when no IEEE 802 network card is available

If a system wants to generate UUIDs but has no IEE 802 compliant network card or other source of IEEE 802 addresses, then this section describes how to generate one.

The ideal solution is to obtain a 47 bit cryptographic quality random number, and use it as the low 47 bits of the node ID, with the most significant bit of the first octet of the node ID set to 1. This bit is the unicast/multicast bit, which will never be set in IEEE 802 addresses obtained from network cards; hence, there can never be a conflict between UUIDs generated by machines with and without network cards.

If a system does not have a primitive to generate cryptographic quality random numbers, then in most systems there are usually a fairly large number of sources of randomness available from which one can be generated. Such sources are system specific, but often include:

- the percent of memory in use

- the size of main memory in bytes

- the amount of free main memory in bytes

- the size of the paging or swap file in bytes

- free bytes of paging or swap file

- the total size of user virtual address space in bytes

- the total available user address space bytes

- the size of boot disk drive in bytes

- the free disk space on boot drive in bytes

- the current time

- the amount of time since the system booted

- the individual sizes of files in various system directories

- the creation, last read, and modification times of files in various system directories

- the utilization factors of various system resources (heap, etc.)

- current mouse cursor position

- current caret position

- current number of running processes, threads

- handles or IDs of the desktop window and the active window

- the value of stack pointer of the caller

- the process and thread ID of caller

various processor architecture specific performance counters (instructions executed, cache misses, TLB misses)

(Note that it precisely the above kinds of sources of randomness that are used to seed cryptographic quality random number generators on systems without special hardware for their construction.)

In addition, items such as the computer's name and the name of the operating system, while not strictly speaking random, will help differentiate the results from those obtained by other systems.

The exact algorithm to generate a node ID using these data is system specific, because both the data available and the functions to obtain them are often very system specific. However, assuming that one can concatenate all the values from the randomness sources into a buffer, and that a cryptographic hash function such as MD5 [3] is available, the following code will compute a node ID:

```
#include <md5.h>
#define HASHLEN 16

void GenNodeID(
    unsigned char * pDataBuf,    // concatenated "randomness values"
    long cData,          // size of randomness values
    unsigned char NodeID[6]      // node ID
)
{
  int i, j, k;
  unsigned char Hash[HASHLEN];
  MD_CTX context;

  MDInit (&context);
```

```
  MDUpdate (&context, pDataBuf, cData);
  MDFinal (Hash, &context);

  for (j = 0; j<6; j++) NodeId[j]=0;
  for (i = 0,j = 0; i < HASHLEN; i++) {
        NodeID[j++] ^= Hash[i];
        if (j == 6) j = 0;
    };
    NodeID[0] |= 0x80;      // set the multicast bit
};
```

Other hash functions, such as SHA-1 [4], can also be used (in which case HASHLEN will be 20). The only requirement is that the result be suitably random – in the sense that the outputs from a set uniformly distributed inputs are themselves uniformly distributed, and that a single bit change in the input can be expected to cause half of the output bits to change.

Obtaining IEEE 802 addresses

The following URL

http://stdsbbs.ieee.org/products/oui/forms/index.html

contains information on how to obtain an IEEE 802 address block. Cost is $1000 US.

Security Considerations

It should not be assumed that UUIDs are hard to guess; they should not be used as capabilities.

Acknowledgements

This document draws heavily on the OSF DCE specification for UUIDs. Ted Ts'o provided helpful comments, especially on the byte ordering section which we mostly plagiarized from a proposed wording he supplied (all errors in that section are our responsibility, however).

References

[1] Lisa Zahn, et. al., Network Computing Architecture, Prentice Hall, Englewood Cliffs, NJ, 1990

[2] DCE: Remote Procedure Call, Open Group CAE Specification C309 ISBN 1-85912-041-5 28cm. 674p. pbk. 1,655g. 8/94

[3] R. Rivest, RFC 1321, "The MD5 Message-Digest Algorithm", 04/16/1992.

[4] SHA Spec - TBD

Authors' addresses

Paul J. Leach
Microsoft
1 Microsoft Way
Redmond, WA, 98052, U.S.A.
Email: paulle@microsoft.co

Rich Salz
The Open Group
11 Cambridge Center
Cambridge, MA 02142, U.S.A.
Email r.salz@opengroup.org

Appendix A – UUID Reference Implementation

```
/*
** Copyright (c) 1990- 1993, 1996 Open Software Foundation, Inc.
** Copyright (c) 1989 by Hewlett-Packard Company, Palo Alto, Ca. &
** Digital Equipment Corporation, Maynard, Mass.
** To anyone who acknowledges that this file is provided "AS IS"
** without any express or implied warranty: permission to use, copy,
** modify, and distribute this file for any purpose is hereby
** granted without fee, provided that the above copyright notices and
** this notice appears in all source code copies, and that none of
** the names of Open Software Foundation, Inc., Hewlett-Packard
** Company, or Digital Equipment Corporation be used in advertising
** or publicity pertaining to distribution of the software without
** specific, written prior permission.  Neither Open Software
** Foundation, Inc., Hewlett-Packard Company, nor Digital Equipment
** Corporation makes any representations about the suitability of
** this software for any purpose.
*/
#include <sys/types.h>
#include <sys/time.h>

typedef unsigned long   unsigned32;
typedef unsigned short  unsigned16;
typedef unsigned char   unsigned8;
typedef unsigned char   byte;

#define CLOCK_SEQ_LAST              0x3FFF
#define RAND_MASK                   CLOCK_SEQ_LAST

typedef struct _uuid_t {
    unsigned32          time_low;
    unsigned16          time_mid;
```

```
    unsigned16          time_hi_and_version;
    unsigned8           clock_seq_hi_and_reserved;
    unsigned8           clock_seq_low;
    byte                node[6];
} uuid_t;

typedef struct _unsigned64_t {
    unsigned32          lo;
    unsigned32          hi;
} unsigned64_t;

/*
**  Add two unsigned 64-bit long integers.
*/
#define ADD_64b_2_64b(A, B, sum) \
    { \
        if (!(((A)->lo & 0x80000000UL) ^ ((B)->lo & 0x80000000UL))) { \
            if (((A)->lo&0x80000000UL)) { \
                (sum)->lo = (A)->lo + (B)->lo; \
                (sum)->hi = (A)->hi + (B)->hi + 1; \
            } \
            else { \
                (sum)->lo  = (A)->lo + (B)->lo; \
                (sum)->hi = (A)->hi + (B)->hi; \
            } \
        } \
        else { \
            (sum)->lo = (A)->lo + (B)->lo; \
            (sum)->hi = (A)->hi + (B)->hi; \
            if (!((sum)->lo&0x80000000UL)) (sum)->hi++; \
        } \
    }

/*
**  Add a 16-bit unsigned integer to a 64-bit unsigned integer.
*/
#define ADD_16b_2_64b(A, B, sum) \
    { \
        (sum)->hi = (B)->hi; \
        if ((B)->lo & 0x80000000UL) { \
            (sum)->lo = (*A) + (B)->lo; \
            if (!((sum)->lo & 0x80000000UL)) (sum)->hi++; \
        } \
        else \
            (sum)->lo = (*A) + (B)->lo; \
    }

/*
**  Global variables.
*/
```

```
static unsigned64_t     time_last;
static unsigned16       clock_seq;

static void
mult32(unsigned32 u, unsigned32 v, unsigned64_t *result)
{
    /* Following the notation in Knuth, Vol. 2. */
    unsigned32 uuid1, uuid2, v1, v2, temp;

    uuid1 = u >> 16;
    uuid2 = u & 0xFFFF;
    v1 = v >> 16;
    v2 = v & 0xFFFF;
    temp = uuid2 * v2;
    result->lo = temp & 0xFFFF;
    temp = uuid1 * v2 + (temp >> 16);
    result->hi = temp >> 16;
    temp = uuid2 * v1 + (temp & 0xFFFF);
    result->lo += (temp & 0xFFFF) << 16;
    result->hi += uuid1 * v1 + (temp >> 16);
}

static void
get_system_time(unsigned64_t *uuid_time)
{
    struct timeval tp;
    unsigned64_t utc, usecs, os_basetime_diff;

    gettimeofday(&tp, (struct timezone *)0);
    mult32((long)tp.tv_sec, 10000000, &utc);
    mult32((long)tp.tv_usec, 10, &usecs);
    ADD_64b_2_64b(&usecs, &utc, &utc);

    /* Offset between UUID formatted times and Unix formatted times.
     * UUID UTC base time is October 15, 1582.
     * Unix base time is January 1, 1970. */
    os_basetime_diff.lo = 0x13814000;
    os_basetime_diff.hi = 0x01B21DD2;
    ADD_64b_2_64b(&utc, &os_basetime_diff, uuid_time);
}

/*
** See "The Multiple Prime Random Number Generator" by Alexander
** Hass pp. 368-381, ACM Transactions on Mathematical Software,
** 12/87.
*/
static unsigned32 rand_m;
static unsigned32 rand_ia;
static unsigned32 rand_ib;
static unsigned32 rand_irand;
```

```
static void
true_random_init(void)
{
    unsigned64_t t;
    unsigned16 seed;

 /* Generating our 'seed' value Start with the current time, but,
  * since the resolution of clocks is system hardware dependent and
  * most likely coarser than our resolution (10 usec) we 'mixup' the
  * bits by xor'ing all the bits together.  This will have the effect
  * of involving all of the bits in the determination of the seed
  * value while remaining system independent.  Then for good measure
  * to ensure a unique seed when there are multiple processes
  * creating UUIDs on a system, we add in the PID.
  */
    rand_m = 971;
    rand_ia = 11113;
    rand_ib = 104322;
    rand_irand = 4181;
    get_system_time(&t);
    seed  =  t.lo        & 0xFFFF;
    seed ^= (t.lo >> 16) & 0xFFFF;
    seed ^=  t.hi        & 0xFFFF;
    seed ^= (t.hi >> 16) & 0xFFFF;
    rand_irand += seed + getpid();
}

static unsigned16
true_random(void)
{
    if ((rand_m += 7) >= 9973)
        rand_m -= 9871;
    if ((rand_ia += 1907) >= 99991)
        rand_ia -= 89989;
    if ((rand_ib += 73939) >= 224729)
        rand_ib -= 96233;
    rand_irand = (rand_irand * rand_m) + rand_ia + rand_ib;
    return (rand_irand >> 16) ^ (rand_irand & RAND_MASK);
}

/*
**  Startup initialization routine for the UUID module.
*/
void
uuid_init(void)
{
    true_random_init();
    get_system_time(&time_last);
#ifdef NONVOLATILE_CLOCK
```

```
    clock_seq = read_clock();
#else
    clock_seq = true_random();
#endif
}

static int
time_cmp(unsigned64_t *time1, unsigned64_t *time2)
{
    if (time1->hi < time2->hi) return -1;
    if (time1->hi > time2->hi) return 1;
    if (time1->lo < time2->lo) return -1;
    if (time1->lo > time2->lo) return 1;
    return 0;
}

static void new_clock_seq(void)
{
    clock_seq = (clock_seq + 1) % (CLOCK_SEQ_LAST + 1);
    if (clock_seq == 0) clock_seq = 1;
#ifdef NONVOLATILE_CLOCK
    write_clock(clock_seq);
#endif
}

void uuid_create(uuid_t *uuid)
{
    static unsigned64_t     time_now;
    static unsigned16       time_adjust;
    byte                    eaddr[6];
    int                     got_no_time = 0;

    get_ieee_node_identifier(&eaddr);      /* TO BE PROVIDED */

    do {
        get_system_time(&time_now);
        switch (time_cmp(&time_now, &time_last)) {
        case -1:
            /* Time went backwards. */
            new_clock_seq();
            time_adjust = 0;
            break;
        case 1:
            time_adjust = 0;
            break;
        default:
            if (time_adjust == 0x7FFF)
                /* We're going too fast for our clock; spin. */
                got_no_time = 1;
            else
```

```
                time_adjust++;
            break;
        }
    } while (got_no_time);

    time_last.lo = time_now.lo;
    time_last.hi = time_now.hi;

    if (time_adjust != 0) {
        ADD_16b_2_64b(&time_adjust, &time_now, &time_now);
    }

    /* Construct a uuid with the information we've gathered
     * plus a few constants. */
    uuid->time_low = time_now.lo;
    uuid->time_mid = time_now.hi & 0x0000FFFF;
    uuid->time_hi_and_version = (time_now.hi & 0x0FFF0000) >> 16;
    uuid->time_hi_and_version |= (1 << 12);
    uuid->clock_seq_low = clock_seq & 0xFF;
    uuid->clock_seq_hi_and_reserved = (clock_seq & 0x3F00) >> 8;
    uuid->clock_seq_hi_and_reserved |= 0x80;
    memcpy(uuid->node, &eaddr, sizeof uuid->node);
}
```

Attachment L: DHCP Options For Host System Characteristics

INTERNET DRAFT

Mike Henry
Eric Dittert
Intel Corp.
March 26, 1997

DHCP Options For Host System Characteristics
<draft-dittert-host-sys-char-01.txt>

Status of this Memo

This document is an Internet Draft. Internet Drafts are working documents of the Internet Engineering Task Force (IETF), its Areas, and its Working Groups. Note that other groups may also distribute working documents as Internet Drafts.

Internet Drafts are draft documents valid for a maximum of six months. Internet Drafts may be updated, replaced, or obsoleted by other documents at any time. It is not appropriate to use Internet Drafts as reference material or to cite them other than as a "working draft" or "work in progress."

Please check the "1id-abstracts.txt" listing contained in the Internet-Drafts Shadow Directories on ftp.is.co.za (Africa), nic.nordu.net (Europe), munnari.oz.au (Pacific Rim), ds.internic.net (US East Coast), or ftp.isi.edu (US West Coast).

Notice
All product and company names mentioned herein might be trademarks of their respective owners.

Abstract

The interoperability of configuration services based on the Dynamic Host Configuration Protocol (DHCP) [1] in an environment of heterogeneous clients depends on clients accurately identifying themselves and their relevant characteristics to configuration servers. The class identifier provided through DHCP option 60 [2] helps in this regard, but such identifiers essentially only enable clients and servers that are "good friends" to find each other. This draft proposes the definition of two options that convey particular, generally useful information about the client system. This enables all servers to recognize this information, and is a step toward a richer form of interoperability for configuration services.

The proposed options are

* Client System Architecture
* Client Network Device Interface

This draft also proposes a new type of client identifier based on generated UUID/GUIDs to be used in conjunction with the DHCP client identifier option (61).

1.0 Introduction

The use of DHCP to provide clients with configuration information in general, and boot images in particular can be complicated by several circumstances. Among these are

1) clients in the same service domain with different system architectures or hardware configurations
2) clients in the same service domain for which different software configurations are desired
3) the desire to have clients and servers provided by different vendors successfully interact

(By "clients in the same service domain" we mean clients, requests from which can reach the same server.) A key element in enabling the successful use of DHCP in such circumstances is the provision of mechanisms by which clients can accurately identify themselves and their relevant characteristics to a server.

For identifying characteristics of the client that are relevant to the selection of a boot image, the currently available mechanisms are the DHCP class identifier option (code 60) and the DHCP vendor specific information option (code 43). By definition, the vendor specific information option does not address the problem of enabling interoperability of clients and servers provided by different vendors. Information conveyed by the class identifier option could enable interoperability, provided that a sufficiently specific and complete set of class identifiers were defined and agreed to.

We suggest using an alternate approach, in which new, specific options are used to convey the characteristics of the client that determine which boot image(s) could run on the client, and the class identifier is used as a (site-specific) designation of the desired software configuration for the client. Section 2 defines two new options that are useful for conveying the client's hardware configuration.

For identifying the client as a unique entity, the currently available mechanisms is the DHCP client identifier option (code 61) [2]. Section 3 of this draft defines for use in this option an identifier type based on generated GUIDs - identifiers that are

guaranteed to be, or are very, very likely to be unique across time and all clients.

2.0 Client Characteristics Options

The options defined in this section provide the server with explicit knowledge about the client system that is generally useful in selecting an executable that the client can use as a boot image.

2.1 Client System Architecture Option

DHCP clients SHOULD include this option in DHCPDISCOVER and DHCPREQUEST messages. Doing so provides the server with explicit knowledge of the client's system architecture.

DHCP servers that use this option SHOULD include the option in responses that contain a bootfile name. If included, the value of the option MUST denote a system architecture for which the bootfile named is valid. DHCP servers MUST NOT include this option in responses that do not contain a bootfile name.

The format for this option is as follows:

```
 Code  Len   System Arch Code
+-----+-----+-----+-----+
| TBD |  2  | s1  | s2  |
+-----+-----+-----+-----+
```

The currently defined types and their codes are

System Architecture	Code
Intel Architecture PC	1
NEC PC-9800	2

2.2 Client Network Device Interface Option

DHCP clients SHOULD include this option in DHCPDISCOVER and DHCPREQUEST messages. Doing so provides the server with explicit knowledge of the client's network device.

DHCP servers that use this option SHOULD include the option in responses that contain a bootfile name. If included, the value of the option MUST denote a network device for which the bootfile named is valid. DHCP servers MUST NOT include this option in responses that do not contain a bootfile name.

Three types of network device specifications are defined for use with this option:

* devices that support the Universal Network Driver Interface (UNDI), as described in the Net PC design guidelines [3]
* Plug-and-Play devices [4]
* PCI devices [5]

Each devices that supports (UNDI) SHOULD be specified as an UNDI device, regardless of whether it is also a Plug-and-Play device or a PCI device. To specify an UNDI device, the option contains a type code of 1 and the major and minor UNDI version numbers:

```
      Code  Len   Type  Major Minor
+-----+-----+-----+-----+-----+
| TBD |  3  |  1  | m1  | m2  |
+-----+-----+-----+-----+-----+
```

To specify a PCI network device, a type code of 2 is used, and the vendor ID, device ID, class code, and revision are included:

```
 Code  Len   Type  Vendor ID   Device ID   Class code        Rev
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| TBD |  9  |  2  | v1  | v2  | d3  | d4  | c1  | c2  | c3  | r1  |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
```

To specify a Plug-and-Play network device, a type code of 3 is used, and the EISA device ID and the class code are included:

```
        Code  Len   Type  EISA device ID          Class code
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| TBD |  8  |  3  | e1  | e2  | e3  | e4  | c1  | c2  | c3  |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
```

3.0 UUID/GUID-based Client Identifiers

Whenever a client identifier option is included in a DHCP message, it MAY contain an identifier in UUID/GUID format. A client identifier option containing a type code of <TBD> MUST contain a 128-bit GUID as follows:

```
         Code   Len    Type   Client GUID
+------+------+------+------+------+------+
|  61  |  17  |  t1  |  g1  |  g2  |  ... |
+------+------+------+------+------+------+
```

The format of the GUID MUST be as specified in the design guidelines for Net PCs [3].

4.0 References

[1] Droms, R. "Dynamic Host Configuration Protocol", RFC 1531

[2] Alexander,S. and Droms, R., "DHCP Options and BOOTP Vendor Extension" RFC 1533.

[3] Design Guidelines for a Net PC, reference to be provided

[4] Plug-and-Play specification, reference to be provided

[5] PCI specification, reference to be provided

5.0 Authors' Addresses

Mike Henry
Intel Corporation, MS JF3-408
5200 NE Elam Young Pkwy
Hillsboro, OR 97124

Phone: (503) 264-9689
Email: Mike_Henry@ccm.jf.intel.com

Eric Dittert
Intel Corporation, MS JF3-206
5200 NE Elam Young Pkwy
Hillsboro, OR 97124

Phone: (503) 264-8461
Email: Eric_Dittert@ccm.jf.intel.com

Hardware Glossary

A

ACPI Advanced Configuration and Power Interface. A specification that defines a new interface to the system board that enables the operating system to implement operating-system directed power management and system configuration. Following the ACPI allows system manufacturers to build systems consistent with the OnNow design initiative for instantly available PCs.

ACPI hardware Computer hardware with the features necessary to support operating system power management and with the interfaces to those features described using the Description Tables as specified in *Advanced Configuration and Power Interface Specification*.

adapter *See* device.

add-on devices Devices that are traditionally added to the base PC system to increase functionality, such as audio, networking, graphics, SCSI controller, and so on. Add-on devices fall into two categories: devices built onto the system board and devices on expansion cards added to the system through a system board connector such as PCI.

agent Software that runs on a client computer for use by administrative software running on a server. Agents are typically used to support administrative actions, such as detecting system information or running services.

algorithm In compression software, refers to a specific formula used to compress or decompress video or other data.

analog A method of signal representation by an infinitely smooth universe of numeric values. Measurements that are characterized as analog include readings of voltage and current. Analog devices are characterized by dials and sliding mechanisms. *Compare with* digital.

ANSI American National Standards Institute. A standards-setting, non-governmental organization that develops and publishes standards for voluntary use in the United States.

API Application programming interface. A set of routines that an applications program uses to request and carry out lower-level services performed by a computer operating system.

APM Advanced Power Management. 1. A software interface (defined by Microsoft and Intel) between hardware-specific power management software (such

as that located in a system BIOS) and an operating system power management driver. 2. An abridgment of the APM BIOS Interface Specification title.

arbitrator 1. A software module in Windows that handles the allocation of hardware resources among devices. 2. Under Windows 95, the Plug and Play device driver responsible for allocating a specific resource among all drivers that require the resource. For example, VDMAD is a DMA-channel arbitrator, and VPICD has services for allocating IRQ lines. Windows 95 provides arbitrators for standard I/O, memory, hardware interrupt, and DMA-channel resources.

architecture A general term referring to the structure of all or part of a computer system. Also covers the design of system software, such as the operating system, as well as referring to the combination of hardware and basic software that links machines on a computer network.

ASCII American Standard Code for Information Interchange. The most popular coding method used by small computers for converting letters, numbers, punctuation, and control codes into digital form.

assigned configuration Drivers examine this portion of the device node to determine which resources have been allocated for the device. For Plug and Play cards, the assigned resources for a device can change dynamically or from one system start-up to the next.

ATA AT Attachment. An integrated bus usually used between host processors and disk drives. Used interchangeably with IDE.

ATAPI AT Attachment Packet Interface. A hardware and software specification that documents the interface between a host computer and CD-ROM drives using the ATA bus.

B

bandwidth Usually used in reference to the amount of data per unit of time that must move from one point to another, such as from CD-ROM to processor.

BIOS Basic input/output system. A set of routines that works closely with the hardware to support the transfer of information between elements of the system, such as memory, disks, and the monitor. Although critical to performance, the BIOS is usually invisible to the end user; however, programmers can access it.

BIOS enumerator Responsible in a non-ACPI Plug and Play system for identifying all hardware devices on the motherboard of the computer. The BIOS supports an API that allows all Plug and Play computers to be queried in a common manner.

bpp Bits per pixel. The number of bits used to represent the color value of each pixel in a digitized image.

buffer A reserved portion of memory in which data is temporarily held pending an opportunity to complete its transfer to or from a storage device or another location in memory.

bus enumerator In a Plug and Play system, a bus device driver that detects devices located on a specific bus and loads information about devices into the hardware tree.

C

cache A special memory subsystem in which frequently used data values are duplicated for quick access. Cache memory is always faster than RAM.

CD-ROM Compact disc read-only memory. A 4.75-inch laser-encoded optical memory storage medium (developed by NV Philips and Sony Corporation) with the same constant linear velocity (CLV) spiral format as compact audio discs and some videodiscs. CD-ROMs can hold about 550 MB of data.

CI Component Instrumentation. A specification for DMI related to the service layer.

CIM Common Information Model. Describes the WBEM data representation schema that is now a DMTF-sponsored industry standard. CIM evolved from HMMS (HyperMedia Management Schema).

CIMOM CIM Object Manager. A key component of the WBEM architecture. A central message of WBEM is uniform data representation encapsulated in object-oriented fashion in the CIM. CIMOM provides a collection point and manipulation point for these objects. *Formerly* HMOM.

class For hardware, the manner in which devices and buses are grouped for purposes of installing and managing device drivers and allocating resources. The hardware tree is organized by device class, and Windows 95 uses class installers to install drivers for all hardware classes.

class driver A driver that provides system-required, hardware-independent support for a given class of physical devices. Such a driver communicates with a corresponding hardware-dependent port driver, using a set of system-defined device control requests, possibly with additional driver-defined device control requests. Under WDM, the class driver creates a device object to represent each adapter registered by minidrivers. The class driver is responsible for multiprocessor and interrupt synchronization.

codec Coder-decoder. A filter for data that manipulates it in some form, usually by compressing or decompressing the data stream.

COM 1) Component Object Model; the core of OLE. Defines how OLE objects and their clients interact within processes or across process boundaries. 2) Legacy serial port.

compatibility mode An asynchronous, host-to-peripheral parallel port channel defined in the IEEE 1284–1944 standard. Compatible with existing peripherals that attach to the Centronics-style PC parallel port.

compression The translation of data (video, audio, digital, or a combination) to a more compact form for storage or transmission.

Configuration Manager The Windows 95 Plug and Play system component that drives the process of locating devices, setting up their nodes in the hardware tree, and running the resource allocation process. Each of the three phases of configuration management—boot time (BIOS), real mode, and protected mode—have their own configuration managers.

connection A negotiated method of communication between devices, whether implemented in hardware or software.

CPU Central processing unit. The computational and control unit of a computer; the device that interprets and executes instructions. By definition, the CPU is the chip that functions as the “brain” of the computer.

D

data rate The speed of a data transfer process, normally expressed in bits per second or bytes per second.

DDC Display data channel. The Plug and Play baseline for monitors. The communications channel between a monitor and the display adapter to which it is connected. This channel provides a method for the monitor to convey its identity to the display adapter.

DDI Device driver interface.

DDK Device driver kit.

density The degree of darkness of an image. Also, the percent of screen used in an image.

device 1. Any circuit that performs a specific function, such as a parallel port. 2. For WDM, usually refers to a device object, but also refers to a unit of hardware—for example, an audio adapter—that is detected by Plug and Play.

Device Bay An industry specification that defines a mechanism for both peripheral devices and system bays that allows adding and upgrading PC peripheral devices without opening the chassis.

device ID A unique ASCII string created by enumerators to identify a hardware device and used to cross-reference data about the device stored in the registry. Distinguishes each logical device and bus from all others on the system.

device node The basic data structure for a given device, built by the Configuration Manager. Device nodes are built into memory at system startup for each device and enumerator. Each device node contains information about the device, such as currently assigned resources. The complete hierarchical representation of all device nodes, representing all currently installed devices, is referred to as the hardware tree.

devnode *See* device node.

DIB Device-independent bitmap. A file format designed to ensure that bitmap graphics created using one application can be loaded and displayed in another application exactly the way they appeared in the originating application.

digital A method of signal representation by a set of discrete numerical values, as opposed to a continuously fluctuating current or voltage. *Compare with* analog.

disk I/O controller *Also* HDC. A special-purpose chip and circuitry that directs and controls reading from and writing to a computer's disk drive.

DLL Dynamic link library. API routine that User-mode applications access through ordinary procedure calls. The code for the API routine is not included in the user's executable image. Instead, the operating system automatically modifies the executable image to point to DLL procedures at run time.

DMA Direct memory access. A method of moving data from a device to memory (or vice versa) without the help of the microprocessor. The system board uses a DMA controller to handle a fixed number of channels, each of which can be used by only one device at a time.

DMI Desktop Management Interface. A framework created by the DMTF. DMTF specifications define industry-standard interfaces for instrumentation providers and management applications.

DMTF Desktop Management Task Force.

dock To insert a portable computer into a base unit. Cold docking means the computer must begin from a power-off state and restart before docking. Hot docking means the computer can be docked while running at full power. *See also* warm docking.

docking station The base computer unit into which a user can insert a portable computer, expanding it to a desktop equivalent. A typical docking station provides drive bays, expansion slots, all the ports on an equivalent desktop computer, and AC power.

dongle A physical device, attached to a PC's I/O port, that adds hardware capabilities.

driver Kernel-mode code used either to control or emulate a hardware device.

DSP Digital signal processor. An integrated circuit designed for high-speed data manipulations. Used in audio, communications, image manipulation, and other data-acquisition and data-control applications.

DVD Optical disk storage that encompasses audio, video, and computer data.
dynamic detection The process by which a system can detect that a new device has been added or removed from the PC. This process allows the operating system and applications to immediately begin using the added devices or stop using the removed devices without rebooting the system.

E

ECP Extended capabilities port. An asynchronous, 8-bit–wide parallel channel defined by IEEE 1284–1944 that provides PC-to-peripheral and peripheral-to-PC data transfers.

EISA Extended Industry Standard Architecture. A 32-bit PC expansion bus designed as a superset of the ISA bus. Designed to expand the speed and data width of the legacy expansion bus while still supporting older ISA cards.

embedded controller The general class of microcontrollers used to support OEM-specific implementations, mainly in mobile environments. The embedded controller performs complex low-level functions through a simple interface to the host microprocessor(s).

enumeration The process by which logical devices and buses, and their available resources, are identified by Plug and Play during system setup.

enumerator A Plug and Play device driver that detects devices below its own device node, creates unique device IDs, and reports to Configuration Manager during startup. For example, a SCSI adapter provides a SCSI enumerator that detects devices on the SCSI bus.

expansion bus A group of address, data, and control lines that provide a buffered interface to devices located either on the system board or on cards that are plugged into expansion connectors. Common expansion buses included on the system board are USB, PC Card, and PCI.

expansion card A card that connects to an expansion bus and contains one or more devices.

expansion ROM *See* option ROM.

F

FAT File allocation table. A table or list maintained by an operating system to keep track of the status of various segments of disk space used for file storage.

FDC Floppy disk drive controller. A special-purpose chip and associated circuitry that directs and controls reading from and writing to a computer's disk drive.

FIFO First in/first out. A method for processing a queue in which items are removed in the same order they were added.

full duplex In terms of data flow, indicates bidirectional data flow.

G

GB Gigabyte.

GUID Globally unique identifier. A 16-byte value generated from the unique identifier on a adapter, the current date and time, and a sequence number. This is used to allow any party to create identifiers that will be guaranteed not to overlap with other similarly created identifiers.

H

hardware branch The hardware archive root key in the registry that is a superset of the memory-resident hardware tree. Although the hardware tree contains information only about those devices currently detected and running in the system, the registry contains a complete list of all hardware ever installed on the particular computer. The hardware root key is \\Hkey_Local_Machine\Hardware.

hardware tree A record in RAM of the current system configuration based on the configuration information for all devices in the hardware branch of the registry. The hardware tree is created each time the system is started or whenever a dynamic change occurs to the system configuration.

HCI Host controller interface. System-level interface supporting USB.

HCL Hardware Compatibility List. *See* WHQL.

HCT Hardware Compatibility Tests. A suite of tests from WHQL to verify hardware and device driver operations under a specific operating environment. These tests exercise the combination of a device, a software driver, and an

operating system under controlled conditions to verify that all components operate properly.

HDC Hard disk I/O controller.

HID Human Interface Device.

high resolution An adjective describing improvement in display image quality as a result of increasing the number of pixels per square inch.

HMMP HyperMedia Management Protocol. The encapsulation of CIM objects and operations on those objects.

HMMS *See* CIM.

HMOM *See* CIMOM.

Human Interface Device Specification The device class definition developed by the USB standards group for HIDs. Serves as the basis for the WDM input device support, and unifies input devices by providing flexible data reporting, typeless data, and arrayed and variable input and output.

Hz Hertz (cycles per second).

I

I2O Intelligent I/O.

IA-PC Intel Architecture Personal Computer. A general descriptive term for computers built with processors conforming to the architecture defined by the Intel processor family based on the 486 instruction set.

IDE Integrated Device Electronics. A type of disk-drive interface where the controller electronics reside on the drive itself, eliminating the need for a separate adapter card.

IEEE Institute of Electrical and Electronics Engineers. Organization that developed the IEEE 802 standards, among others, for the physical and data-link layers of LANs following the ISO/OSI model.

IETF Internet Engineering Task Force.

IHV Independent hardware vendor.

image resolution The fineness or coarseness of an image as it is digitized; measured in dots per inch (DPI), typically ranging from 200 to 400 DPI.

INF file Information file. A file created for a particular adapter that provides the operating system with information required to set up a device, such as a list of valid logical configurations for the device, the names of driver files associated with the device, and so on. An INF file is typically provided by the device manufacturer on a disk with an adapter.

INI file Initialization file. Commonly used under Windows 3.x and earlier, INI files have been used by both the operating system and individual applications to store persistent settings related to an application, driver, or piece of hardware. In Windows NT and Windows 95, INI files are supported for backward compatibility, but the registry is the preferred location for storing such settings.

instrumentation A mechanism for reporting information about the state of PC hardware and software to enable management applications to ascertain and change the state of a PC and to be notified of state changes.

integrated device Any device—such as a parallel port, graphics adapter, and so on—that is designed on the system board rather than on an expansion card.

interface For parameters on a connection request, a specific set of methods and properties implemented on a medium that a filter connection uses to communicate, such as a specific set of IOCTLs.

I/O Input/output. Two of the three activities that characterize a computer (input, processing, and output). Refers to the complementary tasks of gathering data for the microprocessor to work with and make the results available to the user through a device such as the display, disk drive, or printer.

IOCTL Input/output control. A custom class of IRPs available to User mode. Each WDM class driver has a set of IOCTLs that it uses to communicate with applications. The IOCTLs give the class driver information about intended usage by applications. The class driver performs all IOCTL parameter validation.

IPL Initial program load. A device used by the system during the boot process to load an operating system into memory.

IrDA Infrared Data Association.

IRP I/O request packet. Data structures that drivers use to communicate with each other. The basic method of communication between kernel-mode devices. An IRP is a key data structure for WDM, which features multiple layered drivers. In WDM, every I/O request is represented by an IRP that is passed from one driver layer to another until the request is complete. When a driver receives an IRP, it performs the operation the IRP specifies, and then either passes the IRP back to the I/O Manager for disposal or onto an adjacent driver layer. An IRP packet consists of two parts: a header and the I/O stack locations.

IRQ Interrupt request. A method by which a device can request to be serviced by the device's software driver. The system board uses a PIC to monitor the priority of the requests from all devices. When a request occurs, the microprocessor suspends the current operation and gives control to the device driver associated with the interrupt number issued. The lower the number—for example, IRQ3—the higher the priority of the interrupt. Many devices only support raising requests of specific numbers.

ISA Industry Standard Architecture. An 8-bit (and later, a 16-bit) expansion bus that provides a buffered interface from devices on expansion cards to the PC internal bus.

ISDN Integrated Service Digital Network. A set of communications standards that enable a single phone line or optical cable to carry voice, digital network services, and video.

ISO International Standards Organization.

isochronous Refers to a communication protocol based on time slices rather than handshaking. For example, a process might have 20 percent of total bus bandwidth. During its time slice, the process can stream data.

isolation The Plug and Play process by which cards on an ISA bus are distinguished from each other after system startup.

ISO/OSI International Standards Organization Open Systems Interconnection model. A layered architecture that standardizes levels of service and types of interaction for computers exchanging information through a communications network.

ISR Interrupt service routine. A routine whose function is to service a device when it generates an interrupt.

K

K Kilobyte.

kernel The core of the layered architecture that manages the most basic operations of the operating system, such as sharing the processor between different blocks of executing code, handling hardware exceptions, and other hardware-dependent functions.

kernel mode The processor mode that allows full, unprotected access to the system. A driver or thread running in kernel mode has access to system memory and hardware.

kernel-mode driver Driver for logical, virtual, or physical devices. Part of the underlying operating system that supports ring 0 operations.

L

LAN Local area network. A group of computers and other devices dispersed over a relatively limited area and connected by a communications link that enables any device to interact with any other device on the network. *Compare with* WAN.

legacy Any feature in the PC system based on older technology for which compatibility continues to be maintained in other system components.

local bus Usually refers to a system bus directly connected to the microprocessor on a system board. Used colloquially to refer to system board buses located closer to the microprocessor than are ordinary expansion buses (that is, with less buffering), which are therefore capable of greater throughput.

M

MB Megabyte.

MCD Miniclient Driver. An OpenGL driver model in which the driver is responsible only for handling those features that can be accelerated in hardware, leveraging software implementation to handle the rest of the pipeline.

MIDI Musical Instrument Digital Interface. An industry-standard connection for computer control of musical instruments and devices. A hardware and data standard for communicating between hardware. Most references involve only the data standard, which is a byte stream used for controlling musical instruments and storing the output of such instruments.

minidriver A hardware-specific DLL that uses a operating system-provided class driver to accomplish most actions through functions call and provides only device-specific controls. Under WDM, the minidriver registers each adapter with the class driver, which creates the device object. The minidriver uses the class driver's device object to make system calls.

miniport driver A device-specific kernel-mode driver linked to a Windows NT or WDM port driver, usually implemented as a DLL that provides an interface between the port driver and the system.

MIPS Millions of instructions per second. A common measure of processor speed.

MIS Management information system.

monolithic driver A driver that has many different classes of functionality contained in the same driver.

motherboard *See* system board.

MPEG Motion Picture Experts Group. Used when referring to one of several standard video-compression schemes. A codec for squeezing full-screen, VHS-quality digital video into a small data stream so it can be played from a CD-ROM drive.

multifunction device A piece of hardware that supports multiple, discrete functions, such as audio, mixer, and music, on a single adapter.

multimedia Refers to the delivery of information that combines different content formats (motion video, audio, still images, graphics, animation, text, and so forth).

N

NDIS Network Driver Interface Specification. The interface for network drivers used in Windows and Windows NT. NDIS provides transport independence for network vendors because all transport drivers call the NDIS interface to access the network.

Net PC Network PC. A PC designed to meet the industry specification for Network PC systems, which optimizes PC design for flexibility and manageability in order to reduce the total cost of ownership.

nibble mode An asynchronous, peripheral-to-host channel defined in the IEEE 1284-1944 standard. Provides a channel for the peripheral to send data to the host, which is commonly used as a means of identifying the peripheral.

NMI Nonmaskable Interrupt. An interrupt that cannot be overruled by another service request. A hardware interrupt is called nonmaskable if it bypasses and takes priority over interrupt requests generated by software, the keyboard, and other devices.

non-interlaced The method of scanning all lines on a display from top to bottom in sequential order at a specific rate per second. Unlike television, which uses an interlaced scanning method, computers typically use non-interlaced monitors.

NSP Native signal processing.

NTFS Windows NT file system. An advanced file system designed for use specifically with the Windows NT operating system. NTFS supports file system recovery and extremely large storage media, in addition to other advantages.

O

OEM Original equipment manufacturer. Used primarily to refer to PC systems manufacturers.

OLE Object linking and embedding. A way to transfer and share information among applications. OLE is based on the COM programming model and binary standard.

OnNow A design initiative that seeks to create all the components required for a comprehensive, system-wide approach to system and device power control. OnNow is a term for a PC that is always on but appears off and that responds immediately to user or other requests.

option ROM Optional read-only memory found on PC bus expansion cards. Option ROMs usually contain additional firmware required to properly boot the peripheral connected to the expansion card, for example, a hard drive. *Also* expansion ROM.

P

PC 97 The 1997–98 requirements for PC system and peripheral design defined in *PC 97 Hardware Design Guide* (Microsoft Press, 1996).

PC Card A trademark of PCMCIA. A removable device that is designed to be plugged into a PCMCIA slot and used as a memory-related peripheral.

PCI Peripheral Component Interconnect. A high-performance, 32-bit or 64-bit bus designed to be used with devices that have high bandwidth requirements, such as the display subsystem.

PCMCIA Personal Computer Memory Card International Association. Sometimes used to refer to a controller for a type of expansion card documented in the PCMCIA standards.

PIC Programmable interrupt controller

pixels An abbreviation for picture element. The minimum raster display element, represented as a point, with a specified color or intensity level. One way to measure picture resolution is by the number of pixels used to create images.

planar *See* system board.

Plug and Play (PnP) A design philosophy and set of specifications that describe hardware and software changes to the PC and its peripherals that automatically identify and arbitrate resource requirements among all devices and buses on the system. Plug and Play specifies a set of API elements that are used in addition to, not in place of, existing driver architectures.

Plug and Play BIOS A BIOS with responsibility for configuring Plug and Play cards and system-board devices during system power up. Provides runtime configuration services for system board devices after startup. *See also* ACPI.

port A connection or socket used to connect a device—such as a printer, monitor, or modem—to the computer. Information is sent from the computer to the device through a cable.

port driver A low-level driver that responds to a set of system-defined device control requests and possibly to an additional set of driver-defined (private) device control requests sent down by a corresponding class driver. A port driver insulates class drivers from the specifics of host bus adapters and synchronizes operations for all its class drivers.

POST Power-on self-test. A procedure of the system BIOS that identifies, tests, and configures the PC in preparation for loading the operating system.

power management Mechanisms in software and hardware to minimize system power consumption, manage system thermal limits, and maximize system battery life. Power management involves trade-offs among system speed, noise, battery life, processing speed, and AC power consumption. Power management is required for some system functions, such as appliance operations (including answering machine, furnace control, and so on).

power policy For power management, the decisions that determine how to save energy and when to go to sleep, based on end-user preferences, application needs, and system hardware capabilities.

power resources Resources such as power planes, clock sources, and so on that a device requires to operate in a given power state.

power sources The battery and AC adapter that supply power to a platform.

property In WDM and Windows NT device driver models, an aspect of the device or stream that can be set or retrieved, such as volume level.

property set In WDM and Windows NT device driver models, a method defined to set and get properties on a driver. Each property set has a unique identifier, which represents types of related information and is used to access the property set.

R

RAM Random access memory**.** Semiconductor-based memory that can be read and written by the microprocessor or other hardware devices. Refers to volatile memory, which can be written as well as read.

RAMDAC RAM digital-to-analog converter. A chip built into some VGA and SVGA display adapters that translates the digital representation of a pixel into the analog information needed by the monitor to display it. The presence of a RAMDAC chip usually enhances overall display performance.

raster A rectangular pattern of lines.

raster graphics *Also* bitmapped graphics. Images defined as a set of pixels or dots in a column-and-row format.

rasterization The conversion of vector graphics (images described mathematically as points connected by straight lines) to equivalent images composed of pixel patterns that can be stored and manipulated as sets of bits.

real time In computing, refers to an operating mode under which data is received and processed; the results are returned instantaneously.

real-time processing Processing that supports real-time functions such as telephony.

Redbook audio The data format standard for conventional audio CDs used in home stereo systems.

registry In Windows and Windows NT, the tree-structured hierarchical database where general system hardware and software settings are stored. The registry supersedes the use of separate INI files for all system components and applications that know how to store values in the registry.

resolution Number of pixels per unit of area. A display with a finer grid contains more pixels and thus has a higher resolution and is capable of reproducing greater detail in an image.

resource 1. Any sort of set from which a subset can be allocated for use by a client, such as memory, bus bandwidth, or MIPS. This is not the same as resources that are allocated by Plug and Play. 2. A general term that refers to IRQ signals, DMA channels, I/O port addresses, and memory addresses for Plug and Play.

resource arbitrator In Plug and Play device configuration, a set of functions used by the configuration manager to arbitrate and allocate resources on the PC.

resource conflict In Plug and Play device configuration, the result of more than one device sharing the same, nonshareable resource. Conflicts can cause the device to be partially functional or nonfunctional, or can cause the PC to malfunction completely.

resource data type function A function that describes the resource requirements of an ISA expansion card as well as the programmability available on the card and its interdependencies.

RISC Reduced instruction set computing. A type of microprocessor design that focuses on rapid and efficient processing of a relatively small set of instructions. RISC architecture limits the number of instructions that are built into the microprocessor, but optimizes each so it can be carried out very rapidly—usually within a single clock cycle.

RISC-based Refers to computers based on Windows NT–compatible implementations of RISC processors.

RLE Run-length encoding. A data-compression technique in which successive bytes of identical data are converted into a 2-byte pair, consisting of the repeated data byte and the repeat count.

S

scalability The ability to vary the information content of a program by changing the amount of data that is stored, transmitted, or displayed. In a video image, this translates to creating larger or smaller windows of video on screen (shrinking effect).

scaling Process of uniformly changing the size of characters or graphics.

SCAM SCSI Configured Automatically.

SCI System control interrupt. A system interrupt used by hardware to notify the operating system of ACPI events. The SCI is an active low, shareable, level interrupt.

SCSI Small computer system interface. Pronounced “scuzzy.” An I/O bus designed as a method for connecting several classes of peripherals to a host system without requiring modifications to generic hardware and software.

SDK Software development kit.

Smart Battery subsystem A battery subsystem that conforms to the ACPI requirements and implementation defined in *Smart Battery Charger Specification* and related specifications.

SMI System management interrupt. An operating-system–transparent interrupt generated by interrupt events on legacy systems. By contrast, on ACPI systems, interrupt events generate an operating-system–visible interrupt that is shareable (edge-style interrupts will not work).

SMS 1) Microsoft Systems Management Server. Provides a centralized management service for distributed systems. 2) Short messaging service.

socket services In Windows, a protected-mode VxD that manages PC Card hardware. Provides a protected-mode PCMCIA Socket Services 2.x interface for use by Card Services. A socket-services driver must be implemented for each separate PC Card controller that is used.

software device A filter in kernel streaming and DirectShow (formerly ActiveMovie) that has no underlying hardware associated with it.

Sound Blaster Hardware produced by Creative Labs, Inc., that represents for MS-DOS–based games one of the major hardware interfaces for both audio and music (specifically MIDI) data.

SPI Service Provider Interface.

spin down A power-management capability in which a hard drive shuts down its spindle motor.

static resources Device resources, such as IRQ signals, DMA channels, I/O port addresses, and memory addresses, that cannot be configured or relocated.

SVD Simultaneous voice/data. A technology used in TAPI-based modem technology.

system board *Also* motherboard *or* planar. The primary circuit board in a PC that contains most of the basic components of the system.

system context The volatile data in the system that is not saved by a device driver.

system devices Devices on the system board, such as interrupt controllers, keyboard controller, real-time clock, DMA page registers, DMA controllers, memory controllers, FDC, IDE ports, serial and parallel ports, PCI bridges, and so on. In today's PCs, these devices are typically integrated in the supporting chip set.

T

TAPI Telephony Application Program Interface. A set of Win32-based calls that applications use to control modems and telephones by routing application function calls to the appropriate service provider DLL for a modem.

telephony Telephone technology. The conversion of sound into electrical signals, its transmission to another location, and its reconversion to sound, with or without the use of connecting wires.

U

UART Universal Asynchronous Receiver/Transmitter. A module composed of a single integrated circuit, which contains both the receiving and transmitting circuits required for asynchronous serial communication.

UNC Universal naming convention.

Unimodem Universal modem driver. A driver-level component that uses modem description files to control its interaction with the communications driver, VCOMM.

UPS Uninterruptible power supply. A device connected between a computer and a power source that ensures that electrical flow to the computer is not interrupted because of a blackout and, in most cases, protects the computer against potentially damaging events such as power surges and brownouts.

URB USB request block. Clients send URB transfers to the bus by including a pointer in an IRP to a URB structure; a function within the URB identifies the specific request.

USB Universal Serial Bus. A bidirectional, isochronous, dynamically attachable serial interface for adding peripheral devices such as game controllers, serial and parallel ports, and input devices on a single bus.

USB class The class of filters under WDM that provides a bus interface and bus enumerator for USB.

User mode For Windows and Windows NT, the nonprivileged processor mode in which application code executes, including protected subsystem code in Windows NT.

User-mode drivers Win32-based multimedia drivers and VDDs for MS-DOS–based applications with application-dedicated devices. For more information, see the *Multimedia Drivers* and *Virtual DOS Drivers* documentation in the Windows NT DDK.

V

VAR Value added reseller. A company that resells hardware and software packages to developers and/or end users.

VBI Vertical blanking interval. The time interval between television fields needed for the scanning gun to move from the bottom to the top of the screen for the start of the next field.

VCACHE In Windows, a 32-bit protected-mode cache driver.

VCOMM In Windows, a 32-bit protected-mode communications driver.

VCR Video cassette recorder. An analog magnetic recording and playback machine. Usually used for recording and viewing full-motion video; also useful as a data backup device.

VDD Virtual display driver.

VESA Video Electronics Standards Association. The governing body that establishes standards for the video and graphics portions of the electronics industry.

VGA Video graphics array. A video adapter that supports 640x480-pixels color resolution. Video display standard for boot devices under Windows operating systems. Provides medium-resolution text and graphics.

VM Virtual machine. Software that mimics the performance of a hardware device. For example, a software program that allows applications written for an Intel processor to be run on a Motorola chip interprets the Intel machine instructions, becoming a virtual Intel machine.

VxD Virtual Device Driver. A device driver that runs at the privileged ring 0 protected mode of the microprocessor. Can extend the services of the Windows kernel, supervise hardware operations, or perform both functions. Such driver files are usually named according to the scheme VxD, where x refers to the device or service supported.

W

WAN Wide area network. A communications network that connects geographically separated areas. *Compare with* LAN.

warm docking A method of removing or installing a mobile system in a docking station with which the computer can be docked or undocked while in a reduced power state, such as suspend.

WBEM Web-based Enterprise Management. Technology under development by BMC Software, Inc., Cisco Systems, Inc., Compaq Computer Corporation, Intel Corporation, and Microsoft Corporation, based on standards being developed by DMTF and IETF, to provide a mechanism to specify information exchange between management applications and managed components.

WDL Windows Driver Library. *See* WHQL.

WDM Win32 Driver Model. A 32-bit driver model based on the Windows NT driver model that is designed to provide a common architecture of I/O services and binary-compatible device drivers for both Windows NT and Windows operating systems for specific classes of drivers. These driver classes include USB and IEEE 1394 buses, audio, still-image capture, video capture, and HID-compliant devices such as USB mice, keyboards, and joysticks. Provides a model for writing kernel-mode drivers and minidrivers, and provides extensions for Plug and Play and power management.

WDM DDK Provides the supplementary header files used together with the Windows NT DDK to build WDM drivers.

WDM power management Facilities provided in WDM for drivers to implement power policy and control. DDIs are defined for synchronizing power state changes with other power management activities in the system and for detecting device

idleness. IRPs are defined for setting power sate, enabling wakeup, and querying power status.

WfM Wired for Management initiative, aimed at increasing the manageability of desktop PCs and servers and improving the management software for these systems.

WHQL Windows Hardware Quality Labs. Formerly Microsoft Compatibility Labs. Provides compatibility testing services to test hardware and drivers for Windows and Windows NT. Administers testing for the "Designed for Microsoft Windows" logo programs. Information: http://www.microsoft.com/hwtest/.

Win32 API A 32-bit application programming interface for both Windows and Windows NT that includes sophisticated operating-system capabilities, security, and API routines for Windows-based applications.

Windows Refers to the Microsoft Windows 95 operating system, including any add-on capabilities and any later versions of the operating system.

Windows NT Refers to the Microsoft Windows NT version 4.0 operating system, including any add-on capabilities and any later versions of the operating system, unless specific design issues are defined that relate to version 5.0.

Windows NT DDK Supports Windows NT, provided through MSDN Professional membership. Documents the Windows NT driver model (upon which WDM is based) and is an essential component for building WDM drivers.

Windows NT driver model The layered device driver model used under the Windows NT operating system. For information, see *Inside Windows NT*, by Helen Custer (Microsoft Press, 1993).

WMI Windows Management Instrumentation. Extensions to WDM being developed for Windows NT 5.0 and Windows 98 to provide an operating system interface through which instrumented components can provide information and notifications.

workstation In general, a powerful computer with considerable calculating and graphics capability.

Z

ZAK Zero Administration Kit for Windows NT Workstation. A set of tools, methodologies, and guidelines for IT Managers and OEMs that incorporates and supplements existing Windows NT technologies to allow for simplified implementation of a secure, policy-based management.

Zero Administration Windows A Microsoft initiative that focuses on improving Windows and Windows NT for maximum automation of administrative tasks with centralized control and maximum flexibility.