NOT MEASUREMENT SENSITIVE

MIL-HDBK-284-2
22 JULY 1992

MILITARY HANDBOOK

INTERACTIVE COURSEWARE (ICW) FOR MILITARY TRAINING, PORTABILITY PRACTICES FOR

(PART 2 OF 3 PARTS)

AMSC N/A

AREA ILSS

DISTRIBUTION STATEMENT A. Approved for public release; distribution is unlimited.

FOREWORD

1. This military handbook is approved for use by all Departments and Agencies of the Department of Defense.

2. Beneficial comments, recommendations, additions, deletions, and any pertinent data which may be of use in improving this document should be addressed to: Commander, Naval Sea Systems Command, SEA 55Z3, Department of the Navy, Washington, DC 20362-5101 by using the Standardization Document Improvement Proposal (DD Form 1426) appearing at the end of this document or by letter.

3. Guidance provided in this document is not intended to supplement or duplicate policies and procedures in existing Federal, Department of Defense (DoD), and Military Service regulations. Should conflict arise between this handbook and any of the previously mentioned regulations, the regulations take precedence.

4. This handbook on Interactive Courseware (ICW) portability practices was developed within the DoD with the assistance of the Interactive Multimedia Association (IMA), Compatibility Committee. It is designed to help ICW and ICW device manufacturers, developers, and users implement the mandatory software interface and command requirements for ICW and authoring systems prescribed by MIL-STD-1379, Appendix D. Information and guidance is provided for personnel responsible to define operational training requirements; and for ICW development, acquisition, implementation, and life cycle support.

5. The IMA Recommended Practices adopted by the DoD state that Virtual Device Interface (VDI) Management implementations can support Color Graphics Adapter (CGA-) or Enhanced Graphics Adapter (EGA-) based systems only and be compliant. The DoD Instruction 1322.20, Enclosure 4 "Interim Standards and Practices for Interactive Courseware" further requires that courseware and hardware be compatible with the standard VGA, CGA, and EGA graphics at the Read Only Memory Basic Input-Output System (ROM-BIOS) level. However, it is highly probable that future Government purchases of ICW and ICW Training System (ICWTS) will specify compatibility with the VGA graphics adapter. Therefore, VDI Management developers should develop implementations that support VGA graphics for maximum portability. (See 4.3.2.1)

6. This handbook is dynamic in that it supports implementation of evolving software interface and command requirements for ICW and authoring systems. The IMA is developing additional multimedia portability practices to support digital audio and audio management service groups: additional service groups may also be required to support digital video or other emerging technologies. Appendix D of MIL-STD-1379 and this handbook will be updated to incorporate these new industry standards once they are defined and approved for DoD implementation.

CONTENTS

CONTENTS

CONTENTS

CONTENTS

APPENDIX A

SYSTEM (sy) COMMANDS FOR ICW PORTABILITY

CONTENTS

APPENDIX A - Continued.

CONTENTS

APPENDIX A - Continued.

CONTENTS

APPENDIX B

VISUAL-MANAGEMENT (vm) COMMANDS FOR ICW PORTABILITY

CONTENTS

APPENDIX B - Continued

CONTENTS

APPENDIX B - Continued

CONTENTS

APPENDIX B - Continued

CONTENTS

APPENDIX B - Continued

APPENDIX C

VIDEODISC (vd) COMMANDS FOR ICW PORTABILITY

CONTENTS

APPENDIX C - Continued

CONTENTS

APPENDIX C - Continued

CONTENTS

APPENDIX C - Continued

CONTENTS

APPENDIX C - Continued

CONTENTS

APPENDIX D

XY-INPUT (xy) COMMANDS FOR ICW PORTABILITY

CONTENTS

APPENDIX D - Continued

CONTENTS

APPENDIX D - Continued

APPENDIX E

DIGITAL AUDIO (da) COMMANDS FOR ICW PORTABILITY

APPENDIX F

AUDIO MANAGEMENT (am) COMMANDS FOR ICW PORTABILITY

APPENDIX G

DEFAULT POSITIONS OF ICW GRAPHICS

CONTENTS

APPENDIX G - Continued

CONTENTS

APPENDIX G - Continued

APPENDIX H

ICW PORTABILITY PRACTICES ERROR HANDLING

CONTENTS

APPENDIX H - Continued

CONTENTS

APPENDIX H - Continued

CONTENTS

APPENDIX H - Continued

APPENDIX I

APPLICATION PROGRAMMING EXAMPLES

CONTENTS

APPENDIX J

ICW PORTABILITY PRACTICES HANDBOOK CROSS-REFERENCES

CONTENTS

CONTENTS

1. SCOPE

1.1 Scope. This Handbook provides guidance on implementing the software interface and command requirements established by DoD Instruction 1322.20 and MIL-STD-1379, Appendix D, to ensure interactive courseware (ICW) and authoring system portability. The requirements are implemented using the portability practices recommended in this handbook.

1.2 Application guidance. This handbook defines a standard set of software interface and related command protocol implementation practices. These implementation practices are intended for ICW manufacturers, systems integrators, authoring system developers, and software developers who wish to comply with the Interactive Multimedia Association (IMA) Recommended Practices adopted by the DoD.

1.2.1 Applicability. The software interface and command requirements prescribed by Appendix D of MIL-STD-1379, and the ICW Portability Practices defined in this Handbook apply to ICW applications and authoring systems designed for delivery systems using Microsoft Disk Operating System (MS-DOS) version 2.0 or higher, or its functional equivalent; and the Intel 80X86 family of microprocessors or its functional equivalent. (See 4.3.2.1) These portability practices do not apply to ICW training systems that use exempted operating systems or microprocessor architectures.

1.2.2 How to use this handbook. This handbook provides information and guidance on DoD ICW portability initiatives and implementation of the Interactive Multimedia Association (IMA) Recommended Practices for Multimedia Portability (MS-DOS Based Systems), Release R 1.1 (hereinafter referred to as the IMA Recommended Practices). The IMA Recommended Practices address application program protocols adopted by the DoD. The information in Section 4, General Guidance, provides an introduction to the overall portability initiatives and provides general information pertaining to the IMA Recommended Practices. The Detailed Guidance, Section 5, presents software interface and related command implementation. The handbook appendixes contain information and guidance that supplements Sections 4 and 5.

1.2.2.1 Appendix A, System (sy) Commands for ICW Portability. This appendix provides specific guidance on implementing the system (sy) service group established by IMA, and required by MIL-STD-1379, Appendix D. These commands relate to overall VDI software operation.

1.2.2.2 Appendix B, Visual-Management (vm) Commands for ICW Portability. This appendix provides specific guidance on implementing the visual-management (vm) service group established by IMA, and required by MIL-STD-1379, Appendix D. These commands relate to the visual management of the display screen.

1.2.2.3 Appendix C, Videodisc (vd) Commands for ICW Portability. This appendix provides specific guidance on implementing the videodisc (vd) service group established by

IMA, and required by MIL-STD-1379, Appendix D. These commands control videodisc players.

1.2.2.4 Appendix D, X-Y-Input (xy) Commands for ICW Portability. This appendix provides specific guidance on implementing the XY-input (xy) service group established by IMA, and required by MIL-STD-1379, Appendix D. These commands relate to X-Y input devices such as mice, touchscreens, and light pens.

1.2.2.5 Appendix E, Digital Audio (da) Commands for ICW Portability. This appendix is reserved to support the implementation of the digital audio (da) service group requirements once they are developed and defined in MIL-STD-1379, Appendix D. These commands control digital audio devices.

1.2.2.6 Appendix F, Audio Management (am) Commands for ICW Portability. This appendix is reserved to support the implementation of the audio management (am) service group requirements once they are developed and defined in MIL-STD-1379, Appendix D. These commands relate to system audio management functions.

1.2.2.7 Appendix G, Default Positions of ICW Graphics. This appendix presents a methodology for determining the default position of application program graphics in relation to video.

1.2.2.8 Appendix H, ICW Portability Practices Error Handling. This appendix describes system error codes and presents a listing of the approved codes and their descriptions.

1.2.2.9 Appendix I, Application Programming Examples. This appendix provides examples of VDI management implementation programming.

1.2.2.10 Appendix J, ICW Portability Practices Handbook Cross-References. This appendix is a cross-reference of handbook paragraphs to related paragraphs in MIL-STD-1379, Appendix D, and the IMA Recommended Practices document.

1.3 Parts. MIL-HDBK-284-2 is Part 2 of three parts. Part 1, MIL-HDBK-284-1, Interactive Courseware (ICW) for Military Training, Manager's Guide for Development, Acquisition, and Management of, is used in conjunction with MIL-STD-1379 and this handbook to define and implement standard ICW portability protocol requirements. Part 3, MIL-HDBK-284-3, Interactive Courseware (ICW) for Military Training, Glossary for, contains definitions of all key terms, abbreviations, and acronyms used in MIL-HDBK-284-1 and MIL-HDBK-284-2. Part 3 also contains definitions of other terms related to military training and ICW.

2. APPLICABLE DOCUMENTS

2.1 Government documents

2.1.1 Specifications, standards, and handbooks. The following specifications, standards, and handbooks form a part of this handbook to the extent specified herein.

STANDARD

MILITARY

MIL-STD-1379	Military Training Programs

HANDBOOKS

MILITARY

MIL-HDBK-284-1	Interactive Courseware (ICW) for Military Training, Manager's Guide for Development, Acquisition, and Management of (Part 1 of 3 Parts)
MIL-HDBK-284-3	Interactive Courseware (ICW) for Military Training, Glossary for (Part 3 of 3 Parts)

(Unless otherwise indicated, copies of military specifications, standards and handbooks are available from the Standardization Documents Order Desk, Building 4D, 700 Robbins Avenue, Philadelphia, PA 19111-5094.)

2.1.2 Other Government documents, drawings, and publications. The following other Government documents, drawings, and publications form a part of this document to the extent specified herein.

PUBLICATIONS

DEPARTMENT OF DEFENSE

DoD Instruction 1322.20	Development and Management of Interactive Courseware (ICW) for Military Training

(Copies of DoD Instruction 1322.20 are available from the Navy Aviation Supply Office, Physical Distribution Division, 5801 Tabor Avenue, Philadelphia, PA 19120-5099.)

2.2 Non-Government publications. The following documents form a part of this document to the extent specified herein.

INTERACTIVE MULTIMEDIA ASSOCIATION (IMA)

Recommended Practices for Multimedia Portability, (MS-DOS Based Systems), Release R 1.1.

(Application for copies should be addressed to the Interactive Multimedia Association (IMA), 3 Church Circle, Suite 800, Annapolis, MD 21401-1933.)

MICROSOFT PRESS

DUNCAN, Ray. Advanced MS-DOS Programming, 2nd Edition, 1988

(Application for copies should be addressed to the Microsoft Press, 16011 NE 36th Way, Box 97017, Redmond, WA 98073-9717.)

(Non-Government standards and other publications are normally available from the organizations that prepare or distribute the documents. These documents also may be available in or through libraries or other informational services.)

3. DEFINITIONS

3.1 Key terms. Key terms used in this handbook are defined in Section 3 of MIL-HDBK-284-3.

3.2 Related terms. Additional terms related to ICW and military training, but not used as key terms in this handbook, are also defined in Section 3 of MIL-HDBK-284-3.

3.3 Abbreviations and acronyms. Abbreviations and acronyms used throughout this handbook are defined in Section 4 of MIL-HDBK-284-3.

4. GENERAL GUIDANCE

4.1 Introduction. Interactive courseware training systems (ICWTS) hardware will host a variety of operating system and authoring system applications to speed courseware development. Because these integrated ICWTSs are usually bundled with sophisticated and proprietary interfaces:

a. The ICW and authoring system software written to operate on one ICWTS will require expensive reprogramming to adapt it to another ICWTS having its own distinct and proprietary interfaces, since these interfaces are usually proprietary to the manufacturer. These reprogramming costs can be eliminated or significantly reduced by adopting standard software interfaces to shield the ICW and authoring system software from ICWTS hardware variations.

b. Portable ICW standards will solve the problems associated with ICWTS component variations and operating system environments by establishing standard interfaces. Standards will isolate the courseware and authoring systems from compliant hardware variations by defining standard software interface and command architectures. The courseware and authoring systems will interact through an application to an intermediate layer, such as VDI Management, rather than interacting through a proprietary interface or directly with the hardware and operating system.

4.2 DoD PORTCO initiative. The Portable Courseware Project (PORTCO) is a DoD project that recognizes the need for standard software interfaces. DoD adopted the IMA Recommended Practices for Multimedia Portability (MS-DOS Based Systems), Release R 1.1 as the baseline for PORTCO. The PORTCO strategy supports the Government's movement toward open system environments (OSE) and standard software interface and command architectures. For example:

a. Several existing standards could form the basis for an ICW application interface definition that would allow transporting ICW across a family of hardware platforms. In addition to the IMA Recommended Practices, other standards are being evaluated for possible application within the PORTCO architecture. These standards include POSIX, GOSIP, and X Windows.

b. There are two widely accepted operating environments on which to base the PORTCO architecture: the Microsoft Disk Operating System (MS-DOS) environment and the OSE-based UNIX/POSIX environment. PORTCO established the ICWTS using MS-DOS as the baseline operating system through adoption of the IMA Recommended Practices. PORTCO is expected to evolve toward the UNIX/POSIX operating system and the X Windows environment through extension and adaptation of the IMA Recommended Practices. Other environments may be addressed, such as MS Windows, OS/2, or Apple Macintosh.

c. Authoring system and ICW portability should be addressed by defining a standard interchange format that allows for the media presentation features, presentation control flow, and other interface requirements available in Federal and commercial interface definitions. PORTCO standards defined to support DoD ICW portability requirements should be extendable to support new ICW training technologies that will become available over the next few years.

4.2.1 PORTCO/IMA Recommended Practices architecture. The IMA Recommended Practices establish and define an interface between ICW and authoring system applications, and system hardware. The IMA interface occurs between the application layer and VDI Management. The IMA Recommended Practices architecture is shown in Figure 1.

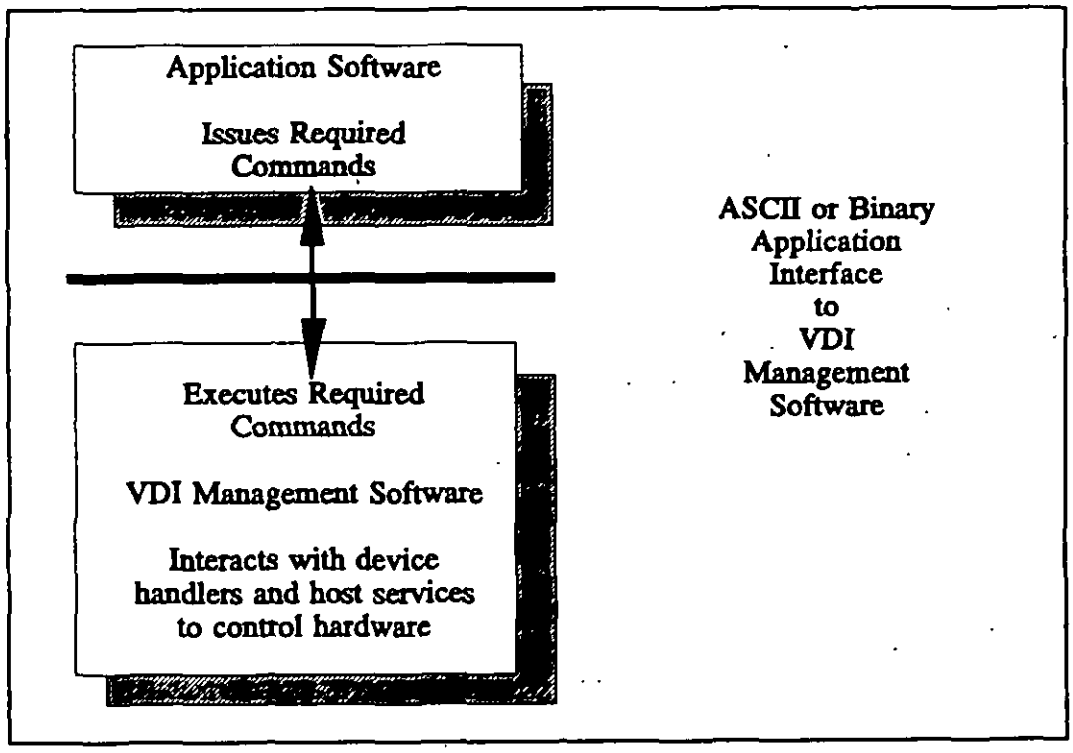

FIGURE 1. General architecture of a compliant system.

4.3 Interactive courseware (ICW) portability practices. The ICW Portability Practices presented in the remainder of this handbook provide information and guidance for implementing the requirements of MIL-STD-1379, Appendix D, "Software Interface and Command Requirements for Interactive Courseware and Authoring Systems". These practices are direct conversions of the IMA Recommended Practices (See 4.3.4).

4.3.1 Introduction. This handbook presents the IMA recommendations for commands and interface mechanisms used in level-three ICW systems. The recommendations are based on functional definitions that were developed with input from the

IMA membership, and adopted by the DoD for implementation in military ICW training programs. The foundation for the IMA definitions includes an extensive set of models that were developed to ensure an approach that addresses current needs and capabilities while providing for future growth in ICW technology.

4.3.1.1 Scope of the ICW portability practices. The ICW portability practices in this handbook and the IMA Recommended Practices provide platform independence but not device interoperability (plug-and-play). Device interoperability requires classes of related devices to furnish functionally identical services at the component level. This document does not address device interoperability. Platform independence lets applications run without modification on any hardware platform based on the same general class of host computers. It requires consistent behavior from different hardware platforms at the application-interface level.

4.3.1.2 Goals of the ICW portability practices. The specific goal of the software interface and command requirements prescribed by MIL-STD-1379, and implemented using the information and guidance in this handbook is furnishing behavioral consistency in ICW applications and authoring systems developed or purchased by the DoD. This goal recognizes that:

a. ICW technology fulfills many types of training and educational requirements, yet platform-specific courseware has severely restricted its use. Many applications require proprietary software, special hardware, or both. There is a critical need for compatibility in ICW technology to promote its usage to meet DoD training requirements; lower costs for integrators, developers, and users; and encourage integration into training program architectures.

b. Behavioral consistency will support operating unmodified ICW application programs on different ICWTSs when both the application and the ICWTS are compliant. However, this handbook does not endorse specific ICW hardware systems or mandate the use of products from specific suppliers.

4.3.1.3 Benefits of the ICW portability practices. DoD adopted the IMA Recommended Practices to establish the standard software interface and command requirements in Appendix D of MIL-STD-1379, and develop the ICW Portability Practices in this handbook. This will benefit the DoD and the ICW industry by furnishing a common ground for manufacturers, systems integrators, courseware developers, and end-users in a field of rapidly changing technologies. Expected benefits include:

a. Broader acceptance of ICW technology and subsequent growth in ICW applications to meet increased DoD demands for quality training.

b. Assurance of long-lasting investments in ICW applications and systems.

c. Reduced need to change and support ICW applications for specific hardware configurations.

d. Increased productivity, lower maintenance costs, and improved application consistency across platforms.

e. Higher quality and less costly ICW applications resulting from a larger marketplace, and the application of resources to courseware development instead of customizing applications, software, and delivery platforms.

f. Greater compatibility between international ICW communities.

4.3.1.4 Handbook conventions. Familiarity with the conventions used in this handbook will aid understanding. These conventions are described as follows:

a. Figures and tables are referenced in the text by number. Figures and tables in the body of the handbook are numbered sequentially starting with 1. Numbers of figures and tables included in an appendix consist of the appendix letter followed by a sequential figure or table number starting with one for each appendix. For example: Table 3 refers to the third table in the body of the handbook; Figure B-3 refers to the third figure in Appendix B.

b. The term "MS-DOS" is used in a generic sense for Microsoft MS-DOS versions 2.0 and higher and compatible operating systems, such as IBM PC DOS versions 2.0 and higher.

c. Numbers are given either in hexadecimal, binary, or decimal format. Hexadecimal numbers have an "H" suffix, for example 006FH. Similarly, binary numbers have a "B" suffix, for example 00001011B. Decimal numbers are written without a suffix.

d. Command names and parameters in the text are in bold face. Parameters consist of the parameter labels and optional parameter values. In the example **vdPlay from=1000**, **vdPlay** is a command, **from** is a parameter label, and **1000** is a parameter value. Command names use mixed lower and upper case for clarity, while parameters use lower case only. However, case is not significant.

e. Examples of ASCII interface commands include calling strings and return strings. In the examples, semicolons separate commands from explanatory comments. This is strictly a convention of convenience and does not imply that the semicolon is a syntactic comment delimiter.

f. Examples of binary interface commands include microprocessor register contents when the interface is called and after it returns control to the application. Again, semicolons separate commands from explanatory comments.

g. The binary examples use memory addresses based on the Intel 80X86 microprocessor ES and DI registers. These registers are used for the addresses of parameter packets that contain one or more structures: each structure consists

of a parameter token number and its associated value. The addresses of individual parameters and values within parameter packets are given as hexadecimal offsets in bytes from the base address in ES:DI. For example, ES:DI[10] is the area of memory 10H (16 decimal) bytes after the memory location pointed to by the combined segment and offset address in ES:DI.

4.3.2 System overview. Figure 1 shows the general software architecture of a compliant system. The ICW Portability Practices are based on sets of high-level commands organized functionally by service group. These commands are issued by an application and passed via an ASCII or binary application interface to the Virtual Device Interface (VDI) Management Software. In the context of this handbook, the term "application" includes authoring system software that may be used to develop or run the application. VDI Management, in turn, executes the commands by calling appropriate low-level services, and passes responses back to the application through the application interface. VDI Management is responsible for making different delivery platforms functionally compatible at the application interface level which allows implementation of platform independent commands. In addition:

a. Although this document describes interfaces for MS-DOS, it is anticipated that other platforms will be addressed in the evolution of both the IMA Recommended Practices and the DoD PORTCO initiative. ICW systems based on the Apple II and the Macintosh are widely used, and applications for OS/2 and Microsoft Windows are starting to appear. Applications also exist for UNIX, VMS, and others.

b. Government procurement policies encourage including OSE application portability profile applications, such as POSIX, GOSIP, and XWindows in contract solicitation documents. While the PORTCO initiative supports this, the architecture will maintain the general approach and command structure established by MIL-STD-1379, Appendix D.

c. The ICW Portability Practices contained in this handbook are designed with the intent to adapt them to other operating systems and non-80X86 hardware platforms. The general structure and functionality of the commands are transferable to other environments. However, substantial portions of the current recommendations, especially the ASCII and binary interfaces, are specific to MS-DOS of necessity.

4.3.2.1 Hardware and operating system assumptions. The ICW Portability Practices make several assumptions about the computer, its operating system, and the ICW hardware. These assumptions limit the scope of the ICW Portability Practices to applications for which these assumptions are true. These assumptions include:

a. The computer is based on an Intel 80X86 or functionally equivalent microprocessor with MS-DOS version 2.0 or higher, PC-DOS version 2.0 or higher, or a functionally equivalent operating system operating in real mode.

b. The computer uses an IBM PC-XT- or PC-AT-compatible ROM BIOS.

c. The computer is based on either an IBM PC-XT, IBM PC-AT, IBM Microchannel, or an Enhanced Industry-standard Architecture (EISA) bus.

d. The system has a graphics/video overlay capability using Color Graphics Adaptor (CGA), Enhanced Graphics Adaptor (EGA), or Video Graphics Array (VGA) graphics; or uses two monitors, one with video and the other with graphics. Whether VDI Management developers implement CGA, EGA, or VGA graphics, the implementation must be compatible with the graphics standard at the ROM BIOS level. (Enclosure 4, DoD Instruction 1322.20).

e. One or more of several XY-input devices may be present (touch screen, mouse, light pen, bit pad, or other).

f. One or more videodisc players or functionally equivalent video sources may be present.

4.3.2.2 Interface and command design criteria. The IMA Recommended Practices used the criteria presented below to define a software platform that should furnish a high degree of portability for a variety of ICW applications while keeping implementation costs and run-time resource requirements to a minimum. The criteria used in designing the IMA interfaces as adopted by the DoD and included in the ICW Portability Practices include:

a. The ICW Portability Practices should not keep application authors from choosing appropriate languages for the tasks at hand. Compatible languages should include, but not necessarily be limited to, general purpose programming languages, ICW authoring systems, artificial intelligence (AI) tools, prototyping systems, and database managers.

b. Access mechanisms should be as consistent as possible, both for a single operating system and across different operating systems.

c. It should be possible to upgrade the ICW Portability Practices to support and implement technological developments without affecting existing applications.

d. The ICW Portability Practices should include both simple, easy-to-use commands for doing simple tasks and sophisticated functions to support the most demanding ICW applications.

e. The commands should not depend on any one operating system, though the interfaces must to some extent be specific to individual operating systems.

f. Except for features that affect the application interfaces, the ICW Portability Practices should not require a specific VDI Management software architecture.

g. Hardware-specific assumptions should be defined in detail.

h. Memory requirements and performance costs should be kept to a minimum. Therefore, implementation decisions should side with simplicity when possible.

4.3.2.3 The rationale for two interfaces. The goal of supporting a variety of programming languages led the IMA to describe two interfaces: an ASCII interface and a binary interface. The rationale for these interfaces includes:

a. One MS-DOS interface should include an installable device driver capable of standard ASCII communications because some programming systems, including several popular interpreters, have primitive facilities for interfacing with other software. However, almost all programming systems can access files and, therefore, use device drivers.

b. More sophisticated programming systems that can issue software interrupts should have a more efficient interface available. This is supported by the binary interface that accesses VDI Management through a software interrupt.

c. The ICW Portability Practices command list is the same for both interfaces, but additional parameters are available to binary interface programmers. For example, the binary interface can pass an entire palette as a pointer to an array of individual palette colors. This is difficult to express with ASCII strings and, therefore, not supported by the ASCII interface.

c. Each command name in the ASCII interface has a binary interface token number that furnishes either the same functionality or a superset thereof. Each ASCII parameter name has a corresponding binary parameter token number. The high degree of consistency between the two interfaces simplifies their implementation and use.

d. Future versions of the ICW Portability Practices may include commands that are available from the binary interface only. However, the facilities furnished by the ASCII interface will remain a strict subset of those available from the binary - interface.

4.3.2.4 Service groups and command organization. Commands that map to related services are separated into service groups. VDI Management and application developers can use only those service groups that include functions required for a given application. This organization also supports adding new service groups in a modular fashion as new capabilities become available. Examples of service groups that may be considered for future addition include: windowing environments and graphical user interfaces, audio management, digital audio, and digital video.

4.3.2.5 Service groups. Currently, the ICW Portability Practices include four service groups: general system (sy), visual management (vm), videodisc (vd), and XY-input device

(xy). The sy service group does system-level hardware and software initialization, furnishes information about the availability of other service groups, implements a command queue, and has commands to help with error handling. The other groups address the functional areas for which they are named. However, the groupings are for convenience of organization and do not necessarily dictate which hardware must actually perform the commands. Additionally:

a. The general system group (sy) is the only group that must be included in all compliant implementations. However, if a technology is covered by an existing service group, it should be supported by that service group. For example, videodisc players should be controlled using the command set in the videodisc (vd) service group only. No proprietary commands should be used.

b. VDI Management developers are responsible for ensuring correct installation of system hardware and software and for making a list of installed service groups available to the application through the system commands. VDI developers also must include a way for users to assign contiguous, logical numbers starting with zero to specific devices within a service group when more than one such device is present. Note that this lets users specify the order of devices within each class, but does not let users assign arbitrary numbers to the devices. Section 4 describes these requirements.

c. A requirement for at least two additional service groups has been identified: a digital audio service group, and an audio management service group. These two groups are not defined: Appendixes E and F are reserved to support them once service group requirements are identified.

4.3.2.6 Core and extended commands and parameters. The command set includes both core and extended commands. Individual commands, in turn, may include both core and extended parameters.

4.3.2.6.1 Core commands and parameters. Core commands in a given service group furnish the general functionality required by ICW applications. If a given service group is implemented, all core commands for that group must also be implemented to be compliant. Similarly, if a command is implemented, all core parameters for the command must be supported.

4.3.2.6.2 Extended commands and parameters. Extended commands and parameters are provided for developers who need to produce both portable courseware, and applications that need special, nonportable capabilities; or who want to take advantage of these capabilities if present and properly handle their absence. Extended commands and parameters are not required for compliance with MIL-STD-1379, Appendix D. However, an extended command may include core parameters. While inclusion of the command is optional, if it is implemented, inclusion of the command's core parameters is required to be compliant. In addition:

a. Because VDI Management developers must support core commands and parameters as a prerequisite to supporting extended commands and parameters, VDI developers can write both portable and nonportable applications using the same tools, authoring systems, and custom libraries.

b. Some extended commands and parameters may be considered for future inclusion in the core command set. However, this does not guarantee that they will be included; an application that relies on these commands and parameters is noncompliant. Except, an application that uses extended commands or parameters when present and still executes properly in their absence is compliant.

4.3.3 IBM PC and compatible graphics modes. Table 1 lists standard graphics modes returned by BIOS interrupt 10H, service 0FH for IBM and compatible personal computers. Graphics support requirements include the following:

a. Compliant systems need not support all listed modes and may support but not require unlisted, nonstandard modes. However, if a system claims support for a listed mode, the mode must be supported as listed. Supporting a standard mode in a nonstandard, nonportable manner makes a system noncompliant.

b. An application written for a compliant system based on one adapter may not be portable to a compliant system based on a different adapter. For example, a compliant application that uses VGA mode 19 will not be portable to a compliant system that is limited to EGA modes.

c. Any VDI Management implementation that requires applications to use a nonstandard graphics mode is noncompliant. Any application that requires a nonstandard mode is noncompliant.

4.3.3.1 Modes 0-3 restrictions. Modes 0 through 3 are overlay modes for all adapter types. Therefore, they are restricted to 200 lines in NTSC and PAL video modes regardless of how many lines the adapter would normally use.

4.3.4 Tracking IMA Recommended Practices. Because this handbook is based almost entirely upon the IMA Recommended Practices, Appendix J provides a cross-reference of handbook paragraphs, tables, and figures to their equivalent in the IMA Recommended Practices. This appendix also cross-references handbook paragraphs, tables, and figures to related information in Appendix D of MIL-STD-1379.

5. DETAILED GUIDANCE

5.1 Interface application. The portability practices include two interfaces for MS-DOS; the binary interface and the ASCII interface. By providing two interfaces, the portability practices can be used with a variety of programming languages. Languages that can issue interrupts will typically use the binary interface for speed and efficiency. Languages without this capability can use standard file I/O and ASCII strings with the ASCII interface. The rest of section 5.1 explains how to use both interfaces. Section 5.2 gives detailed information on implementing the interfaces. The basic characteristics of each interface are described as follows.

5.1.1 The binary interface. When using the binary interface, applications communicate with VDI Management by passing and receiving binary values across a software interrupt. The binary interface uses an assignable software interrupt in the range 60H-66H to request VDI Management software services. An application loads the microprocessor's registers with a command code requesting a specific service and a pointer to a parameter packet containing parameter codes and values.

5.1.2 The ASCII interface. When using the ASCII interface, applications communicate with VDI Management through a device driver using file I/O and ASCII strings. The ASCII interface uses an MS-DOS device driver for communications between the application and VDI Management. The application writes command strings to, and reads response strings from the device driver. A command string consists of a command name followed by parameters and values.

5.1.3 Introduction to parameters and values. Both interfaces use labeled parameters and associated values. Parameters may have associated calling values, return values, or both. Every parameter value passed to VDI Management is associated with a parameter identifier. Therefore, only those parameters that are actually required by the command need be specified. For example:

a. Consider the **vdPlay** command, binary interface token number 3081. **vdPlay** instructs the player to play a video segment. Applications can accompany this command with several optional parameters including **from**, **to**, and **speed**.

   (1) A **from** parameter and its associated value causes the player to search to a specified frame before entering play mode. Without the **from**, the play starts with the frame that is current when the application issues the command.

   (2) A **to** parameter causes the player to play to a specified frame and stop. Without the **to**, the play continues until the application intervenes or the player reaches the edge of the videodisc.

   (3) With a **speed** parameter, the segment plays at the specified speed. Without a **speed**, the segment plays at the normal speed of either 30 or 25

frames per second depending on the video standard, NTSC or Phased Alternations of Lines (PAL), respectively.

b. With the ASCII interface, each parameter value must be preceded by a parameter name so that a parser can decipher the supplied arguments. However, each parameter name does not necessarily require an associated value. Some parameters used to query the system do not have associated values. A command string is variable in length with its length determined by the number of parameters and values.

c. With the binary interface the parameter packet is an array of parameter structures. Each structure contains a parameter's numeric identifier and the parameter's value or, for parameters without associated calling values, the space for a return value. The parameter packet size depends on the number of parameters.

d. Continuing the **vdPlay** example, to use the ASCII interface to play from the current frame to the edge of the disc or until a vdStill command is issued requires simply: **vdPlay**. However:

(1) To search to frame 1000, then play to frame 1500 use: **vdPlay from=1000, to=1500.**

(2) You might also use: **vdPlay from 1000 to 1500; vdPlay from, 1000, to, 1500;** or **VDPLAY TO 1500 FROM 1000.** All are equivalent because case and parameter order are not significant, and equals signs and commas are optional.

e. For **vdPlay** without a parameter, the binary interface does not pass a parameter packet with **vdPlay**. In the examples in (1) and (2), above, it passes a packet containing four 32-bit memory blocks. The first block contains the 24 decimal -- the token number for the from parameter. The second block contains 1000 -- the value of **from**. The third block contains 48 -- the token number for the to parameter, and the fourth contains **1500** -- the value of **to**.

5.1.3.1 Parameter order. The order of parameters is insignificant with both the binary and ASCII interfaces. Because order is insignificant, supplying a parameter more than once in a command string or parameter block is an error. This simplifies designing VDI Management parsers and indirectly establishes the maximum number of parameters that can be passed to either interface in a single call. This maximum is equal to the number of parameters for the single command with the largest number of defined parameters.

5.1.4 Using the binary interface. This section explains how to use the binary interface under MS-DOS and gives detailed information on parameter packets and processor registers.

5.1.4.1 General procedure. Applications will typically take the following steps to call VDI Management through the binary interface: (The software interrupt used depends on the contents of the environment space, see 5.2.4.1).

a. Build in application memory a packet containing parameter token numbers and values to pass to VDI Management with the command.

b. Load the token number for the command into the AX register.

c. Load the number of parameters contained in the packet into the BX register.

d. Load the segment and offset address of the parameter packet into the ES:DI register pair. (The segment address goes in ES and the offset relative to that segment goes in DI).

e. Issue the appropriate software interrupt.

f. On return from the software interrupt, check the value of AX. If AX is not zero, it contains an error code (see 5.5) and the application should take appropriate action to recover.

5.1.4.2 Confirming the binary interface. Calling the binary interface through its interrupt vector could have dire results if the interface is not installed. Therefore, applications that use this interface need a way to confirm its existence without calling it with a command such as **sylnit**. The binary interface includes a 16-byte signature of the form "IVPRACTICES" followed by five NUL (OOH) characters to verify the interface is installed. This signature resides at the address pointed to by the interrupt vector specified by the IVINT environment variable minus 10H or, if IVINT is not set, at the address pointed to by the default interrupt vector, 60H, minus 10H (see 5.2.4.1). Applications should also:

a. Determine the address pointed to by the interrupt vector specified by IVINT or if IVINT is not set, the default interrupt vector 60H, then retrieve the information stored in the 16-byte paragraph immediately preceding that address to confirm that the binary interface is installed. The ICW Portability Practices assume that languages that use the binary interface have this capability.

b. Use **sygetstate** to confirm the version number of the VDI Management module.

5.1.4.3 Parameter packets. A parameter packet contains 8 bytes of memory for each parameter. The 8-bytes are divided into two 32-bit blocks. The first 32-bits contain the parameter token number; the second 32-bits contain the parameter value (see 5.2.4.2 for detailed information on parameter value formats).

5.1.4.3.1 Parameter token numbers. Parameter token numbers are constants, as defined in paragraph 5.3.1.2. VDI Management uses them to determine which parameters the application is passing. Parameter values may take different types depending on the

information being passed. Valid types include signed 32-bit integers, pointers consisting of a 16-bit segment and a 16-bit offset, and 32-bit unsigned bit fields. For example:

a. *Assume an application has initialized VDI Management and displayed video on the* monitor. The application now decides to play a video segment starting at frame 1000 and ending at frame 2000. The application sets up the registers and parameter packet as follows:

AX	3081	; token number of **vdPlay** command
BX	2	; packet contains two parameters

ES:DI is a pointer to a packet containing:

Block 1	24	; token number of **from** parameter
Block 2	1000	; value of **from** parameter
Block 3	48	; token number of **to** parameter
Block 4	2000	; *value of **to** parameter*

b. All values in the above example use decimal notation.

5.1.4.3.2 Parameter packet length. The length of the parameter packet varies with the number of parameters. A packet with five parameters is laid out as shown in Table 2. Note that addresses of parameter tokens and values within a packet are described as hexadecimal offsets in bytes from the base address in ES:DI. For example, ES:DI[20] is the area of memory 20H (32 decimal) bytes after the location pointed to by ES:DI. Each 32-bit block uses 4 bytes, so this is the fifth block in the packet. Longer packets are possible by simply adding the extra parameters at the end. In addition:

a. To determine how much memory to allocate for a parameter block that can *handle any command for the binary interface, multiply the maximum number of* parameters that can be passed by any command by the memory required for one parameter. For example, assume that **syBigCommand** has the longest parameter list at 22 parameters. Each parameter uses 4 bytes for its token number and 4 for its value. Therefore, allocate 8 x 22 bytes or a 176-byte block.

b. When the binary interface is called, registers other than AX, BX and ES:DI are insignificant and may contain any value. If a command has no parameters, BX is zero and ES:DI are insignificant.

c. On return, all registers are unchanged except AX. AX contains zero, if the command was successful, or an error code indicating a problem.

5.1.4.4 Return values to parameters. *The contents of a parameter packet may* change depending on the nature of the command. Some commands, such as vdGetState, return information to the application. When such a command executes, the application

passes a parameter packet with sufficient space for the requested information. The space is allocated using the format described in paragraph 5.1.4.3. In addition:

a. A parameter/value structure both defines the requested information and furnishes a return location. On entry into VDI Management, the parameter value is insignificant and can be any value. When VDI Management has derived the requested value, either by self inspection, calculation, or interrogating the hardware, it puts the value into the appropriate parameter value block.

b. For **vdGetState frame**, the application might pass:

AX	3078	; token number of **vdGetState** command
BX	1	; packet contains one parameter
ES:DI[0]	23	; token number of **frame** parameter
ES:DI[4]	Can be any value	

On return, the values might be:

AX	0	; command executed correctly
BX	1	; packet contains one parameter
ES:DI[0]	23	; token number of **frame** parameter
ES:DI[4]	12345	; frame number 12345 is currently displayed

All values in the above example use decimal notation.

c. Applications should check AX and assume that if an error has occurred (AX ≠ 0), any values returned in the packet are meaningless.

d. If the binary interface returns a pointer to a string, the string is in upper case. This is an arbitrary decision to make return strings easier to parse and to make binary and ASCII return strings consistent.

e. VDI Management may round parameter values such as play speeds. If so, VDI Management changes the values in the parameter packet to the actual values used after rounding; subsequent queries return actual values instead of requested values. Refer to the applicable service group appendix for specific information on rounding of command parameter values.

5.1.5 Using the ASCII interface. This section explains how to use the ASCII interface under MS-DOS, and includes an explanation of command and response strings.

5.1.5.1 General procedure. To issue commands to VDI Management via the ASCII interface, applications use standard file I/O to communicate with a device driver named IVDEV. The exact method of communicating with the driver depends on the programming system. Applications will typically take the following steps to call VDI Management through the ASCII interface:

a. Format a command string specifying the required command and parameters.

b. Open the device driver for writing.

c. Write the command string to the device driver.

d. Close the device driver.

e. Open the device driver for reading.

f. Read the response string from the device driver.

g. Close the device driver.

h. Parse the response string. If the string is "OK" or contains expected return values, continue processing normally. If the string is "ERROR n..." where "n..." is an error number, take action to recover from the error.

5.1.5.2 Confirming the ASCII interface. Applications that use the ASCII interface must be able to confirm that the interface and its associated device driver, IVDEV, are properly installed. Simply verifying that IVDEV can be opened is insufficient because some languages may automatically create a file named "IVDEV" if it does not exist. Therefore, an application should open the file, write a syinit command to it (see Appendix A) and read the response string. Typical responses are:

a. If the response string consists of the ASCII characters "OK" followed by CR/LF, syinit was successful. The application should continue normally.

b. If the response string consists of "ERROR n..." where "n..." is an error number followed by CR/LF, the ASCII interface is installed, but VDI Management could not be initialized, indicating improper installation or improper use of syinit. The application should handle the error, probably by displaying an error message and exiting.

c. If the function used to read IVDEV returns a read error or the response string consists of anything other than "OK" or "ERROR n...," either the device driver is not installed and the file open function has created a bogus IVDEV file, or serious problems exist with VDI Management. The application should exit with an error message and, if necessary, either delete the bogus file (preferred) or tell the user that a bogus IVDEV file may have been created.

d. If IVDEV is not installed and a file of the same name is automatically created, it will contain the string the application tried to write to the driver. If this happens, and IVDEV is then installed as a device driver, the bogus file cannot be deleted from the command line because MS-DOS will assume it is supposed to delete

the device driver, which is illegal. Therefore, the file must be deleted with the device driver unloaded.

5.1.5.3 Command strings. A command string is a series of printable ASCII characters terminated with a carriage return (CR, ASCII 0DH). The ASCII interface discards line feeds (LF, ASCII 0AH) following the CR. The string starts with a command name typically followed by a parameter list. Paragraph 5.2.3 gives detailed information on formal command string syntax and parameter value data types. For example:

a. Assume an application has initialized VDI Management and turned video on. To play a videodisc segment starting at frame 1000 and ending at frame 2000, the application could issue the command string "vdPlay from=1000,to=2000 [CR]." vdPlay is the command name. It corresponds to the token number passed in the AX register with the binary interface. The substrings from and to are parameter labels used by VDI Management to determine which parameters are being passed. The substrings "1000" and "2000" are parameter values. Each value is associated with the preceding label, so 1000 is the value of from and 2000 is the value of to.

b. Because the string ends with CR, VDI Management can determine how many parameters the application is passing by inspection. Unlike the binary interface, the ASCII interface does not require a parameter count.

5.1.5.4 Response strings. Response string contents from the ASCII interface depend on the nature of the command string. If the command was correct and did not ask for information, the response string is "OK". If the command asked for information, the ASCII interface returns a series of comma-separated parameter values. If an error occurred, the response string consists of "ERROR" followed by a space, followed by the error number as ASCII digits. All response strings end with CR/LF. For example, consider:

a. Issue the command: "vdPlay from=1000,to=2000." If this executes correctly, the ASCII interface returns "OK". However, the command: "vdPlay from=1000,from=2000" generates the response string "ERROR 54" (Parameter used more than once).

b. If an application issues the command: "vmGetState vlevel,glevel" which asks about graphics and video fade levels, the response might be "255,200". This says that the video level is 255 and the graphics level is 200. Note there is no space following the comma that separates these two return values.

c. All alpha characters returned by the ASCII interface are upper case. This is an arbitrary decision to make return strings easier to parse and consistent between implementations.

5.1.6 Mixing ASCII and binary commands. For a VDI Management module to be compliant, it must furnish both the device driver and software interrupt access methods,

although both do not have to be installed. If both methods are loaded, VDI Management should behave correctly when an application mixes ASCII and binary commands. For example, a well behaved VDI Management should:

a. If an application issues syQueue on via the binary interface then issues a series of commands to the device driver, the commands issued via the driver should be queued correctly. Similarly, an ASCII command and its binary counterpart should behave identically assuming both commands have the same parameters available.

b. To conserve memory and enhance performance, applications that use only one interface should clearly state whether the ASCII or binary interface should be installed. Application vendors also should state the amount of memory required by the application after VDI management has been loaded to alert users of any potential memory problems.

5.2 Interface implementation. This section discusses implementation details for VDI Management, and the ASCII and binary interfaces. It includes installation issues, operating system requirements and reentrancy issues, ASCII string and parameter value formats, device driver buffer behavior and communications modes, establishing the binary software interrupt number, and binary data types and formats.

5.2.1 Installation issues. Installation issues include the installation of VDI Management and assigning logical device numbers at the time of installation. The following paragraphs discuss these issues.

5.2.1.1 VDI Management installation. Specific methods of installing VDI Management are outside the scope of the ICW Portability Practices. VDI developers can use any appropriate method. Two possibilities are terminate-and-stay- resident (TSR) software, and software that takes an application name as a command-line parameter and spawns the application (see 5.1.6b).

5.2.1.2 Logical device numbers. Some commands accept a logical device number as a parameter. These include videodisc commands (see Appendix C), which may be used on systems with multiple videodisc players, and XY-input commands (see Appendix D), which may be used on systems with multiple XY-input devices.

5.2.1.2.1 Multiple devices. Support for multiple devices within a service group is not a compliance requirement. For example, if an integrated system supports only one videodisc player, the VDI developer can implement a VDI Management module that supports only one player as logical device number zero, and be compliant. To require VDI support for multiple, unspecified devices would equate to establishing device interoperability (see 4.3.1.1).

5.2.1.2.2 Assigning logical device numbers. When logical device numbers are allocated, VDI Management lets users assign numbers to specific devices using an

appropriate set up facility. Device numbers within a service group must be contiguous and start at zero, and:

a. VDI Management does not assume that any particular type of device within a service group will be assigned any particular logical device number. For example, if both PAL and NTSC videodisc players are installed, VDI Management makes no assumptions about which type of player is assigned logical device number zero, and which is assigned number one.

b. Logical device numbers are separate for each service group. For example, players can be assigned contiguous logical device numbers 0 through 15 starting with zero. XY-input devices can also be assigned contiguous logical device numbers from 0 through 15 starting with zero. If there is one device in a service group, then that device is assigned logical device number zero; if there are two devices, they are assigned logical device numbers zero and one.

5.2.1.2.3 Device mapping. Because of the range of available physical devices and the need to furnish ongoing support for existing applications, it is impossible to prescribe a general way for applications to examine or change the device number mapping. If an application asks for the device type and discovers a device that was not invented when the application was written, it could not use this information while executing. However, applications will often need to know the relationship between logical device numbers and the physical devices. Therefore, application authors must clearly state any requirements for a particular device number mapping, so that users can set up VDI Management appropriately. Mapping requirements include:

a. Assume, for example, that a system has two videodisc players. One player supports PAL and the other supports NTSC. These players can be mapped to logical device numbers in exactly two ways: the NTSC player can be installed as device zero and the PAL player as device one, or the PAL player as device zero and the NTSC player as device one. An application that requires the NTSC player be device zero cannot change the logical device numbering at run time. Therefore, the application developer must clearly inform the user that the NTSC player must be installed as device zero when the user installs VDI Management.

b. If an application needs information about the mapping, it must have a mechanism for the user to provide that information. Appropriate techniques include command-line parameters and files in application-specific formats. If necessary, well written applications should use an installation program that asks which logical device number to use for each peripheral. Well documented applications should carefully explain what kinds of physical devices are appropriate for each selection in the installation procedure.

c. For each service group that supports multiple devices, providing a setup facility for users to assign logical device numbers when VDI Management is installed is a compliance requirement (MIL-STD-1379, Appendix D). Assigning contiguous

device numbers starting with zero for each service group that supports multiple devices is also a compliance requirement.

5.2.2 Operating system issues. Operating system issues include basic operating system requirements and considerations about the lack of reentrancy under MS-DOS. The following paragraphs discuss these issues.

5.2.2.1 Operating system requirements. To support the MS-DOS version of the ICW *Portability Practices, the operating system is MS-DOS version 2.0 or later, or a* functionally equivalent operating system. Versions of MS-DOS prior to 2.0 cannot use installable device drivers. Therefore, systems that cannot run MS-DOS 2.0 or later cannot comply with the MS-DOS version of the portability practices regardless of the attached ICW hardware.

5.2.2.2 Specific version MS-DOS compliance. Developers may provide VDI Management software that requires a specific version of MS-DOS. They should describe this type of application as either "Compliant only when used with MS-DOS version N.nn" or, more often, "Compliant only when used with MS-DOS version N.nn or later." Such VDI Management software should test the MS-DOS version number and decline to *execute if it is not supported. Similarly, application authors may require a specific version* of MS-DOS. Therefore, users of ICW hardware and courseware should ensure that the versions of MS-DOS, VDI Management, and courseware they purchase are compatible.

5.2.2.3 MS-DOS reentrancy limitations. At certain times — specifically when MS-DOS has suspended processing a function request to service an interrupt — programs are not allowed to call MS-DOS. Consider the tick chain. Approximately 18 times per second, the operating system calls all routines on the tick chain. Tick routines that call MS-DOS must check the MS-DOS critical section flag to ensure that they do not call MS-DOS at an inappropriate time. VDI developers should consider the following:

a. *Typically, high-level programmers need not be concerned about such issues.* Unless a program includes interrupt handlers or tick routines, it will not have control when MS-DOS cannot be called. If a programming system links an interrupt automatically, the system's design ensures that conflicts are handled correctly.

b. Programmers who simply call MS-DOS or BIOS functions using standard methods will not encounter problems. However, those who use the tick chain or change MS-DOS or BIOS interrupt vectors must deal with the lack of reentrancy.

c. VDI Management assumes it can call MS-DOS after any call to the binary *interface. Therefore, application software should not call the binary interface* when it is unsafe to call MS-DOS, or when a previous call to VDI Management has been interrupted. If an application installs interrupt handlers, it should also provide mechanisms to ensure that VDI Management is called at appropriate times only.

5.2.2.4 Background processing. If VDI Management does any background processing outside of application calls, it must ensure that MS-DOS is called only when it is safe to do so. Also, the device driver, which is loaded within MS-DOS, must not be called at those times when it is unsafe to call MS-DOS or when VDI Management has been interrupted. In addition:

a. VDI Management may need to do some background processing, and the device driver may have to do some work of its own to overcome reentrancy limitations. For example, assume an ASCII command uses the filing system. Within a device driver, the MS-DOS filing system is already executing and may not be reentered. If the device driver or its support routines must call MS-DOS, the request should be queued and issued after the driver has returned. One method for doing this involves hooking interrupt 21H and executing the DOS call immediately after MS-DOS returns from the device driver.

b. Do not attempt to call the binary interface directly from within the device driver because the interface, in turn, calls VDI Management. VDI Management must then be able to call MS-DOS.

5.2.3 ASCII interface issues. ASCII interface issues include ASCII text string formats, parameter value formats, device driver buffer behavior, device driver communications modes, and using IOCTL functions.

5.2.3.1 ASCII string formats. Command strings are simply tokens separated by delimiters. Return strings consist of "OK", "ERROR n...", or comma-separated values. The following paragraphs describe command and return strings formats.

5.2.3.1.1 Command string tokens. A command string is a series of tokens, separated by delimiters, and ending with a CR (ASCII 0DH). Tokens are strings of one or more printable characters. They include command names, parameter identifiers, and parameter values. Command names consist of characters in the ranges "A" to "Z" and "a" to "z." Case is not significant. The command name "vdPlay" could, for example, be supplied as "vdplay," "VDPLAY," or even "Vdplay." Parameter labels consist of characters "A" to "Z", "a" to "z", and "1" and "2". Again, case is not significant. Parameter values consist of characters in the ranges "a" to "z", "A" to "Z", and "0" to "9", and the characters "+" and "-".

5.2.3.1.2 Command string delimiters. Delimiters are the characters equals sign (ASCII 3DH), space (ASCII 20H), HT (ASCII 09H), LF (ASCII 0AH) and comma (ASCII 2CH). Redundant delimiters are ignored. They include all leading delimiters, all trailing delimiters between the last token and the CR, all trailing delimiters after the CR, and any instance of two or more adjacent delimiters.

5.2.3.1.3 Command string length. Command strings can be at most 255 characters including the CR. Redundant delimiters and terminal LFs do not count towards the 255-character limit.

5.2.3.1.4 Multiple commands in one string. Multiple commands separated by CRs or CR/LFs may be included in a single string and sent to the ASCII interface with a single write operation. The 255-character length limit not counting redundant delimiters applies to the command string as a whole. If a string contains multiple commands, they all must fit within the 255-character limit. In addition:

a. The response string for a multiple command string will only contain the response to the last command in that string. If an error occurs in a command other than the last command, VDI Management does not abort later commands in the string.

b. Using **syCheckError** is the only way to detect an error occurring before the last command in the string (see Appendix A, **syCheckError**) Because the response string contains the response to the last command only, the response will be lost if a multiple command string contains a query command in any string position except the last.

5.2.3.1.5 Response strings. Response strings always end with the CR/LF pair. Most response strings are either "OK" on success, or "ERROR n..." on failure where "n.." is an error number. However, some commands request parameter values. In this case:

a. If a string contains multiple values, each value is separated from the next by one comma with no spaces. Response strings contain no delimiters before the first value or between the last value and CR/LF.

b. Response strings are limited to 255 characters including the terminal CR/LF. Unlike command strings, the terminal LF counts toward the response string length limit.

5.2.3.2 ASCII string formal syntax. The formal syntax description for ASCII command and response strings shown in Table 3 uses a notation derived from the Backus Naur Form (BNF). This syntax uses the following rules:

a. Angle brackets (< >) enclose items that are defined by the formal descriptions.

b. Vertical bars (|) separate sets of alternatives—in deriving a valid command string, only one of the alternatives should be chosen.

c. Square brackets ([ ]) enclose optional items or sets of items. Their presence or absence does not affect a string's validity.

d. Spaces separate sets of required items that should occur in the given order.

e. The ":=" sign means "consists of." The item on the left of ":=" consists of the definition on the right.

f. The strings "equals", "space", "HT", "LF", and "comma" are delimiters and stand for the indicated characters.

g. "CR" is a terminator and stands for the indicated character.

h. "CR/LF" indicates the two characters carriage return and line feed.

i. Items in quotes (" ") are string literals and stand for themselves.

5.2.3.3 ASCII parameter value formats. ASCII interface parameter values include numeric parameters, decimal integer representations of bit fields, and text. Their formats are discussed below.

5.2.3.3.1 Integers. Numeric parameters include device numbers, mode numbers, time periods, and error numbers. These are passed in decimal integer format, which is defined as an optional sign (" + " or "-") followed by a string of decimal digits.

5.2.3.3.2 Bit fields. A bit field is represented as an unsigned decimal integer. The integer is the sum of the bit values in the field. For example, the support value returned by **syGetState** is derived by treating the Boolean supported/not supported values as elements in a bit field. In addition:

a. Table 4 lists the decimal numbers returned for the different service groups. The value of the bit field is the sum of the values for each group. The system (sy) group is always present if VDI Management is working.

b. Assume a system supports vm and vd, but not xy, da, or am commands. The **syGetState** return value is 1 + 2 + 4 or decimal 7. If the sy, vm, vd, and xy command service groups are supported, the return value is 15.

5.2.3.3.3 ASCII text. Some return strings for query parameters include values other than numbers. These strings consist of at most upper-case alpha characters, decimal digits, commas, and a sign (+ or -).

5.2.3.4 Device driver buffer behavior. The way in which the device driver buffers character strings must be compatible across different versions of MS-DOS. When the driver loads, it allocates a command buffer and a response buffer of 255 bytes each. At start-up, the command buffer is empty and the response buffer contains the string "ERROR 19" (Device driver read before write). If an application tries to read the driver before writing to it, the application reads this string. This applies only to the first time an application accesses the driver after boot time. The response buffer is not automatically reset to "ERROR 19" after an application exits. Also consider that:

a. When the driver leads a character from the DOS transfer buffer, the driver checks to see if it is a redundant delimiter, an LF, or a CR. If it is neither, the driver simply stores the character and returns. If it is a redundant delimiter or an LF,

the driver discards the character and returns. If it is a CR, the driver and VDI Management process the contents of the command buffer. The driver then either returns control to the application, or processes any additional characters that are being written to it. (Note that multiple commands may be included in a single write operation.)

b. When MS-DOS tries to read a character from the driver, the driver checks the response buffer. If the buffer contains characters, the driver returns the first character and deletes it from the buffer. If the buffer is empty, the driver returns end of file (EOF, ASCII 1AH under MS-DOS).

c. Some programming languages internally buffer device driver writes and do not furnish a way to flush the buffer other than closing the file to which it is attached. Users of such languages must be able to force a write to the driver without losing the response to the forced write. Therefore, closing and reopening the driver empties the command buffer but does not change the contents of the response buffer.

d. If one application ends and a new application starts, the driver does not flush the response buffer until it receives a CR. Therefore, an application should not read the buffer before writing at least one command or it may read an invalid response left by the previous application.

e. If an application writes a command longer than 255 characters, the driver discards the command by clearing the buffer and ignoring additional characters up to the next CR. The driver then issues "ERROR 17" (Command too long).

5.2.3.5 Device driver function and mode considerations. Device drivers fall into two categories: block device drivers and character device drivers. VDI ASCII interface drivers will typically be implemented as character device drivers. Several issues must be considered in the implementation and use of character device drivers. These include the use of device driver specific functions, the use of interrupt (INT) 21H file functions, and ASCII (cooked) and binary (raw) driver I/O modes. The following paragraphs discuss these issues to help VDI implementers address them consistently. However, a detailed explanation of device driver implementation is beyond the scope of this handbook. [1]

5.2.3.5.1 Device driver specific functions. Device drivers communicate with MS-DOS using a special interface with two entry points: the strategy routine and the interrupt routine. These routines use a special set of MS-DOS device driver interrupt functions: interrupts 00H through 10H, 13H, 17H, and 18H are used for communications between

[1] Several excellent references exist on device drivers and general MS-DOS programming. One such reference is: DUNCAN, Ray. "Advanced MS-DOS Programming", 2nd Edition (see 2.2).

MS-DOS and the device driver. These device driver specific functions are not interrupt 21H functions.

5.2.3.5.2 Device driver function implementation. Developers must make several choices about which device driver functions to actually implement as opposed to simply supporting the function call without implementing the function. Function call support without implementation is done by setting the done flag in the status header, and returning. The version of MS-DOS and the driver type (character) determine which device driver functions must be supported without implementation, which must actually be implemented, and which are optional.

5.2.3.5.2.1 MS-DOS version 2.0 and higher. Device driver functions 00H to 0CH must be supported. Functions 00H, 04H to 08H, 0AH, and 0BH should be implemented. Implementation of functions 03H and 0CH is optional. Functions 01H, 02H, and 09H apply only to block device drivers: these should be supported, but not implemented.

5.2.3.5.2.2 MS-DOS versions 3.0 and 3.1. Support and implementation requirements are the same as those for version 2.0 and higher, except that functions 0DH, 0EH, and 10H may optionally be implemented for versions 3.0 and 3.1. Function 0FH applies only to block device drivers: it should be supported, but not implemented.

5.2.3.5.2.3 MS-DOS version 3.2 and higher. Support and implementation requirements are the same as those required of version 2.0 and higher, except that functions 0DH, 0EH, 10H, and 13H may optionally be implemented. Functions 0FH, 17H, and 18H only apply to block device drivers: these should be supported, but not implemented.

5.2.3.5.3 Cooked and raw I/O modes. The device driver must support both ASCII (cooked) mode and binary (raw) mode. Support of both modes is a compliance requirement. Application authors may freely use INT 21H; function 44H, subfunction 1 to switch between these modes.

5.2.3.5.4 Driver compliance. Any device driver that meets the criteria listed in 5.2.3.5.2 and 5.2.3.5.3 is compliant. Device driver functions listed as optional are not required. However, if a device driver uses device driver functions that are supported only by MS-DOS version 3.0 or higher, or MS-DOS version 3.2 and higher, the implementation should clearly state the version of MS-DOS required.

5.2.3.5.5 Interrupt 21H file functions. Application authors should note that MS-DOS interrupt 21H supports two classes of file functions. One class manipulates files with file handles; the other uses file control blocks. Only the read/write functions that use handles work with device drivers.

5.2.4 Binary interface issues. Binary interface issues include establishing which software interrupt will be used and parameter value formats.

5.2.4.1 Setting the software interrupt. The binary interface software interrupt is a user interrupt in the range 60H through 66H. The default is interrupt 60H. When VDI Management loads, it checks the environment space for the variable IVINT. The variable value is a two-character string representing a value 60H through 66H, and:

a. The variable is set with a command line in the autoexec.bat file; for example: "set IVINT = 66". If the variable is set, VDI Management loads its interrupt handler at the specified vector. If the variable is not set, the handler is loaded at vector 60H. If the variable value is invalid, VDI Management declines to execute and returns "ERROR 165" (Invalid interrupt number).

b. When an application starts, it also checks for IVINT. If the variable is present, the application uses the specified software interrupt. If IVINT is not present, the application uses the default. (Note that assuming an application has verified the binary interface signature, the application need not verify the interrupt number for validity. If the number is invalid, VDI Management will not load).

5.2.4.2 Binary parameter value formats. Binary parameter values include integers, bit fields, strings, pointers, and color arrays. These formats are described below.

5.2.4.2.1 Integers. Integer quantities are passed as 32-bit, 2's complement, signed numbers, in the range of -2,147,483,648 through +2,147,483,647. This is consistent with most high-level programming languages. Examples of integers include device numbers, graphics mode numbers, error numbers, speeds, and times.

5.2.4.2.2 Bit fields. Bit fields are best viewed as 32 individual bits that can each take a value of 0 or 1. Specific bits in a field correspond to specific items of information that can have only two states -- one or zero: these states correspond to on or off, true or false, or present or absent. Additionally:

a. Table 5 lists bits in the bit field returned by syGetState. This function sets bits according to which service groups are present in a VDI Management installation. Note that the system (sy) group is always present if VDI Management is working.

b. If, for example, the VDI implementation supports vm and vd commands, but not xy, da, or am the least significant byte returned by syGetState is 00000111B or 07H. Refer to Section 60 of Appendix I for a code fragment showing how to analyze bit fields.

5.2.4.2.3 Pointers. For MS-DOS, pointers follow standard far-pointer format. The most significant 16 bits of the pointer contain the segment address; the least significant 16 bits contain the offset within the segment. This is the format used by the majority of MS-DOS compilers, such as Microsoft C and JPI TopSpeed Modula-2.

5.2.4.2.4 Strings. Strings returned via the binary interface are passed by reference via far pointers. In addition:

a. The return value is a far pointer to an ASCII string of up to 255 printable characters followed by a NULL character (00H). Alphabetical characters in the return string are upper case.

b. VDI Management allocates memory to hold return strings. An application should not change this memory even though it knows the string's address. Otherwise, dire consequences may result.

5.2.4.2.5 Color arrays. The **vmGetPalette** and **vmSetPalette** commands use arrays containing palette information. These arrays are passed by reference via far pointers. The parameter packet contains three parameters; a logical color parameter, a length parameter, and an array address parameter.

5.2.4.2.5.1 Array parameter. The **array** parameter value is a far pointer to a memory block containing an array of palette colors. Each palette color value is a 32-bit structure containing four 1-byte values described as follows:

a. Byte 0, the least significant byte, represents B(lue);

b. Byte 1 represents G(reen);

c. Byte 2 represents R(ed); and

d. Byte 3, the most significant byte, is reserved and must be set to zero. VDI Management returns "ERROR 51" (Parameter value invalid or out of range) if the reserved byte is not zero.

5.2.4.2.5.2 Length and color parameters. The **length** parameter is the number of 32-bit structures in the color array. The **color** parameter is the logical color number to which the first palette color structure is assigned. The second palette color structure is assigned to the next contiguous logical color number; the third structure to the third logical color; and so on up to **length** logical colors. Assume a parameter packet contains logical **color** = 4, **length** = 3, and **array** = 3000:0820. The array memory block is interpreted as:

3000:0820 -- 32-bit word for logical color 4
3000:0824 -- 32-bit word for logical color 5
3000:0828 -- 32-bit word for logical color 6

5.3 Command and parameter summaries. This section includes summary tables of the commands and parameters for both the ASCII and binary interfaces with token numbers for the binary interface. The commands and parameters summarized in this section form the basis of the ICW Portability Practices. Detailed information on the specific service group commands is provided in Appendixes A (sy commands), B (vm commands), C (vd commands), and D (xy-input commands). The digital audio (da) and audio management (am) command sets are assigned for future compatibility, but have not

been defined yet. Appendixes E and F are reserved for the da and am service groups. These command sets will be included after the IMA defines them.

5.3.1 Command names and token numbers. The binary interface uses token numbers instead of command names. These numbers map directly to ASCII command name equivalents in terms of definition and functionality. To assign binary token numbers, each ASCII command name is first divided into a service group prefix, such as "sy" or "vm," and a command word, such as "GetState" or "Init."

5.3.1.1 Service group prefix values. Table 6 lists the prefix value of each service group. As the table shows, service group prefixes have values that are multiples of 1024 (0400H). This has the advantages of leaving room for logically grouping additional commands as the portability practices evolve, and allowing the determination of which service group a token number is in with a single shift right and compare operation.

5.3.1.2 Command word values. Table 7 lists the value for each command word. Commands words are numbered 1–20 in alphabetical order. However, because the numbers must be "cast in stone" for backwards compatibility, new words will be appended to the list and the correspondence of alphabetical order to numeric order will not be maintained as the portability practices evolve. This approach offers the advantages of the ability to determine the command word by simply subtracting the service group value and looking up the word, and the increased efficiency of contiguous numbers in a lookup table.

5.3.1.3 Combined service group prefix and command name values. Table 8 lists complete ASCII command names and their binary token numbers organized by service group. Each token number is the sum of the service-group prefix value and the command word value. For some commands, this offers the advantage of deriving token numbers from different prefixes that use the same command word. For example:

a. Using the C language, a series of #define statements might include:

```
#define SY 1024
#define VM 2048
#define INIT 7
```

Then, the token number for syinit and vdInit could be derived with:

```
#define SYINIT SY + INIT
#define VDINIT VD + INIT
```

b. Table 8 also indicates whether the commands are core or extended. Core commands must be implemented for a given service group to be compliant (MIL-STD-1379). Extended commands are optional and should be considered nonportable unless an application is written to use them if present and handle their absence.

5.3.2 Parameter names and token numbers. Parameter numbers are contiguous starting with one. The majority of parameter token numbers map to ASCII parameter names. However, some binary parameter numbers such as array and length have no ASCII equivalents. In addition:

a. Table 9 lists parameter names and their binary token numbers. Parameters are numbered 1 through 67 in alphabetical order. However, because the numbers must be "cast in stone" for backwards compatibility, new parameters will be appended to the list and the correspondence of alphabetical order to numeric order will not be maintained as the portability practices evolve.

b. Table 9 also indicates whether the parameters are core or extended. Core parameters for a given command must be implemented for compliance (MIL-STD-1379). *Extended parameters are optional and should be considered non-portable* unless an application is written to use them if present and also handle their absence.

5.3.3 System (sy) Commands. System commands (sy) are those that relate to the overall VDI Management system software operation. sy commands are discussed in detail in Appendix A to this handbook.

5.3.4 Visual Management (vm) commands. This series of commands relate to the visual management (vm) of the display screen. vm commands control graphics and video displays, visual signal routing, and video and graphics modes. The vm commands are addressed more fully in Appendix B.

5.3.5 Videodisc (vd) commands. Videodisc (vd) commands are those used to control videodisc players. vd commands are used to initialize, query, and control videodisc players connected to the system. vd commands are discussed further in Appendix C of this handbook.

5.3.6 XY-Input (xy) commands. Appendix D of this handbook contains the detailed information necessary for proper implementation of the xy-input commands. xy commands provide a uniform way to obtain information from these devices and define coordinate spaces. These commands relate to XY-input devices, such as mice, touchscreens and light pens.

5.3.7 Digital audio (da) commands. Appendix E of this handbook is reserved for the da command set. The ICW Portability Practices necessary to support and implement this *service group will be added when requirements are defined.*

5.3.8 Audio management (am) commands. Appendix F of this handbook is reserved for the am command set. The ICW Portability Practices necessary to support and implement this service group will be added when requirements are defined.

5.4 Graphics default positions. To ensure compatibility between the ICW device, and the software interface and command protocols, the active graphics display for a given ICW application should always have the same position relative to the active video and be of the same size. Procedures for determining the size and position of the application graphics relative to the active video are provided in Appendix G.

5.5 Error handling. Error handling protocols are established to support implementation of the ICW Portability Practices. When an application issues a command, VDI Management may not be able to execute the command or return the requested information. When this occurs, some form of error identification is required.

5.5.1 General information. The ASCII interface returns errors as response strings consisting of the word "ERROR" followed by a space and the error number. For example, "ERROR 49" signals that a command included insufficient parameters. The binary interface returns error numbers in the microprocessor's AX register on return from the software interrupt (AX = 0 indicates success).

5.5.1.1 Error codes. Some VDI Management implementations may not use all the error codes. For example, a system that does not use the MS-DOS filing system probably would not use the filing-system error codes. Filing-system error codes are included, however, for use by future digital audio commands. Regardless of current error handling requirements, implementors should try to be complete and should not omit error codes simply for convenience.

5.5.1.2 Error messages. VDI Management implementations may supply textual error messages (see **syErrorMsg** in Appendix A). Although this is not a compliance requirement, if an implementation does support textual error messages, it must use the summary messages (excluding the period) given in Appendix H.

5.5.1.3 Error recovery. VDI Management should try to recover before returning an error response to the application. For example, if a communications error occurs, VDI Management should return an error only after repeated retries have failed. However, VDI Management cannot handle the error of a user forgetting to install VDI Management or the incorrect interface being used by an application. Applications should guard against this by confirming that VDI Management and the proper interface are installed (see 5.1.4.2 and 5.1.5.2).

5.5.2 Error Listings. Error codes and summary messages are provided in Appendix H of this handbook. An explanation of circumstances which should invoke specific error codes is also provided.

5.6 Application programming examples. Appendix I contains several brief programming examples that use the ASCII and binary interfaces. The examples are intended to furnish a starting point for programmers. They are not intended to be overly sophisticated or complete. That would require details beyond the scope of this handbook.

6. NOTES

6.1 Intended use. This handbook is intended to be used in conjunction with MIL-STD-1379 for implementation of the ICW software interface and command requirements established for military training programs. It is designed to aid manufacturers and ICW developers in the implementation of ICW portability practices outlined herein which support application of ICW portability protocols in DoD military training programs.

6.2 Subject term (key word) listing.

Application
Array
ASCII interface
Audio management (am) service group
Binary interface
Command string
Core command
Core parameter
Digital audio (da) service group
Extended command
Extended parameter
ICW Portability Practices
IMA Recommended Practices
Parameter packet
Response string
Service group
System (sy) service group
Token numbers
VDI management
Videodisc (vd) service group
Virtual device interface (VDI)
Visual management (vm) service group
XY-Input (xy) service group

TABLE 1. IBM-compatible graphics modes.

Mode [1] (decimal)	Type	Resolution	Colors	Overlay mode	Adapter [2]
0,1	Text	40 x 25	16	Yes	CGA, EGA, VGA
2,3	Text	80 x 25	16	Yes	CGA, EGA, VGA
4,5	Graphics	320 x 200	4	Yes	CGA, EGA, VGA
6	Graphics	640 x 200	2	Yes	CGA, EGA, VGA
7	Text	80 x 25	Mono	No	EGA, VGA
13	Graphics	320 x 200	16	Yes	EGA, VGA
14	Graphics	640 x 200	16	Yes	EGA, VGA
15	Graphics	640 x 350	Mono	No	EGA, VGA
16	Graphics	640 x 350	16	No	EGA, VGA
17	Graphics	640 x 480	2	Yes	VGA
18	Graphics	640 x 480	16	Yes	VGA
19	Graphics	320 x 200	256	Yes	VGA

[1] Does not include modes exclusive to the IBM PCjr and internal BIOS modes.
[2] CGA = color graphics adapter, EGA = enhanced graphics adapter, and VGA = video graphics array.

TABLE 2. Parameter block layout.

Address	Contents
ES:DI[0]	Parameter 1 token number
ES:DI[4]	Parameter 1 value
ES:DI[8]	Parameter 2 token number
ES:DI[C]	Parameter 2 value
ES:DI[10]	Parameter 3 token number
ES:DI[14]	Parameter 3 value
ES:DI[18]	Parameter 4 token number
ES:DI[1C]	Parameter 4 value
ES:DI[20]	Parameter 5 token number
ES:DI[24]	Parameter 5 value

TABLE 3. Formal syntax for ASCII command and response strings.

<command string>	:=	[<delimiter string>] <command name> [<parameter string>] [<delimiter string>] CR[1]
<response string>	:=	"OK"CR/LF | "ERROR" space <digit string> CR/LF | <result string>CR/LF
<parameter string>	:=	<parameter> | <parameter> <parameter string>
<parameter>	:=	<delimiter string> <parameter id> [<delimiter string> <parameter value>][1]
<parameter id>	:=	<letter string> | <letter string> <digit string>[2]
<command name>	:=	<letter string>[2]
<result string>[3]	:=	<parameter value> | <parameter value> comma <result string>
<parameter value>	:=	<letter string> | <number> | <filename> | <hex string> | <word string> | <printable string>
<delimiter string>	:=	<delimiter> | <delimiter> <delimiter string>[1]
<word string>	:=	<letter string> | <letter string> space <word string>
<printable string>	:=	<printable> | <printable> <printable string>
<letter string>	:=	<letter> | <letter> <letter string>
<number>	:=	[<sign>] <digit string>
<digit string>	:=	<digit> | <digit> <digit string>
<hex string>	:=	<hex digit> | <hex digit> <hex string>
<file name>	:=	<legal file name string for operating system>
<delimiter>	:=	equals | space | HT | comma | LF
<printable>	:=	<printable character on computer used to implement VDI Management, but may not be space, backspace, comma, CR, LF, or HT>
<letter>	:=	"A"|"B"|"C"|"D"|"E"|"F"|"G"|"H"|"I"|"J"|"K"|"L"|"M"|"N"|"O"|"P"|"Q"|"R"|"S"|"T"|"U"|"V"|"W"|"X"|"Y"|"Z"|"a"|"b"|"c"|"d"|"e"|"f"|"g"|"h"|"i"|"j"|"k"|"l"|"m"|"n"|"o"|"p"|q"|"r"|"s"|"t"|"u"|"v"|"w"|"x"|"y"|"z"|
<digit>	:=	"0"|"1"|"2"|"3"|"4"|"5"|"6"|"7"|"8"|"9"
<hex digit>	:=	"0"|"1"|"2"|"3"|"4"|"5"|"6"|"7"|"8"|"9"|"A"|"B"|"C"|"D"|"E"|"F"|"a"|"b"|"c"|"d"|"e"|"f"
<sign>	:=	"+"|"-"

[1] Redundant delimiters in a <delimeter string> are ignored and shall not count toward length limit for <command string>s.

[2] This is not a complete definition. The items on the left can take on a limited range of values.

[3] Alpha characters, if present, in <result string>s shall be limited to upper case.

NOTE: The formal description above omits the 255-character limit for command and response strings, and the equivalency of lower and upper case in command strings.

TABLE 4. ASCII bit field values.

Service Group	Decimal value returned by syGetState
System (sy)	1
Visual Management (vm)	2
Videodisc (vd)	4
XY-input (xy)	8
Digital audio (da) [1]	16
Audio management (am) [1]	32

[1] Currently not defined. Included for future compatibility.

TABLE 5. Binary bit field values.

Service group	Least significant byte value if group is present	
	Binary	Hexadecimal
System (sy)	00000001	01
Visual Management (vm)	00000010	02
Videodisc (vd)	00000100	04
XY-Input (xy)	00001000	08
Digital Audio (da) [1]	00010000	16
Audio Management (am) [1]	00100000	32

[1] Currently not defined. Included for future compatibility.

TABLE 6. Service group prefix values for the binary interface.

Service group prefix	Value (Decimal)	Value (Hexadecimal)
sy	1024	0400
vm	2048	0800
vd	3072	0C00
xy	4096	1000
da [1]	5120	1400
am [1]	6144	1800

[1] *Currently not defined. Included for future compatibility.*

TABLE 7. Command word values for the binary interface.

ASCII Name	Value (Decimal)	Value (Hexadecimal)
CheckError	1	01
ErrorMsg	2	02
Fade	3	03
GetInput	4	04
GetPalette	5	05
GetState	6	06
Init	7	07
PassThru	8	08
Play	9	09
Queue	10	0A
Scan	11	0B
Search	12	0C
Set	13	0D
SetGraphics	14	0E
SetPalette	15	0F
SetTrans	16	10
SetVideo	17	11
Step	18	12
Still	19	13
Stop	20	14

TABLE 8. ASCII command name summary.

ASCII name	Token number (decimal)	Token number (hexadecimal)	Type
sy service group			
syCheckError	1025	0401	Core
syErrorMsg	1026	0402	Extended
syGetState	1030	0406	Core
syInit	1031	0407	Core
syQueue	1034	040A	Core
syStop	1044	0414	Core
vm service group			
vmFade	2051	0803	Core
vmGetPalette	2053	0805	Core
vmGetState	2054	0806	Core
vmInit	2055	0807	Core
vmSetGraphics	2062	080E	Core
vmSetPalette	2063	080F	Core
vmSetTrans	2064	0810	Core
vmSetVideo	2065	0811	Core
vd service group			
vdGetState	3078	0C06	Core
vdInit	3079	0C07	Core
vdPassThru	3080	0C08	Core
vdPlay	3081	0C09	Core
vdScan	3083	0C0B	Core
vdSearch	3084	0C0C	Core
vdSet	3085	0C0D	Core
vdStep	3090	0C12	Core
vdStill	3091	0C13	Core
xy service group			
xyGetInput	4100	1004	Core
xyGetState	4102	1006	Core
xyInit	4103	1007	Core
xySet	4109	100D	Core

TABLE 9. A summary of parameter labels including binary token numbers.

Parameter name	Token number (decimal)	Token number (hex)	Parameter name	Token number (decimal)	Token number (hex)	Parameter name	Token number (decimal)	Token number (hex)
array	1	01	from	24	18	tsources	46	2E
audio1	2	02	g	25	19	time	47	2F
audio2	3	03	glevel	26	1A	to	48	30
b	4	04	gmode	27	1B	transcolor	49	31
buttons	5	05	horzpix	28	1C	vertpix	50	32
cdisplay [1]	6	06	idxdisplay	29	1D	video	51	33
chapter [1]	7	07	ivver	30	1E	vlevel	52	34
clear	8	08	length	31	1F	vmode	53	35
color	9	09	logcolors	32	20	wait	54	36
command	10	0A	mfgname	33	21	width [3]	55	37
cursor [2]	11	0B	mfgver	34	22	xmax	56	38
defdevice	12	0C	motion	35	23	xmaxclip	57	39
defsource	13	0D	physcolors	36	24	xmin	58	3A
device	14	0E	pmsg	37	25	xminclip	59	3B
direction	15	0F	r	38	26	xoffset	60	3C
disctype	16	10	remote [1]	39	27	xpos	61	3D
dlevel	17	11	speed	40	28	ymax	62	3E
door [1]	18	12	spin	41	29	ymaxclip	63	3F
emulation	19	13	state	42	2A	ymin	64	40
enable	20	14	support	43	2B	yminclip	65	41
errno	21	15	tbuttons	44	2C	yoffset	66	42
execute	22	16	tdevices	45	2D	ypos	67	43
frame	23	17						

[1] Currently used only as an extended parameter in the vd service group.
[2] Currently used only as an extended parameter in the xy service group.
[3] Currently used only as an extended parameter in the vm service group.

APPENDIX A

SYSTEM (sy) COMMANDS FOR ICW PORTABILITY

10. SCOPE

10.1 Scope. This appendix describes commands that relate to overall VDI software operation. These commands initialize the basic ICW system (but not other service groups), obtain specific information about system software and its configuration, retrieve error information, queue commands for subsequent execution, and free resources when the VDI software is not in use. Table A-1 lists the commands covered in this section, their token numbers, and their types.

10.2 Application guidance. This appendix is written to support implementation of the software interface and command requirements prescribed by MIL-STD-1379, Appendix D that relate to overall software operation in ICW applications. Each of the sy commands listed in Table A-1 is described in a separate section of this appendix, beginning with Section 30, syCheckError command.

10.2.1 Terms, abbreviations, and acronyms used in this appendix. Key terms, abbreviations, and acronyms used in this appendix are defined as specified in Section 3 of the basic handbook.

20. APPLICABLE DOCUMENTS.

20.1 Government documents.

20.1.1 Specifications, standards, and handbooks. The following specifications, standards, and handbooks form a part of this appendix to the extent specified herein.

STANDARD

MILITARY

MIL-STD-1379 Military Training Programs

(Unless otherwise specified, copies of military specifications, standards and handbooks are available from the Standardization Documents Order Desk, Building 4D, 700 Robbins Avenue, Philadelphia, PA 19111-5094.)

20.2 Non-Government publications. The following documents form a part of this document to the extent specified herein.

APPENDIX A

INTERACTIVE MULTIMEDIA ASSOCIATION (IMA)

Recommended Practices for Multimedia Portability, (MS-DOS Based Systems), Release R 1.1

INTERACTIVE VIDEO INDUSTRY ASSOCIATION (IVIA) (now the IMA)

Recommended Practices for Interactive Video Portability (Working Document), Release R 1.0

(Application for copies of the IMA/IVIA documents should be addressed to the Interactive Multimedia Association (IMA), 3 Church Circle, Suite 800, Annapolis, MD 21401-1933.)

(Non-Government standards and other publications are normally available from the organizations that prepare or distribute the documents. These documents also may be available in or through libraries or other informational services.)

30. syCheckError COMMAND

30.1 syCheckError command summary. **syCheckError** is a core command having a binary token number of 1025 decimal. **syCheckError** returns the number of the last error detected by VDI Management, if present, and the command and logical device number that caused the error. **syCheckError** then clears this error information.

30.2 Description. VDI Management may detect errors that do not occur in immediate response to application commands. Such errors may occur, for example, after player motion commands, fade and dissolve commands, queued commands, and others that execute over time have been accepted. **syCheckError** is provided to detect these types of errors, although it will return the last error regardless of the error's cause. For example:

a. Assume a player accepts a valid **vdPlay** command without a **wait** modifier (see Appendix C). VDI Management will return success immediately ("OK" for the ASCII interface, AX = 0 for the binary interface). If the player then fails during the specified motion sequence, an error state exists. **syCheckError** determines if such a situation has arisen and, if so, returns the error.

b. Similarly, if a fade is successfully initiated and subsequently fails, an unreported error results.

c. A queued command may cause an error although **syQueue execute** was successful. **syCheckError** returns the error resulting from the queued command. This is the only way to detect an error resulting from a queued command.

d. If an ASCII command string contains multiple commands (see 5.2.3.1.4) and a command other than the last command in the string causes as error, **syCheckError** is the only way to detect that error.

30.3 Implementation. While actual implementations may vary in the way they store and translate the information needed by **syCheckError**, in theory VDI Management maintains four buffers for returning error information: the response buffer, the check error buffer, the check command buffer, and the check device buffer. The contents of these buffers (or the translations thereof) depend on which interface is used.

30.3.1 ASCII interface. Buffer contents using the ASCII interface are as follows:

a. The response buffer contains the return string for the current command, in this case **syCheckError**.

b. The check errno buffer contains the error number as "n..." of the most recent error caused by any command. If no error has occurred or the buffer has been cleared, the buffer content is zero (0).

c. The check command buffer contains the name of the command that caused the error. If no error has occurred or the buffer has been cleared, the contents are "OK".

d. The check device buffer contains the logical device number of the device that caused the last error, or "-1" if no error has occurred or the buffer has been cleared. (Note that "-1" is returned because zero is a valid logical device number.)

30.3.2 Binary interface. When the application uses the binary interface, buffer contents should be the following:

a. The response buffer contains the value to be returned in the AX register for the current command, in this case **syCheckError**.

b. The check errno buffer contains the error number of the most recent error caused by any command. The buffer contains zero (0) if no error has occurred or the buffer has been cleared.

c. The check command buffer contains the token number of the command that caused the error, or zero if no error has occurred or the buffer has been cleared.

d. The check device buffer contains the logical device number of the device that caused the last error, or "-1" if no error has occurred or the buffer has been cleared.

30.4 Command parameters. The syCheckError command has three defined parameters: **command**, **device**, and **errno**. At least one of these parameters must be used with **syCheckError** or an error is returned by VDI management. All are core parameters that must be supported by compliant VDI Management implementations. Table A-2 lists ASCII and binary parameter information for **syCheckError**.

30.4.1 Command parameter. The command parameter requests the command that caused the last error, if present. Issuing syCheckError with command clears the error number in the check errno buffer, the command name or token in the check command buffer, and the device number in the check device buffer.

30.4.2 Device parameter. The device parameter requests the logical device number of the device associated with the command that caused the last detected error. Issuing **syCheckError device** clears the error number in the check errno buffer, the command name or token number in the check command buffer, and the device number in the check device buffer. In addition:

a. The device number returned by the device parameter may not be the actual device that caused the error, although it usually will be.

b. For example, the command **vdSet device=1,video=0**, which turns off the video channel for player 1, may actually be implemented on the overlay board. If this command fails, **syCheckError device** returns "1" -- the device number associated with the failed command. It would not return the number of the overlay board which is undefined under the current portability practices.

30.4.3 Errno parameter. The **errno** parameter requests the error number of the last error detected. Issuing **syCheckError** with **errno** clears the command name or token in the check command buffer, the logical device number in the check device buffer, as well as the error number in the check errno buffer.

30.5 Implementation notes. When using **syCheckError**, note the following:

a. **syCheckError** does not queue errors. For example, if a player motion command resulting in an unreported error is followed by a fade command resulting in an unreported error, the unreported error for the player motion command is lost when the fade command error is stored.

b. Only **syCheckError** and **syInit** can clear the check errno, check device, and check command buffers. However, **syInit** also reinitializes VDI Management, clearing the queue, canceling any pending commands, and setting up interrupts. *Therefore, it should not be used simply to clear an error state.*

c. If **syCheckError** itself causes an error because, for example, it is issued with an invalid parameter, it does not clear the check buffers. Instead, VDI Management loads the check buffers with the same information it would load for any other command causing an error. A subsequent correct call to **syCheckError** returns and clears this information.

d. Trying to queue **syCheckError** causes error 177 (Command cannot be queued) at the time of the attempt.

30.6 Return values.

30.6.1 *ASCII returns*. *On success*, the return is: For the **command** parameter, the name of command causing the last error as an upper-case alpha string, or "OK" if no error exists. For the **device** parameter, the logical device number, "0" through "15" that caused the error, or "-1" if no error. For the **errno** parameter, the last error number as "n...", or "0" if there is no error. On failure, the return is "ERROR n...".

30.6.2 Binary returns. On success, the return is AX = 0. Value associated with the **command** parameter is the token number of the command that caused the error, or 0 if no error is detected. Value associated with the **device** parameter is the logical device number of the device that caused the error, or "-1" if no error. Value associated with the errno

parameter is the last error number as a 32-bit integer, or 0 if there is no error. On failure, the return is AX = error number. Any return values in the parameter block addressed by ES:DI are undefined and should be ignored.

30.7 Related commands. The **syErrorMsg**, **syInit**, and **syQueue** command requirements should also be reviewed in relation to implementing the **syCheckError** command.

30.8 Examples. The following are ASCII and binary examples of the **syCheckError** command.

30.8.1 ASCII.

Pass non-**syCheckError** command that causes an error	**syErrorMsg errno** (returns) "ERROR 53"	; Missing ; parameter value
Now query for last command in error.	**syCheckError errno, command** (returns) "SYERRORMSG"	; clears all last ; error ; information
Re-query after clearing	**syCheckError command errno** (returns) "OK,0"	; note that **errno** ; is also ; cleared.

30.8.2 Binary.

Query for last error and related command token	AX	1025	; **syCheckError** decimal ID
	BX	2	; number of parameters ; (required for return)
	ES:DI[0]	21	; **errno** decimal ID
	ES:DI[4]	Any value	; place holder for value on ; return
	ES:DI[8]	10	; **command** decimal ID
	ES:DI[C]	Any value	; place holder for value ; on return
After return	AX	0	; returns 0 if successful ; (non-zero if not)
	ES:DI[4]	error no.	; number of most recent error, ; if present (zero if no

		; error, undefined if AX ≠ 0)
ES:DI[C]	token no.	; token number of command
		; causing error (zero if no
		; error, undefined if AX ≠ 0)

40. syErrorMsg COMMAND

40.1 syErrorMsg command summary. **syErrorMsg** is an extended command having a binary token number of 1026 decimal. **syErrorMsg** returns an upper case ASCII string of up to 255 characters that describes the specified error number.

40.2 Description. When an ASCII interface command causes an error, the error is reported as "ERROR n..." where "n..." is the error number expressed in ASCII digits. The binary interface returns the error number in the AX register. Applications can use the error number with **syErrorMsg** to request the error description. VDI Management developers may opt to keep error descriptions in memory or a separate file. However, the binary interface allocates a 256-byte buffer for the message.

40.3 Command parameters. The syErrorMsg command has two defined parameters: **errno**, and **pmsg**. Both are core parameters that must be supported by compliant VDI Management implementations. Table A-3 lists ASCII and binary parameter information for **syErrorMsg**.

40.3.1 Errno parameter. The **errno** parameter specifies the error number for which a descriptive string will be returned.

40.3.2 Pmsg parameter. The **pmsg** parameter for the binary interface returns a pointer to the location of the descriptive string. The string can consist of up to 255 upper case characters plus a terminal null character.

40.4 Implementation notes. When using **syErrorMsg**, note the following:

a. Applications that implement **syErrorMsg** should conform to the error handling and error message requirements of MIL-STD-1379. Handbook paragraph 5.5 addresses these requirements. Appendix H lists strings for specific error numbers and messages.

b. Trying to queue **syErrorMsg** causes error 177 (Command cannot be queued) at the time of the attempt.

40.5 Return values.

40.5.1 ASCII returns. On success, the return is an error description string. On failure, the return is "ERROR n...".

40.5.2 Binary returns. On success, the return is AX = 0. Value associated with the **pmsg** parameter is a 32-bit pointer to a null-terminated upper case error description. The description is a string of up to 255 characters plus the terminal null character, excluding a terminal period. Value associated with the **errno** parameter is unchanged. On failure, the

return is AX = error number. Any return values in the parameter block addressed by ES:DI are undefined and should be ignored.

40.6 Related commands. The syCheckError and syQueue command requirements should also be reviewed in relation to implementing the syErrorMsg command.

40.7 Examples. The following are ASCII and binary examples of the syErrorMsg command.

40.7.1 ASCII.

Pass an invalid parameter	syErrorMsg errno=55 (returns) "ERROR 48"
Get string describing error 48	syErrorMsg errno=52 (returns) "PARAMETER INVALID FOR THIS COMMAND"
Pass insufficient parameters	syErrorMsg (returns) "ERROR 49"
Get string describing error 49	syErrorMsg errno=49 (returns) "INSUFFICIENT PARAMETERS"

40.7.2 Binary.

Get string describing error 176	AX	1026	; syErrorMsg decimal ID
	BX	2	; number of packets
	ES:DI[0]	21	; errno decimal ID
	ES:DI[4]	176	; decimal error number
	ES:DI[8]	37	; pmsg decimal ID
	ES:DI[C]	any value	; place holder for value on return
After return	AX	0	; returns 0 if successful (non-zero ; if not)
	ES:DI[C]	pointer	; pointer to message string

50. syGetState COMMAND

50.1 syGetState command summary. syGetState is a core command having a binary token number of 1030 decimal. syGetState returns the service groups for which VDI Management was configured at installation, the version of the IMA Recommended Practices/DoD ICW Portability Practices supported, and VDI Management manufacturer name and version information.

50.2 Command parameters. The syGetState command has four defined parameters: ivver, mfgname, mfgver, and support. All are core parameters that must be supported by compliant VDI Management implementations. Table A-4 lists ASCII and binary parameter information for syGetState.

50.2.1 Ivver parameter. The ivver parameter returns the version number of the IMA Recommended Practices/DoD ICW Portability Practices with which the VDI Management software complies. This number is the same as the document release number times 1000. For example, ivver would return Release 1.1 of the IMA Recommended Practices as 1100. ICW applications can use this number to determine compatibility with VDI Management implementations. An application that requires a given IMA/DoD version number will also be compatible with any higher version number.

CAUTION

> This forward compatibility does not include applications based on the Interactive Video Industry Association (IVIA) (now the IMA) Recommended Practices for Interactive Video Portability (Working Document), Release R 1.0. Most applications written for IVIA Release R 1.0 will not run on IMA Release R 1.1 VDI Management implementations.

50.2.2 Mfgname and mfgver parameters. The mfgname and mfgver parameters return the VDI Management manufacturer's name and software version number respectively. This information is required to confirm compliance with the recommended practices and, for software maintenance purposes, to obtain the manufacturer and version number for technical support. An application may also use this information to determine if a particular implementation that provides extended commands is present. In addition:

a. Because typical version numbers often contain decimal fractions, mfgver returns an integer representation of the actual version number times 1000. For example, a version number of 1.1 is returned as 1100.

b. Mfgname returns a string of up to eight printable characters. The string cannot include white space (ASCII 20H, 09H), backspace (ASCII 08H), comma (ASCII 2CH), CR (ASCII 0DH), or LF (ASCII 0AH). The binary interface returns a null-

terminated string. Therefore, a total length may be nine characters including the terminal NULL.

50.2.3 Support parameter. The **support** parameter returns a bit field or an ASCII representation thereof that specifies service groups for which VDI Management was *configured during software installation*. An application should typically issue **syGetState support** immediately after **syInit** to find out if software support exists for required service groups. Subsequent to **syInit**:

a. Typically, an application will then issue the specific **xxInit** command for each represented service group that the application will use. The **xxInit** commands for individual service groups initialize software support for devices within the group and verify communications with the requisite hardware.

b. Table A-5 shows return values for each service group after a **syGetState support**. The actual value returned is the sum of the listed values for all installed service groups. For example, a binary status return of 00000111B means that system, visual management, and videodisc are supported; but XY-input, digital audio, and audio management are not. An ASCII return value of "7" means the same.

50.2.4 Parameters resulting in errors. If a parameter causes an error, syGetState returns immediately with an error message. The command does not return partial responses for other parameters that do not cause errors.

50.3 Implementation notes. When using **syGetState**, note the following:

a. Values for mfgname and mfgver are not under IMA control. In the future, **mfgname** may be required to be unique and registered with the IMA.

b. Trying to queue **syGetState** causes error 177 (Command cannot be queued) at the time of the attempt.

50.4 Return values.

50.4.1 ASCII returns. On success, the return is a comma-separated string with response for each specified parameter as described in 50.2. On failure, the return is "ERROR n...".

50.4.2 Binary returns. On success, the return is AX = 0. Value associated with **ivver** is a 32-bit integer as described in 50.2.1. Values associated with **mfgname** parameter is a pointer to a null-terminated string as described in 50.2.2. Value associated with the **mfgver** parameter is a 32-bit integer as described in 50.2.3. Value associated with ***support*** *parameter* is a 32-bit field as described above. On failure, the return is AX =

error number. Any return values in the parameter block addressed by ES:DI are undefined and should be ignored.

50.5 Related commands. The **syQueue**, **vdGetState**, **vdInit**, **vmGetState**, **vmInit**, **xyGetState**, and **xyInit** command requirements should also be reviewed in relation to implementing the **syGetState** command.

50.6 Examples. The following are ASCII and binary examples of the **syGetState** command.

50.6.1 ASCII.

Get services *supported by* installed system	**syGetState support** *(returns)* "15"	; system, visual management, ; videodisc, and XY ; commands are supported
Get version number of IMA/DoD practices	**syGetState ivver** (returns) "1100"	; conforms with IMA/DoD ; recommended practices ; version R 1.1
Get manufacturer and version	***syGetState mfgname,mfgver*** (returns) "IVMAKER,1300"	; VDI Management written ; by IVMAKER, version 1.3

50.6.2 Binary.

Get services supported by installed system	AX	1030	; **syGetState** decimal ID.
	BX	1	; number of packets.
	ES:DI[0]	43	; **support** decimal ID.
	ES:DI[4]	Any value	; *no associated value.*
After return	AX	0	; returns 0 if successful ; (non-zero if not)
	ES:DI[4]	Bit field	; **support** parameter bit ; field

60. syInit COMMAND

60.1 syInit command summary. **syInit** is a core command having a binary token number of 1031 decimal. **syInit** initializes VDI Management and the **sy** service group, and confirms communications between VDI Management and the application.

60.2 Description. The specific actions taken by syInit are highly implementation dependent. However, regardless of the implementation, **syInit** does the minimum required to prepare the system for other VDI Management commands. It does not replace the initialization commands for other service groups. For example, **syInit** does not verify communications with a videodisc player or change the video display. However, it may need to attach proper interrupts to proper ports, set proper software interrupts, disable non-ICW operating modes, and do other basic start-up chores. **syInit** also does the following specific initialization tasks:

a. Sets the default logical device or logical source for all service groups to zero.

b. Clears the error buffers used by **syCheckError**.

c. Issues **syQueue clear,state=0** to clear the command queue and turn it off.

60.3 Command parameters. The syInit command has no command parameters.

60.4 Implementation notes. When using **syInit**, note the following:

a. Application programs should make no assumptions about the state of the ICWTS or the presence of service groups after **syInit**. To determine which service groups are present and to enable them, an application should:

   (1) Issue **syInit**.

   (2) Issue **syGetState support** to determine which services are present.

   (3) Issue the initialization commands for the service groups to be used by the application.

   (4) If required, initialize non-ICW devices and allocate additional memory. At the developer's discretion, these tasks may be done either before or after issuing **syInit** and other VDI Management initialization commands.

b. Because the specific actions taken by **syInit** are implementation dependent and affect the state of VDI Management, if an application reissues **syInit** after the start-up sequence given above, it should also repeat steps a(3) and a(4) above to ensure that the system is in a known state.

60.5 Return values.

60.5.1 ASCII returns. On success, the return is "OK". On failure, the return is "ERROR n...".

60.5.2 Binary returns. On success, the return is AX = 0. On failure, the return is AX = error number.

60.6 Related commands. The syCheckError, syGetState, syQueue, vdInit, vmInit, and xyInit command requirements should also be reviewed in relation to implementing the syInit command.

60.7 Examples. The following are ASCII and binary examples of the syInit command.

60.7.1 ASCII.

Initialize VDI Management	syInit (returns) "OK"

60.7.2 Binary.

Initialize VDI Management	AX	1031	; syInit decimal ID
	BX	0	; no parameters
After return	AX	0	; returns 0 if successful ; (non-zero if not)

70. syQueue COMMAND

70.1 syQueue command summary. syQueue is a core command having a binary token number of 1034 decimal. syQueue manages a fixed-length internal queue with exactly 10 slots for storing up to 10 commands. The queue can be turned on and off, cleared, and executed.

70.2 Description. syQueue stores commands in an internal queue for later execution. It may be used to collect commands that have critical timing requirements and should be executed together. One example is a set of changes that should occur during a vertical blanking interval to avoid screen disturbances, such as changing the palette and setting a transparent color. Queued commands are always executed in the order in which they were queued and, if possible, adjacent commands are executed in the same vertical interval.

70.3 Command parameters. The syQueue command has three defined parameters: clear, execute, and state. All are core parameters that must be supported by compliant VDI Management implementations. Table A-6 lists ASCII and binary parameter information for syQueue.

70.3.1 Clear parameter. The clear parameter clears the queue of all commands without executing them. Clear does not change the queue's state (on or off). If the queue is on when cleared, subsequent commands are accumulated until the queue is explicitly turned off. Clearing an empty queue has no effect and is not an error.

70.3.2 Execute parameter. The execute parameter instructs VDI Management to execute all commands in the queue as quickly as possible. syQueue execute does not clear the queue or affect the queue's state (on or off). Additionally:

a. A queue that has been turned off remains executable.

b. Executing an empty queue has no effect and is not an error.

70.3.3 State parameter. The state parameter turns the queue on and off, as follows:

a. State = 1 instructs VDI Management to store up to 10 commands for later execution. If more than 10 commands are issued while the queue is on, the extra commands cause error 176 (Queue full), and those commands already in the queue are left intact.

b. State = 0 instructs VDI Management to resume immediate execution of commands without storing them in the queue. Commands already in the queue remain unchanged and unexecuted.

c. Turning on a queue that is already on, or turning it off when it is already off has no effect and is not an error.

70.3.4 Combining parameters. The **execute** parameter always takes precedence. To execute commands and clear the queue, use **syQueue clear execute** or **syQueue execute clear**. Because **execute** has the higher priority, **syQueue** acts on **execute** first in both examples. Similarly, **syQueue state = 1 execute** and **syQueue state = 0 execute** work as expected, executing the queue then turning it on or off, respectively.

70.4 Unqueueable commands. **syQueue** cannot queue the commands listed in Table A-7 either because requested information would be lost after **syQueue execute**, or because their behavior could disrupt the queue or the execution of subsequent queued commands. For example:

a. If an application tries to queue an unqueueable command except **syQueue**, which executes immediately, the illegal command returns error 177 (Command cannot be queued) immediately. This error and error 176 (Queue full) are the only errors that can be returned while the queue is on.

b. **syQueue** ignores the unqueueable commands listed in Table A-7, except to return error 177. The unqueueable command is not queued and does not affect the status of the queue. Similarly, if the queue is full, **syQueue** ignores all attempts to queue additional commands, except to return error 176 (see 70.6).

70.5 Queued commands causing errors. When a queued command results in an error, the error is not detected until syQueue execute. When the error is detected, **syQueue** returns immediately without executing any remaining commands in the queue. However, **syQueue** does not return an error. Therefore, it is good practice to issue - **syCheckError** immediately after **syQueue execute** to determine if an error occurred during queue execution. This is the only systematic way to detect such errors.

70.6 Implementation notes. When using **syQueue**, note the following:

a. **syQueue** always executes immediately and cannot be queued.

b. With the binary interface, **syQueue** queues parameter packets by value. That is, VDI Management stores the contents of packets internally until it receives an **syQueue clear** or the 10-command limit is reached. Any changes to values in packets that VDI Management would normally make, such as changing requested player speeds to actual speeds after rounding, if necessary, are lost. Such values can be retrieved with GetState commands.

c. If **vmSetPalette** is queued, the pointer information in the command's parameter block is stored with the queue (see Appendix B). However, the information in the

palette array is not stored, but is read when the queue is executed. Therefore, if an application changes the contents of the palette array between issuing and executing the queued command, the changed array will be used. An application should not de-allocate the memory for the palette array before clearing the queue. Doing so could load invalid palette information.

d. Trying to queue more than 10 queueable commands causes error 176 (Queue full). Trying to queue any unqueueable command except **syQueue**, which executes immediately, causes error 177 (Command cannot be queued). Error 177 takes precedence over error 176.

e. An unknown command can be queued and will not return error 2 (Unknown command) until the queue is executed.

f. **SyInit** always executes immediately and effectively issues **syQueue clear, state=0** regardless of the on/off status of the queue.

70.7 Return values.

70.7.1 ASCII returns. On success, the return is "OK". On failure, the return is "ERROR n...".

70.7.2 Binary returns. On success, the return is AX = 0. On failure, the return is AX = error number.

70.8 Related commands. The vmSetPalette command requirements should also be reviewed in relation to implementing the **syQueue** command.

70.9 Examples. The following are ASCII and binary examples of the **syQueue** command.

70.9.1 ASCII.

```
Turn on queue and      syQueue state=1
store commands          (returns) "OK"
                       (command 1)
                        (returns) "OK"
                       (command 2)
                        (returns) "OK"
                       (command 3)
                        (returns) "OK"
                       (command 4)
                        (returns) "OK"          ; commands 1-4 are stored and
                                                ; not executed.
```

Turn off queue and execute new commands immediately	**syQueue state=0** (returns)"OK" (command 5) (returns) "OK" (command 6) (returns) "OK" (command 7) (returns) "OK"	; subsequent commands are not ; queued ; commands 5–7 are each ; executed immediately
Execute queue, then unqueued commands	**syQueue execute** (returns) "OK" (command 8) (returns) "OK" (command 9) (returns) "OK"	; commands 1-4 are rapidly ; executed from the queue. ; commands 8 and 9 are each ; executed immediately.
Re-execute queue and clear.	syQueue execute,clear (returns) "OK"	; commands 1-4 are rapidly ; executed and cleared.

70.9.2 Binary.

Turn on syQueue	AX	1034	; syQueue decimal ID
	BX	1	; number of parameters
	ES:DI[0]	42	; state decimal ID
	ES:DI[4]	1	; value for "on"
After return	AX	0	; Returns 0 if successful ; (non-zero if not)
Clear queue and stop accumulating commands	AX	1034	; syQueue decimal ID
	BX	2	; number of parameters
	ES:DI[0]	8	; clear decimal ID
	ES:DI[4]	Any value	; no associated value
	ES:DI[8]	42	; state decimal ID
	ES:DI[C]	0	; value of "off"
After return	AX	0	; Returns zero if ; successful (non-zero if ; not)

80. syStop COMMAND

80.1 syStop command summary. syStop is a core command having a binary token number of 1044 decimal. **syStop** frees all possible resources used by the interfaces and VDI Management to make those resources available for non-ICW use.

80.2 Description. **syStop** reduces the interfaces and VDI Management to their minimum possible configurations without actually unloading the VDI software. The command frees resources such as file handles and interrupts for use by non-ICW applications. The command's actions are highly implementation and configuration dependent, and:

a. **syStop** does not change the graphics mode. Therefore, applications must handle the mode separately after exit. However, if VDI Management turned mode trapping on, which it typically would, **syStop** turns it off.

b. After an **syStop**, all VDI commands are undefined except syInit. Upon resumption of ICW activities, a **syInit** must be issued before any other ICW command.

80.3 Command parameters. The syStop command has no parameters.

80.4 Implementation notes. When using **syStop**, note that trying to queue **syStop** causes error 177 (Command cannot be queued) at the time of the attempt.

80.5 Return values.

80.5.1 ASCII returns. On success, the return is: "OK". On failure, the return is "ERROR n...".

80.5.2 Binary returns. On success, the return is AX = 0. On failure, the return is AX = error number.

80.6 Related commands. The syInit and syQueue command requirements should also be reviewed in relation to implementing the syStop command.

80.7 Examples. The following are ASCII and binary examples of the **syStop** command.

80.7.1 ASCII.

Place VDI Management in inactive state	**syStop** (returns) "OK"

80.7.2 Binary.

Place VDI Management in inactive state	AX	1044	; syStop decimal ID
	BX	0	; no parameters
After return	AX	0	; returns 0 if successful ; (non-zero if not)

TABLE A-1. System (sy) commands summary.

ASCII Command name[1]	Binary Interface token number (Decimal)	Type[2]
syCheckError	1025	Core
syErrorMsg	1026	Extended
syGetState	1030	Core
syInit	1031	Core
syQueue	1034	Core
syStop	1044	Core

[1] Upper or lower case for command names is not significant.

[2] Compliant implementations must support "Core" commands. Supporting "Extended" commands is optional, and these commands should be considered non-portable unless properly handled when absent.

TABLE A-2. syCheckError parameters.

ASCII Parameters						
Parameter[1]	**Core or extended**	**Associated calling value**	**Type as ASCII**	**Associated return value**	**Type as ASCII**	**Default if parameter not used**
command	Core	None	N/A	Command name that caused error, or "OK" if no error	Text	No action
device	Core	None	N/A	Logical device number that caused error, 0-15, or -1 if no error	Integer	No action
errno	Core	None	N/A	Last detected error number or zero if no error	Integer	No action
Binary Parameters						
Parameter[1]	**Core or extended**	**Token number (decimal)**	**Type**	**Associated calling value**	**Associated return value**	**Default if parameter not used**
command	Core	10	Integer	Any value	Command token that caused error or zero if no error	No action
device	Core	14	Integer	Any value	Logical device number that caused error, 0-15, or -1 if no error	No action
errno	Core	21	Integer	Any value	Last detected error number or zero if no error	No action

[1] At least one parameter is required or an error is returned.

TABLE A-3. syErrorMsg parameters.

ASCII Parameters						
Parameter	**Core or extended**	**Associated calling value**	**Type as ASCII**	**Associated return value**	**Type as ASCII**	**Default if parameter not used**
errno[1]	Core	Error number	Integer	Error description string	Text	Causes error
Binary Parameters						
Parameter	**Core or extended**	**Token number (decimal)**	**Type**	**Associated calling value**	**Associated return value**	**Default if parameter not used**
errno[2]	Core	21	Integer	Error number	None	Causes error
pmsg[2]	Core	37	Pointer	Any value	Pointer to error description string	Causes error

[1] **Errno** must be specified or an error is returned.

[2] Both the **errno** and **pmsg** parameters must be specified or an error is returned.

TABLE A-4. syGetState parameters.

ASCII Parameters						
Parameter[1]	Core or extended	Associated calling value	Type as ASCII	Associated return value	Type as ASCII	Default if parameter not used
ivver	Core	None	N/A	Recommended practices version number x 1000	Integer	No action
mfgname	Core	None	N/A	Manufacturer's name[2]	Text	No action
mfgver	Core	None	N/A	Manufacturer's version number x 1000	Integer	No action
support	Core	None	N/A	Value of bit field for installed service groups	Integer	No action
Binary Parameters						
Parameter[1]	Core or extended	Token number (decimal)	Type	Associated calling value	Associated return value	Default if parameter not used
ivver	Core	30	Integer	Any value	Recommended practices version number x 1000	No action
mfgname	Core	33	Pointer	Any value	Pointer to manufacturer's name string[3]	No action
mfgver	Core	34	Integer	Any value	Manufacturer's version number x 1000	No action
support	Core	43	Bit field	Any value	Bit field of installed service groups	No action

[1] At least one parameter is required or an error is returned.

[2] Eight characters max. Restricted to printable characters and cannot include white space (ASCII 20H, 09H), backspace (ASCII 08H), comma (ASCII 2CH), CR (ASCII 0DH), or LF (ASCII 0AH).

[3] Eight characters max plus terminal NULL. Restricted to printable characters and cannot include white space (ASCII 20H, 09H), backspace (ASCII 08H), comma (ASCII 2CH), CR (ASCII 0DH), or LF (ASCII 0AH).

TABLE A-5. syGetState support return values.

Service Group	Binary interface return value (low byte)	ASCII interface return value
System (sy)	00000001	01
Visual management (vm)	00000010	02
Videodisc (vd)	00000100	04
XY input (xy)	00001000	08
Digital audio (da)[1]	00010000	16
Audio management (am)[1]	00100000	32

[1] Currently not defined. Included for future compatibility.

TABLE A-6. syQueue parameters.

ASCII Parameters						
Parameter[1]	Core or extended	Associated calling value	Type as ASCII	Associated return value	Type as ASCII	Default if parameter not used
clear	Core	None	N/A	None	N/A	No action
execute	Core	None	N/A	None	N/A	No action
state	Core	1 (on) | 0 (off)	Integer	None	N/A	No action
Binary Parameters						
Parameter[1]	Core or extended	Token number (decimal)	Type	Associated calling value	Associated return value	Default if parameter not used
clear	Core	8	N/A	Any value	None	No action
execute	Core	22	N/A	Any value	None	No action
state	Core	42	Integer	1 (on) | 0 (off)	None	No action

[1] At least one parameter is required or an error is returned.

TABLE A-7. Unqueueable commands.

syCheckError	vdGetState	xyGetInput
syErrorMsg	vdInit	xyGetState
syGetState	vmGetPalette	xyInit
syQueue	vmGetState	
syStop	vmInit	

APPENDIX B

VISUAL-MANAGEMENT (vm) COMMANDS FOR ICW PORTABILITY

10. SCOPE

10.1 Scope. This appendix describes commands that relate to the visual management of the display screen. The commands in this section control the graphics display, video display, visual signal routing, video modes, and graphics modes. Table B-1 lists the commands covered in this summary, their token numbers, and their types.

10.2 Application guidance. This appendix is written to support implementation of the software interface and command requirements prescribed by MIL-STD-1379, Appendix D that relate to overall visual-management software operation in ICW applications. Each of the commands listed in Table B-1 is described in a separate section of this appendix, beginning with Section 40, vmFade command.

10.2.1 Terms, abbreviations, and acronyms used in this appendix. Key terms, abbreviations, and acronyms used in this appendix are defined as specified in Section 3 of the basic handbook.

20. APPLICABLE DOCUMENTS.

20.1 Government documents.

20.1.1 Specifications, standards, and handbooks. The following specifications, standards, and handbooks form a part of this appendix to the extent specified herein.

STANDARD

MILITARY

MIL-STD-1379 Military Training Programs

(Unless otherwise specified, copies of military specifications, standards and handbooks are available from the Standardization Documents Order Desk, Building 4D, 700 Robbins Avenue, Philadelphia, PA 19111-5094.)

APPENDIX B

30. GENERAL GUIDANCE

30.1 Terms of reference. Figure B-1 shows a basic conceptual definition of a video overlay subsystem. It is intended to convey the overlay card's functionality, but not the hardware implementation. The functionality applies to overlay boards that are either based primarily on graphics synchronized to a video signal or have the video corrected to match the graphics. This conceptual definition assumes that:

a. The definition includes two sources: plane 0 (video) from an external source such as a videodisc player, and plane 1 (graphics) from the computer. Each source has an associated level control or fader.

b. The palette does logical-to-physical color conversion and controls transparency. It can be described in terms of logical colors (number of colors that can be simultaneously displayed) and physical colors (palette size).

c. The keyer is responsible for selecting either plane 0 or plane 1 at a pixel rate for output to the display.

d. The dissolve unit:

(1) In its simplest form, is a switch between video only and graphics over video (a graphics ON/OFF capability).

(2) At the next level of complexity, supports plane 0<-->plane 1 cross-fades.

(3) In future systems, may become a pixel-rate translucency setting controlled by a palette extension (not shown).

30.2 General information and assumptions. The general information and assumptions described in this section are used in the definition of the visual-management (vm) commands. The material below is based on Intel 80X86 microprocessor architecture, MS-DOS compatible operating systems, and standard IBM-compatible graphics modes.

30.2.1 Overlayable graphics modes. Paragraph 4.3.3 describes IBM-compatible graphics modes as they should be returned by BIOS interrupt 10H, service 0FH; including whether the modes are text or graphics, mode resolutions, the adapters that support them, and whether or not they are overlayable on video. In addition:

a. Compliance with MIL-STD-1379 requires that graphics modes 0 through 3 be overlayable when the selected video mode is NTSC or PAL.

b. When the video mode is not set to "native", modes zero through three are restricted to 200 lines, regardless of the actual number of line that would normally be displayed by a given monitor adapter. Refer to the **vmSetGraphics** and **vmSetVideo** commands described later in this appendix.

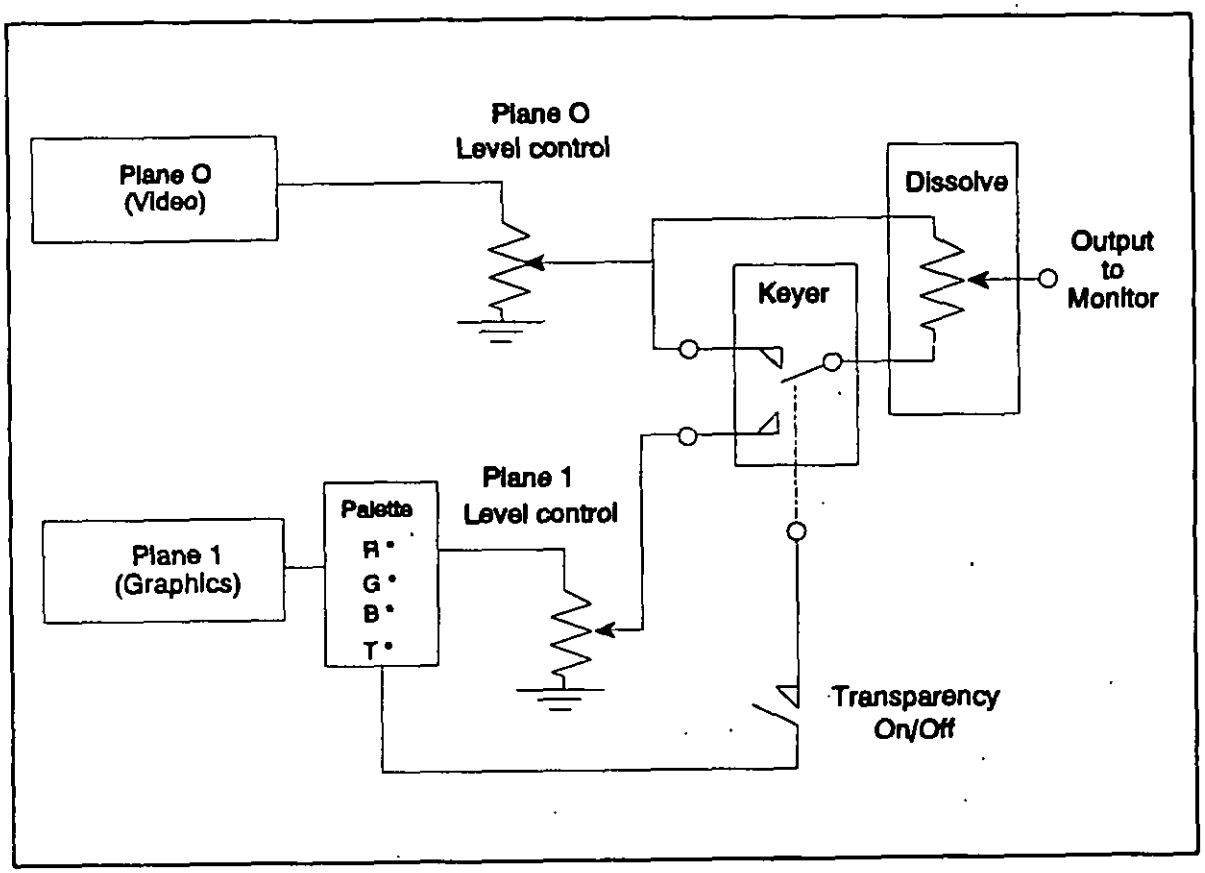

FIGURE B-1. Simplified functional model of a video overlay subsystem.

30.2.2 Mode trapping. The vm commands do not control trapping interrupt 10H (Mode trapping). VDI developers may, and probably should, implement mode trapping to protect against disruption of the graphics and background video by applications that change modes using direct interrupt 10H calls instead of the VDI vm commands. This is especially important for applications that may use separate graphics function libraries and similar tools.

30.2.3 Genlock control. The ability to turn genlock on and off is not included in the vm command set. The ICW Portability Practices assume that all video inputs and graphics are synchronous at all times from the application's viewpoint. Controlling genlocking is a video device handler issue that should be addressed by VDI Management developers and ICWTS integrators.

30.2.4 Graphics registration to the background video. Appendix G provides detailed information on assumed positions of graphics relative to background video. Proper use of the information in Appendix G should furnish reliable registration within about two pixels,

both horizontally and vertically. However, applications with critical registration requirements should include a position reference frame to allow dynamic positioning of the graphics plane at run time by the user. The vm command set includes commands for varying the graphics origin, both horizontally and vertically. An extended parameter supports setting the total width of the active graphics.

30.2.5 VGA graphics versus CGA and EGA graphics. Some graphics modes are displayed differently by VGA adapters versus CGA and EGA adapters, especially 620 X 200 and 320 X 200 resolution modes. VGA adapters display these modes so that the width of the active graphics area is equal to the width of the background video. CGA and EGA adapters leave right and left borders in these modes. Consequently:

a. For CGA and EGA applications to be portable to VGA-based systems, the VGA system must have the ability to display these modes as they would be displayed by a CGA or EGA adapter when overlay is on. Appendix G provides detailed information on VGA emulation of CGA and EGA graphics displays.

b. MIL-STD-1379 requires VGA emulation of CGA and EGA graphics displays in overlay modes.

30.2.6 Logical versus physical colors. The vm commands distinguish between logical and physical colors. A typical computer can not display all available colors simultaneously. For example, in VGA mode 19, the system can simultaneously display 256 colors taken from 262,144 possible colors. The vm commands refer to the number of colors that can be displayed simultaneously -- here 256 -- as the number of available logical colors. The commands refer to the total possible colors -- here 262,144 -- as the number of available physical colors, which is also commonly referred to as the palette size.

30.3 Rounding methods for fades and dissolves. The vm vmFade command must be supported (see MIL-STD-1379). However, compliant VDI Management software can be developed for ICW hardware without fade circuitry, although the hardware must support a fade on/off capability. To support systems with fade circuitry, visual-management should treat variations in available fade levels consistently.

30.3.1 Fade and dissolve levels. Applications pass graphics and video fade levels to VDI Management as integers in the range 0 through 255, which represent full off and full on, respectively. Intermediate values represent a linear transition of intensity. Dissolve levels are also passed as integers, where 0 represents display of video only and 255 represents hard keying with transparent colors at full video and opaque colors at full graphics.

30.3.2 Level value rounding. VDI Management allocates each available hardware setting a level in the range from 0 through 255, and rounds passed values to the nearest possible level. For example, if the hardware furnishes four fade levels with intensities of

full off, 1/3 on, 2/3 on, and full on; these are allocated the numeric values 0, 85, 170, and 255, respectively. Passed values in the range 0 through 42 are rounded to full off, 43-127 to 1/3 intensity, 128-212 to 2/3 intensity, and 213-255 to full on. Additionally:

a. If a fader is uneven, rounding ranges are adjusted accordingly. For example, if a fader can only do full off, 1/2 on, 3/4 on, and full on; these are allocated values of 0, 128, 192, and 255.

b. For fades and dissolves with non-zero time periods, levels are calculated as:

$$\textit{Current level} = \textit{start level} + \left(\frac{\textit{time since start}}{\textit{fade duration}} \times (\textit{end level} - \textit{start level}) \right)$$

c. The levels are then rounded in the usual way. Fade and dissolve times are expressed in milliseconds (ms). Assume a fade from full off to full on over 2550 ms (2.55 seconds) on a system that can fade to off, 1/3 on, 2/3 on, and full on. The fader stays off for the first 425 ms, at 1/3 from 425 to 1275 ms, at 2/3 from 1275 to 2125 ms, and at full on from 2125 to 2550 ms. The calculated fade and dissolve levels round in the usual way.

d. Applications can assume that levels 0 and 255 are available. On a system with no fade or dissolve circuitry, VDI Management switches to full off for level values 0-127 and full on for values of 128-255.

CAUTION

Application authors should not assume times are exact. On typical systems, the resolution of the tick interrupt or system clock restricts the accuracy of timings.

30.4 Palette issues. This section addresses several issues relating to graphics palettes including: initialization commands and palette settings; the effects of **vmSetPalette** on true CGAs and EGAs; return values from **vmGetPalette** on true CGAs and EGAs; mode 4 support on true CGAs; large palette support in 16-color, 200-line EGA modes on EGAs that include large-palette support but otherwise do not have extended palette hardware; and suggested default color palettes. The vm commands referred to in this section are described in a separate section of this appendix, beginning with Section 40, **vmFade** command.

30.4.1 Initialization commands and palette settings. The effects of initialization commands on palette settings are defined below. This includes the definition of a series of commands to set the default palette during initialization or reinitialization.

30.4.1.1 Relevant commands. The relevant initialization commands are **sylnit**, **vmInit**, and **vmSetGraphics**. Their effects on the vm service group are:

30.4.1.1.1 sylnit. **sylnit** sets the default video source to zero. This is functionally equivalent to **vmSetVideo defsource = 0**. (See Section 60 **sylnit** command in Appendix A.)

30.4.1.1.2 vmInit. **vmInit** initializes the overlay hardware and the vm service group. In addition:

a. **vmInit** does not change the graphics mode or palette, or clear the screen.

b. **vmInit** disables transparency, sets the transparency state of all colors to off, turns CGA/EGA emulation on for applicable modes, sets the graphics fade level to full on, and sets the video fade level to full off. This is functionally equivalent to **vmSetTrans enable = 0**,clear, **vmSetGraphics emulation = 1**, and **vmFade glevel = 255**,**vlevel = 0**.

30.4.1.1.3 vmSetGraphics. **vmSetGraphics gmode** sets the graphics display mode. When the graphics mode changes, the palette is reset to its default physical colors.

30.4.1.2 Initialization. To initialize the overlay card and the vm service group at the start of an application and select the default palette, three commands should be issued. These are:

```
sylnit                          ;initialize VDI, set defsource = 0
vmInit                          ;initialize overlay, turn transparency off
vmSetGraphics gmode = <mode>    ;set graphics mode, select default palette
```

30.4.1.3 Reinitialization. To reinitialize the vm service group after start-up, an application should not use **sylnit** because it affects other service groups. Instead, use **vmInit** followed by **vmSetVideo defsource = 0** and **vmSetGraphics gmode = <mode>**.

30.4.2 Effects of vmSetPalette on true CGAs and EGAs. On true CGAs and EGAs the physical colors for some logical colors cannot be changed. On such systems, **vmSetPalette** does nothing when asked to change a fixed color. In addition:

a. For example, mode 4 has four logical colors that, by default, are associated with the physical colors black, green, red, and yellow. Color zero, the background color, can be changed to any of 16 available colors. Therefore, on a true CGA the command **vmSetPalette color = 0**,**r = <value>**,**g = <value>**,**b = <value>** changes the background color by, if necessary, rounding the requested **r**, **g**, and **b** values to the actual values of 1 of the 16 available colors. However, the command **vmSetPalette color = 1**, **r = <value>**,**g = <value>**,**b = <value>** has no effect, because logical color one is always green and cannot be changed.

b. In mode 6, a two-color mode, logical color zero is always black, but color one can be changed to any of 16 colors. Therefore, on a true CGA adapter the command **vmSetPalette color=1,r=<value>,g=<value>,b=<value>** changes the foreground color. However, the command **vmSetPalette** *color=0,r=<value>,g=<value>, b=<value> has no effect.*

d. These limitations do not apply to VGAs emulating CGA/EGA, or to adapters with extended palette hardware. Such adapters have digital-to-analog converter (DAC) registers that can be programmed to change the physical color for any logical color. Available physical colors always should be the maximum number of colors that the hardware can furnish.

e. This behavior lets applications use the best available palette. Consider an application that uses graphics mode 4. The application can use **vmSetPalette** to request a particular palette. A true CGA uses the fixed green, red, yellow palette. An EGA uses the best approximation from 16 physical colors. A VGA uses the best approximation from 262,144 physical colors. In contrast an application using mode 16 would not be supported by CGA, would use the best approximation form 64 physical colors for EGA, and from 262,144 physical colors for VGA.

30.4.3 Return values from **vmGetPalette** on true CGAs and EGAs. **vmGetPalette** always reports the actually displayed physical colors. Therefore, if the physical color associated with a logical color is constant, the reported **r**, **g**, and **b** parameter values are always the same.

CAUTION

True CGA/EGA hardware does not have readable palette registers. For such hardware, VDI Management cannot detect palette changes unless they are made with **vmSetPalette**. Therefore, **vmGetPalette** is unreliable if developers use graphics tools that make palette changes with ROM-BIOS calls or direct hardware manipulation. (This problem does not occur with VGAs because VDI Management can read the DAC registers to retrieve palette values.)

30.4.4 Mode 4 support on true CGAs. **vmSetPalette** does not support the black, cyan, magenta, white palette in CGA graphics mode 4. Therefore, an application that must run on a true CGA and needs the alternate palette should use the ROM-BIOS to select this palette by issuing an Int 10H call with AX=0b00H, BX=0101H.

CAUTION

Using Int 10H invalidates **vmGetPalette** return values (See 30.4.3).

30.4.5 Large palette support in 16-color, 200-line EGA modes. Some EGAs support a large palette of 64 physical colors for 16-color, 200-line EGA modes (0–5, 13, and 14) but do not have extended palette hardware with programmable DAC registers. For these modes, the large palette is an enhanced alternative to the standard palette of 16 physical colors, which is also called the small palette. However, the small palette is implemented by all EGAs and is consistent with original IBM palette definitions. For maximum portability using EGAs that support a choice of either the large or small palette for 16-color, 200-line modes, the small palette of 16 physical colors should be used. In addition:

a. In modes 0-5, 13 and 14, **vmSetGraphics gmode = <mode>** should default to the small palette.

b. These limitations do not apply to VGAs emulating CGA/EGA or to adapters with extended palette hardware, except that changing the graphics mode to a 16-color mode results in the default small palette. For adapters with extended palette hardware, available physical colors always should be the maximum number of colors that the hardware can furnish.

30.4.6 Suggested default palettes. The following information is for the benefit of hardware vendors who have a choice of default color palette settings. However, exact **r**, **g**, and **b** values may vary slightly from system to system. Therefore, application developers should not count on these values being exact on all systems.

30.4.6.1 CGA mode 4. In CGA graphics mode 4, the default color palette is black, green, red and yellow. This corresponds to the **vmSetPalette r,g,b** values listed in Table B-2.

30.4.6.2 CGA mode 6. In CGA graphics mode 6, the default color palette is black and white. This corresponds to the **vmSetPalette r,g,b** values listed in Table B-3.

30.4.6.3 EGA 16- and 64-color modes. In EGA 16- and 64-color modes, both of which support 16 logical colors, the default color palette should be black, blue, green, cyan, red, magenta, brown, light gray, lark gray, light blue, light green, light cyan, light red, light magenta, yellow, and white. This corresponds to **vmSetPalette r,g,b** values listed in Table B-4.

30.4.6.4 VGA 256-color modes. In VGA 256-color modes the **vmSetPalette r,g,b** values listed in Table B-5 are suggested. (Maximum numbers are 252 instead of 255 because standard VGA palette values use 6-bit numbers.)

40. vmFade COMMAND

40.1 vmFade command summary. **vmFade** is a core command having a binary token number of 2051 decimal. **vmFade** sets the absolute levels of the graphics plane (**glevel**) and the video plane (**vlevel**), and the relative levels of video to graphics (**dlevel**) displayed on the screen. The specified level parameter changes to the specified level value over the specified **time**. The command returns immediately or, with the **wait** parameter, after the specified level is reached. Figure B-1 shows the relationship of the level parameters to the video overlay subsystem.

40.2 Description. Actual video and graphics levels set by VDI Management may vary from requested levels depending on system capabilities. If a system supports less than 256 levels for a given level parameter, VDI Management rounds the requested level to the closest supported level and **vmGetState** returns the actual level that was set after any required rounding when **vmFade** has finished execution. The binary version of **vmFade** returns the actual level that will be set after any rounding in the parameter packet (see 30.3).

40.3 Command parameters. The **vmFade** command has five (5) defined parameters: **dlevel**, **glevel**, **vlevel**, **time**, and **wait**. All are core parameters that must be supported by compliant applications. Table B-6 lists ASCII and binary parameter information for **vmFade**.

40.3.1 Dlevel parameter. The **dlevel** parameter creates transitions or dissolves between video only and hard keying with transparent colors at full video and opaque colors at full graphics. The parameter can be used to go from all video to all graphics or, when set to middle values, to create "ghosting" or highlighted effects with video showing through graphics. If a system cannot do dissolves, it switches to all video when the **dlevel** value is 0–127 and to hard keying when the **dlevel** value is 128–255. In addition:

a. With **dlevel = 0** the video plane only is visible. With **dlevel = 255**, opaque colors display graphics at full intensity and transparent colors display video only, assuming transparent colors are set for physical transparency and transparency is enabled (see 100.1). Assuming **dlevel = 255**, to create a transition from graphics only to video only, turn transparency off with **vmSetTrans enable = 0**, then issue **vmFade dlevel = 0**.

b. **Dlevel** differs from **glevel** and **vlevel** (see below) in that it controls the relative mix of video and graphics. Unlike **glevel** and **vlevel**, **dlevel** lets graphics appear mixed with video so that video and graphics are both visible in a ratio determined by the **dlevel** parameter. In contrast, **glevel** and **vlevel** set the total amount of video or graphics signal used.

c. Because **dlevel** sets the ratio of video to graphics, it is affected by **glevel** and **vlevel** values. For example, if **vlevel**=0, **vmFade dlevel**=255 will display graphics at full intensity but will not display video because the video signal has been turned off.

40.3.2 Glevel parameter. The glevel parameter sets the absolute intensity of the graphics plane in the range 0–255 (full off to full on). If a system supports graphics off and on only, it switches graphics off if the **glevel** value is 0–127 and on if the **glevel** value is 128–255. In addition:

a. A setting of **glevel**=0 is a fade to black, not to background video.

b. Transparent graphics are not affected by **glevel**. If a transparent color is set with **vmSetTrans** and transparency is enabled, video should still be visible through the transparent color after **vmFade glevel**=0.

40.3.3 Vlevel parameter. The vlevel parameter sets the absolute intensity of the video plane in the range 0–255 (full off to full on). If a system supports video off and on only, it switches video off if **vlevel** is 0–127 and on if **vlevel** is 128–255. A setting of **vlevel**=0 is a fade to black.

40.3.4 Time parameter. The time parameter specifies the number of milliseconds over which the fade or dissolve occurs, and:

a. If necessary, VDI Management rounds **time** to the nearest value the system supports.

b. The **time** parameter functions the same even if the hardware or system software does not support fades or dissolves.

40.3.5 Wait parameter. If the **wait** parameter is specified, vmFade does not return until the fade or dissolve has reached the specified level value at the end of the specified **time**. If **wait** is not used, vmFade returns immediately and the fade or dissolve executes as a background processing task. The **wait** parameter functions the same even if the system does not support fades or dissolves.

40.4 Implementation notes. When using **vmFade**, note the following:

a. **vmGetState** issued with the appropriate level parameter returns the current dissolve or fade level. This command can be used to determine if a background fade is complete. However, when a fade or dissolve is complete, the value returned by vmGetState may not agree with the requested level because of rounding. Therefore, programmers should test for limits instead of exact values.

b. If a system cannot do fades or dissolves, it switches to level 255 when a level parameter is set to 128–255, and to level 0 when a level is set to 0–127. However, this does not affect the wait and time parameters. For example, if the system is incapable of dissolves, after the commands:

```
vmFade dlevel=0
vmFade dlevel=255,time=60,wait
```

the system will remain at level 0 for 30 seconds, then switch to level 255, then return after another 30 seconds (see 30.3).

c. Only one level may be set with a single call to vmFade. However, vmFade commands can be queued with syQueue to create the effect of multiple, simultaneous fades and dissolves.

40.5 Return values.

40.5.1 ASCII returns. On success, the return is: "OK". On failure, the return is "ERROR n...".

40.5.2 Binary returns. On success, the return is AX = 0. Value associated with the dlevel, glevel, or vlevel parameters is a 32-bit integer that gives the actual level that is set by VDI Management after rounding, if rounding is required. On failure, the return is AX = error number.

40.6 Related commands. The syqueue, vmGetState, and vmSetTrans command requirements should also be reviewed in relation to implementing the vmFade command.

40.7 Examples. The following are ASCII and binary examples of the vmFade command.

40.7.1 ASCII.

Dissolve to all video over 1500 ms (1.5 s), do not return until complete	vmFade dlevel=0,time=1500,wait (returns) "OK"
Set display to hard keying immediately	vmFade dlevel=255 (returns) "OK"
Set video and graphics to 50% relative intensity	vmFade dlevel=128 (returns) "OK"

Fade graphics to full intensity over 300 - ms (0.3 seconds)	vmFade glevel = 255,time = 300 (returns) "OK"
Fade video to zero over 1000 ms, do not return until - complete.	vmFade vlevel = 0,time = 1000,wait (returns) "OK"
Set video to zero immediately	vmFade vlevel = 0 (returns) "OK"
Set video to half intensity immediately	vmFade vlevel = 128 (returns) "OK"

40.7.2 Binary.

Dissolve to 0 over 2500 ms (2.5 s) and return when done	AX	2051	; vmFade decimal value
	BX	3	; number of parameters
	ES:DI[0]	17	; dlevel decimal ID
	ES:DI[4]	0	; dlevel value
	ES:DI[8]	47	; time decimal ID
	ES:DI[C]	2500	; time value in ; milliseconds
	ES:DI[10	54	; wait decimal ID
	ES:DI[14]	Any value	; no associated value
After return	AX	0	; returns 0 if successful ; (non-zero if not)
	ES:DI[4]	0	; actual level set after ; rounding if required

50. vmGetPalette COMMAND

50.1 vmGetPalette command summary. **vmGetPalette** is a core command having a binary token number of 2053 decimal. **vmGetPalette** returns the amount of red, green, and blue components in a specified logical color via the ASCII interface, or one or more sets of component values for contiguous logical colors via the binary interface.

50.2 Description. vmGetpalette returns the proportions of the red, green, and blue components in a logical color as values in the range 0 through 255, where 255 is fully saturated for each component. Component values are returned for single colors via the ASCII interface, and for single or multiple colors via the binary interface depending on the calling parameters. In addition:

a. **vmGetState logcolors** returns the number of available logical colors and **vmGetState physcolors** returns the number of available physical colors (palette size) for the current graphic mode (see 60.2.8).

b. **vmGetPalette** returns the component values for the physical colors to logical colors assigned by **vmSetPalette** (see 90.2). For example, a system might support 16 logical colors from a palette of 4096 physical colors. Logical color 3 might be bright cyan with component values of **r=0, b=255, g=255.**

50.3 Command parameters. The **vmGetPalette** command has six defined parameters: **color, r, g, b,** length, and array. All are core parameters that must be supported by compliant VDI Management implementations. Table B-7 lists ASCII and binary parameter information for vmGetPalette.

50.3.1 Color + r, g, and b parameters. These parameters apply to both the ASCII and binary interfaces. The **color** parameter defines the logical color number for which **r, g,** and **b** component values are returned. Logical color numbers range from zero to the value returned by vmGetState logcolors minus one. In addition:

a. Exactly one **color** parameter must be listed. Specifying **color** twice causes error 54 (Parameter used more than once). Omitting **color** or failing to include at least one of the **r, g,** and **b** parameters causes error 49 (Insufficient parameters). Specifying a **color** number of less than zero or greater than **logcolors** minus one causes error 51 (Parameter value invalid or out of range).

b. Any or all of **r, g,** and **b** can be listed with a **color**. vmGetPalette returns a comma-separated list of the requested integer values via the ASCII interface or a 32-bit integer for each requested component via the binary interface.

50.3.2 Color + length and array parameters. The **length** and array parameters are available with the binary interface only. They furnish a way to pass a pointer to an array for storing a set of palette values in application memory. Additionally:

a. Using color arrays in the binary interface is described in 5.2.4.2.5.

b. Using **length** and **array** with any **r**, **g**, or **b** parameter causes error 50 (Parameters can not be used together). Specifying a length of less than one, or a color number of less than zero or greater than **logcolors** minus one causes error 51 (Parameter value invalid or out of range). Error 51 is also returned if the sum of **color** plus **length** is greater than **logcolors**.

50.3.3 Parameters resulting in errors. If a parameter causes an error, **vmGetPalette** returns immediately with an error message. The command does not return partial responses for other parameters that do not cause errors.

50.4 Implementation notes. When using **vmGetPalette**, note the following:

a. Values returned by vmGetPalette may not exactly match values set with **vmSetPalette** because of rounding caused when **vmSetPalette** component values do not match the component levels available on a specific system. For example, a system with four levels per component (0, 85, 170, and 255) will return a component value of 85 even though the value specified by **vmSetPalette** was 50.

b. VDI Management does not maintain palette arrays that are directly accessible by applications. Palette arrays for **vmGetPalette** are allocated by the application. To allocate memory in bytes for a palette array, use **length** x 4.

c. Trying to queue **vmGetPalette** causes error 177 (Command cannot be queued) at the time of the attempt.

50.5 Return values.

50.5.1 ASCII returns. On success, the return is a comma-separated list of requested **r**, **g**, and **b** component values in the range "0" through "255" for **color**. On failure, the return is "ERROR n...".

50.5.2 Binary returns. On success, the return is AX = 0. Values associated with **r**, **g**, and **b** parameters are 32-bit integers in the range 0–255 for **color**. Value associated with **length** parameter is a 32-bit integer giving the number of 4-byte structures in a palette array allocated by the application. Value associated with **array** parameter is a 32-bit pointer to the palette array. With **length** and **array**, the value associated with **color** is the first logical color in a contiguous series in the palette array. On failure, the return is

AX = error number. Any return values in the parameter block addressed by ES:DI are undefined and should be ignored.

50.6 Related commands. The syQueue, vmGetState, and vmSetPalette command requirements should also be reviewed in relation to implementing the vmGetPalette command.

50.7 Examples. The following are ASCII and binary examples of the vmGetPalette command.

50.7.1 ASCII.

Return values for all components for logical color zero.	vmGetPalette color=0,r,g,b (returns) "255,127,0"	; Logical color 0 has ; red=255, green=127, ; and blue=0.
Get red component for logical color one.	vmGetPalette color=1,r (returns) "170"	; logical color 1 has ; a red component of ; 170.

50.7.2 Binary.

Get green component value for color number 3	AX	2053	; vmGetPalette decimal ID
	BX	2	; number of parameters
	ES:DI[0]	9	; color decimal ID
	ES:DI[4]	3	; color number
	ES:DI[8]	25	; g decimal ID
	ES:DI[C]	any value	; place holder for value on ; return

After return	AX	0	; returns 0 if successful ; (non-zero if not)
	ES:DI[C]	g value	; green value for color 3.
Get color values for colors 3--9 (binary interface only)	AX	2053	; vmGetPalette decimal ID
	BX	3	; number of parameters
	ES:DI[0]	9	; color decimal ID
	ES:DI[4]	3	; first color to list in ; palette array
	ES:DI[8]	31	; length decimal ID
	ES:DI[C]	7	; number of color structures ; in palette array
	ES:DI[10]	1	; array decimal ID
	ES:DI[14]	pointer	; pointer to palette array in ; application memory
After return	AX	0	; returns 0 if successful ; (non-zero if not)
	ES:DI[14]	pointer	; pointer to same array with ; updated component values

60. vmGetState COMMAND

60.1 vmGetState command summary. **vmGetState** is a core command having a binary token number of 2054 decimal. **vmGetState** returns information about the state of the visual-management service group including the current settings of variable parameters and available resources such as palette size and number of video sources.

60.2 Command parameters. The vmGetState command has eighteen defined parameters: **color**, **defsource**, **dlevel**, **emulation**, **enable**, **horzpix**, **glevel**, **gmode**, **logcolors**, **physcolors**, **transcolors**, **tsources**, **vertpix**, **vlevel**, **vmode**, **width**, **xoffset**, and **yoffset**. All are core parameters that must be supported by compliant VDI Management implementations, except **width** which is an extended parameter. Implementing an extended parameter is optional (see 4.3.2.6). Tables B-8 and B-9 list ASCII and binary parameter information, respectively, for **vmGetState**.

60.2.1 Color parameter. The **color** parameter requests the transparency setting for a specified logical color number. A return value of one means that the specified color is set to transparent; zero means the specified color is opaque (see **enable** below).

60.2.2 Enable parameter. The **enable** parameter returns one if logically transparent colors are currently physically transparent to the video plane. Transparent colors are those which have been set to transparent with **vmSetTrans color=(logical color number),state=1**. After **vmSetTrans enable=1**, these colors reveal the video plane. After **vmSetTrans enable=0**, all graphics colors including transparent colors are visible and entirely cover the video plane.

60.2.3 Defsource parameter. The defsource parameter returns the default logical video source in the range 0–15. Note that a video source is always selected, but the source number does not necessarily equal the default device number. For example, logical player zero may be logical video source one. This mapping is determined at VDI installation/configuration time. The default source is defined as source zero unless **vmSetVideo defsource** is used to change it.

60.2.4 Dlevel, glevel, and vlevel parameters. The **dlevel**, **glevel**, and **vlevel** parameters return current levels in the range 0–255 for the dissolve level, graphics plane, and the video plane, respectively. Additionally:

a. The return values are actual values and may differ from the values requested by **vmFade** because of rounding.

b. The values returned by these parameters are the levels at the time of the request. These values may not equal the requested or actual target levels for dissolves and fades that may be in progress.

60.2.5 Emulation parameter. The **emulation** parameter returns the state of VGA emulation of CGA and EGA graphics versus VGA native mode. Implementations that support CGA or EGA graphics only always return one (on). (Refer to 30.2 and Appendix G for more information on VGA emulation of CGA and EGA graphics.)

60.2.6 Gmode parameter. The **gmode** parameter returns the current graphics mode. For MS-DOS, Intel 80X86 architecture systems, the mode returned is the same value that would be returned by a request to BIOS interrupt 10H, service 0FH (see 4.3.3).

60.2.7 Horzpix and vertpix parameters. The **horzpix** and **vertpix** parameters return the current pixel resolution; for example 640 and 200, respectively. These parameters are especially useful for determining the resolution of text modes where the number of pixels displayed on the screen varies from one graphics device to another (CGA, EGA, VGA).

60.2.8 Logcolors and physcolors parameters. The **logcolors** parameter returns the number of logical colors that are available. The **physcolors** parameter returns the range of physical colors that can be assigned to logical colors. Both return values are determined by the capabilities of the graphics hardware and mode.

60.2.9 Transcolors parameter. The **transcolors** parameter returns the total number of logical colors that can be made transparent with **vmSetTrans**.

60.2.10 Tsources parameter. The tsources parameter returns the total number of video sources for which VDI Management was installed.

60.2.11 Vmode parameter. The **vmode** parameter returns the video mode set by **vmSetVideo**. The **vmode** is either 0 (native), 1 (NTSC), or 2 (PAL). Native mode is a non-overlay mode, but setting native mode does not change overlay parameters. NTSC and PAL value returns mean the system is configured for the indicated video standard and the returned mode has been set with vmSetVideo.

60.2.12 Width parameter. The width parameter returns the total graphics width in nanoseconds. This parameter lets applications accurately establish the right edge of the active graphics area relative to background video (see Appendix G). **Width** is an extended parameter. Using an unimplemented extended parameter causes error 52 (Parameter invalid for this command).

60.2.13 Xoffset and yoffset parameters. The **xoffset** and **yoffset** parameters return the offset of the graphics plane relative to the video plane in pixels, as set by **vmSetGraphics**. As described in 80.2.4 and Appendix G:

a. The origin of the graphics plane is the upper left corner of the graphics display area.

b. Some systems may not be able to set these parameters in one-pixel increments and will round to the nearest possible offset. For such systems, **xoffset** and **yoffset** will return the actual values set after any required rounding. The return values may not be the values requested by vmSetGraphics due to rounding.

60.2.14 Parameters resulting in errors. If a parameter causes an error, vmGetState returns immediately with an error message. The command does not return partial responses for other parameters that do not cause errors.

60.3 Implementation notes. When using vmGetState, note that trying to queue vmGetState causes error 177 (Command cannot be queued) at the time of the attempt.

60.4 Return values.

60.4.1 ASCII returns. On success, the return is a comma-separated list of values for requested parameters as described in 60.2. On failure, the return is "ERROR n...".

60.4.2 Binary returns. On success, the return is AX = 0. Values associated with requested parameters are 32-bit values of the types given in Table B-9. On failure, the return is AX = error number. Any return values in the parameter block addressed by ES:DI are undefined and should be ignored.

60.5 Related commands. The syGetState, syQueue, vdGetState, vmFade, vmGetPalette, vmInit, vmSetGraphics, vmSetVideo, vmSetTrans, and xyGetState command requirements should also be reviewed in relation to implementing the vmGetState command.

60.6 Examples. The following are ASCII and binary examples of the vmGetState command.

60.6.1 ASCII.

Get the current graphics level and mode	vmGetState glevel,gmode (returns) "0,14"	; graphics level currently ; set to 0 and graphics mode ; is 14 (640 X 200).
Determine whether logical color 3 is transparent	vmGetState color=3 (returns) "1"	; logical color 3 is ; transparent

60.6.2 Binary.

Get current graphics level, graphics mode, and number of physical colors.	AX	2054	; vmGetState decimal ID
	BX	3	; number of parameters
	ES:DI[0]	26	; glevel decimal value
	ES:DI[4]	any value	; place holder for glevel ; return value
	ES:DI[8]	36	; gmode decimal ID
	ES:DI[C]	any value	; place holder for gmode ; return value.
	ES:DI[10]	36	; physcolors decimal ID
	ES:DI[14]	any value	; place holder for physcolors ; return value.
After Return	AX	0	; returns 0 if successful or ; non-zero if not.
	ES:DI[4]	level value	; contains current graphics ; level.
	ES:DI[C]	gmode value	; contains current graphics ; mode number.
	ES:DI[14]	physcolors value	; contains available physical ; colors.

70. vmInit COMMAND

70.1 vmInit command summary. vmInit is a core command having a binary token number of 2055 decimal. **vmInit** initializes the visual management hardware and software, placing both in a known state.

70.2 Command parameters. The vmInit command has no parameters.

70.3 Conditions set by vmInit. The vmInit command sets the parameters listed in Table B-10 to the specified values.

70.4 Implementation notes. When using **vmInit**, note the following:

a. In typical VDI Management implementations, **vmInit** turns mode trapping on to intercept video BIOS interrupt 10H calls. This is done to prevent applications making graphics mode changes without VDI Management's knowledge. This is especially important for applications that use graphics libraries and similar tool kits. If **vmInit** turns mode trapping on, syStop should turn it off.

b. The default **defsource** is video source zero. However, if an application uses **vmSetVideo defsource** to change the source, a subsequent **vmInit** does not reset the source to zero, and any applicable parameters affect the source set by vmSetVideo.

c. Trying to queue **vmInit** causes error 177 (Command cannot be queued) at the time of the attempt.

70.5 Return values.

70.5.1 ASCII returns. On success, the return is "OK". On failure, the return is "ERROR n...".

70.5.2 Binary returns. On success, the return is AX = 0. On failure, the return is AX = error number.

70.6 Related commands. The syInit, syStop, vdInit, and **xyInit** command requirements should be reviewed in relation to implementing the **vmInit** command.

70.7 Examples. The following are ASCII and binary examples of the **vmInit** command.

70.7.1 ASCII.

Initialize visual management services	**vmInit** (returns)"OK"

70.7.2 Binary.

Initialize visual management services	AX	2055	; **vmInit** decimal ID
	BX	0	; number of parameters
After return			
	AX	0	; returns 0 if successful ; (non-zero if not)

80. vmSetGraphics COMMAND

80.1 vmSetGraphics command summary. **vmSetGraphics** is a core command having a binary token number of 2062 decimal. **vmSetGraphics** sets the graphics mode and the position of the graphics plane relative to the video plane.

80.2 Command parameters. The **vmSetGraphics** command has five defined parameters: **emulation**, **gmode**, **width**, **xoffset**, and **yoffset**. Four are core parameters: **width** is an extended parameter. All core parameters must be supported by compliant VDI Management implementations. Implementation of extended parameters is optional (see 4.3.2.6). Table B-11 lists ASCII and binary parameter information for **vmSetGraphics**.

80.2.1 Emulation parameter. The **emulation** parameter controls VGA emulation of CGA and EGA horizontal graphics positioning in common overlay modes (see 30.2.5 and Appendix G). If **emulation**=**1**, the default set by **vmInit**, then a VGA adapter will leave the same borders on the right and left edges of active graphics that a true CGA or EGA adapter would leave. If emulation=0, then the graphics from a VGA adapter cover the entire width of the background video. Issuing vmSetGraphics emulation=0 on a true CGA- or EGA-based system returns error 194 (Unsupported graphics mode).

80.2.2 Gmode parameter. The **gmode** parameter sets the graphics display mode. This parameter places mode changes under VDI Management control to keep screen disruption to a minimum (as opposed to using mode functions furnished separately with development systems). In addition:

a. For MS-DOS, Intel 80X86-architecture systems, The **gmode** value passed is in accordance with IBM graphics mode numbers as returned by BIOS interrupt 10H, service 0FH (see 4.3.3).

b. Requesting an unsupported graphics mode returns error 194 (Unsupported graphics mode).

c. Changing the graphics mode results in the default palette for the new mode with the transparency states of all colors set to off.

80.2.3 Width parameter. The **width** parameter sets the total graphics width in nanoseconds. This parameter lets applications accurately establish the right-hand edge of the active graphics area relative to background video. When using the **width** parameter:

a. If the graphics width is critical, an application should typically display a videodisc position-reference frame for interactively setting **width**.

b. **Width** is an extended parameter. Using an unimplemented extended parameter causes error 52 (Parameter invalid for this command).

c. **Width** should be implemented so that the entire graphics display expands and contracts in a way that maintains the original image as accurately as possible. Adjusting to a shorter width by truncating the right edge of the graphics display is noncomplaint.

80.2.4 Xoffset and yoffset parameters. The **xoffset** and **yoffset** parameters set the offset of the upper left corner of the graphics display area relative to video. These parameters shift the entire graphics display area up, down, left, and right within the video raster in one-pixel increments. Positive values shift down and right: negative values shift up and left. Refer to Appendix G for more information on graphics registration. In addition:

a. Some systems may not be able to set these parameters in one-pixel increments. Such systems should round to the nearest possible offset. For example, assume a system can only increment by four pixels. **vmSetGraphics xoffset = 2** would result in no offset while **vmSetGraphics xoffset = 3** would result in an actual offset of four. Note that **vmGetState xoffset** returns the actual value set after rounding, not the requested value.

b. Offset values are absolute, not cumulative. Issuing **vmSetGraphics yoffset = 4** twice results in an offset of four, not eight. Values that exceed the maximum that a system can shift the graphics plane result in the maximum possible shift.

c. The offset values set by vmSetGraphics remain in effect until explicitly reset by **vmSetGraphics** or vmInit. They do not change to compensate for graphics mode changes. Therefore, apparent offsets may change with graphics mode changes because of differences in pixel sizes among modes.

80.3 Implementation notes. When using **vmSetGraphics**, note the following:

a. **vmGetState** returns the actual X and Y offsets. These values will not agree with the values set by vmSetGraphics if the specified values exceed the maximum amount the system can shift the graphics plane or the system has rounded the values to compensate for a lack of resolution.

b. The **xoffset** and **yoffset** parameters are for correcting graphics registration to video. Applications should not use them for special effects such as scrolling the screen because they may cause screen disturbances.

80.4 Return values.

80.4.1 ASCII returns. On success, the return is "OK". On failure, the return is "ERROR n...".

80.4.2 Binary returns. On success, the return is AX = 0. On failure, the return is AX = error number.

80.5 Related commands. The vmGetState and vmInit command requirements should also be reviewed in relation to implementing the vmSetGraphics command.

80.6 Examples. The following are ASCII and binary examples of the vmSetGraphics command.

80.6.1 ASCII.

Set X and Y offsets to -1 and 5	vmSetGraphics xoffset=-1,yoffset=5 (returns)"OK"
Set graphics mode 16 (EGA 640 X 350)	vmSetGraphics gmode=16 (returns)"OK"

80.6.2 Binary.

Shift graphics right one and down two.	AX	2062	; vmSetGraphics decimal ID
	BX	2	; number of parameters
	ES:DI[0]	60	; xoffset decimal ID
	ES:DI[4]	1	; pixel offset value
	ES:DI[8]	66	; yoffset decimal ID
	ES:DI[C]	2	; pixel offset
After return	AX	0	; returns 0 if successful and ; non-zero if not.

90. vmSetPalette COMMAND

90.1 vmSetPalette command summary. vmSetPalette is a core command having a binary token number of 2063 decimal. **vmSetPalette** assigns red, green, and blue component values to the specified logical color via the ASCII interface, or to one or more contiguous logical colors via the binary interface.

90.2 Description. vmSetPalette sets the proportions of the red, green, and blue components in a logical color as values in the range 0–255, where 255 is fully saturated. Component values are set for single colors via the ASCII interface and for single or multiple colors via the binary interface depending on the calling parameters. In addition:

a. **vmGetState logcolors** returns the number of available logical colors for a system, and **vmGetState physcolors** returns the number of available physical colors for the current graphics mode.

b. vmSetPalette assigns physical colors to logical colors and **vmGetPalette** returns the component values for the assigned colors. For example, a system might support 16 logical colors from a palette of 4096 physical colors. Logical color 3 might be bright cyan with component values of r=0, b=255, g=255.

90.3 Command parameters. The vmSetPalette command has six defined parameters: **color**, **r**, **g**, **b**, **length**, and **array**. All are core parameters that must be supported by compliant VDI Management implementations. Table B-12 lists ASCII and binary parameter information for **vmSetPalette**.

90.3.1 Color + r, g, and b parameters. These parameters apply to both the ASCII and binary interfaces. The **color** parameter defines the logical color number for which **r**, **g**, and **b** component values are set. Logical **color** numbers range from zero to the value returned by vmGetState logcolors minus one. Additionally:

a. VDI Management maps the specified component levels to the color as closely as possible given the size of the available palette. For example, if the palette furnishes four color levels (0, 85, 170, and 255) for each component (64-color palette), **vmSetPalette color**=1,r=110 results in a mapped value of r=85.

b. Exactly one **color** parameter must be given. Specifying **color** twice causes error 54 (Parameter used more than once) while omitting **color** entirely or failing to include at least one of **r**, **g**, and **b** causes error 49 (Insufficient parameters). Specifying a color number of less than zero or greater than **logcolors** minus one causes error 51 (Parameter value invalid or out of range) Any or all of **r**, **g**, and **b** can be specified in the same call.

90.3.2 Color + length and array parameters. The length and array parameters are available with the binary interface only. They provide a way to pass a pointer to an array for storing a set of palette values in application memory. These parameters function as described in 5.2.4.2.5 and 50.3.2.

90.4 Implementation notes. When using **vmSetPalette**, note the following:

a. Use **syQueue** to set multiple logical colors in the same vertical interval via the ASCII interface.

b. Component values returned by **vmGetPalette** may not agree exactly with values set by **vmSetPalette** because of rounding. For example, a system with 4 levels per component (0, 85, 170, 255) will return a component value of 85 even though the value specified by **vmSetPalette** was 55.

c. VDI Management does not maintain palette arrays that are directly accessible by applications. Palette arrays for **vmSetPalette** must be allocated by the application. To allocate memory in bytes for a palette array, use **length** X 4.

d. If **vmSetPalette** is used with a palette array and queued with **syQueue**, do not deallocate the array before executing the queue.

90.5 Return values.

90.5.1 ASCII returns. On success, the return is "OK". On failure, the return is "ERROR n...".

90.5.2 Binary returns. On success, the return is AX = 0. On failure, the return is AX = error number.

90.6 Related commands. The syQueue, **vmGetPalette**, and **vmGetState** command requirements should also be reviewed in relation to implementing the **vmSetPalette** command.

90.7 Examples. The following are ASCII and binary examples of the **vmSetPalette** command.

90.7.1 ASCII.

Set red to 63 for color 0. Do not change other components	**vmSetPalette color=0,r=63** (returns)"OK"

Set color 1 to fully saturated blue	vmSetPalette color=1,r=0,g=0,b=255 (returns)"OK"
Set color 2 to bright white	vmSetPalette color=2,r=255,g=255,b=255 (returns)"OK"

90.7.2 Binary.

Set green to 127 for color 3. Do not change other components	AX	2063	; vmSetPalette decimal ID
	BX	2	; number of parameters
	ES:DI[0]	9	; color decimal ID
	ES:DI[4]	3	; color number
	ES:DI[8]	25	; g decimal ID
	ES:DI[C]	127	; green value
After return	AX	0	; returns 0 if successful, and ; non-zero if not.
Set component values for colors 3-9	AX	2063	; vmSetPalette decimal ID
	BX	3	; number of parameters
	ES:DI[0]	9	; color decimal ID
	ES:DI[4]	3	; first color of array list
	ES:DI[8]	31	; length decimal ID
	ES:DI[C]	7	; number of color structures in ; palette array
	ES:DI[10]	1	; array decimal ID
	ES:DI[14]	pointer	; pointer to palette array in ; application memory
After return	AX	0	; returns zero if successful, ; and non-zero if not.

100. vmSetTrans COMMAND

100.1 vmSetTrans command summary. vmSetTrans is a core command having a binary token number of 2064 decimal. vmSetTrans sets logical colors to transparent or opaque and turns physical transparency on and off.

100.2 Command parameters. The vmSetTrans command has four defined parameters: clear, color, enable, and state. All are core parameters that must be supported by compliant VDI Management implementations. Table B-13 lists ASCII and binary parameter information for vmSetTrans.

100.2.1 Clear parameter. The clear parameter sets the transparency state (see 100.2.2) of all logical colors to zero (off). Note that this not only turns transparency off but also changes the values of color attributes. In addition:

a. The clear parameter does not affect any parameters other than the state parameters for all logical colors. Use the enable parameter (see 100.2.3) to turn transparency off without changing the transparency settings of the colors.

b. Using vmSetTrans clear when no transparent colors are set does nothing and is not an error. Using clear with any other parameter returns error 50 (Parameters cannot be used together).

100.2.2 Color and state parameters. The color and state parameters work together to set logical colors to opaque or transparent. vmSetTrans color= (logical color number),state=1 makes colors transparent; vmSetTrans color= (logical color number),state=0 makes colors opaque. To temporarily override transparent colors, use vmSetTrans enable.

100.2.3 Enable parameter. The enable parameter controls physical transparency on the display screen. vmSetTrans enable=1 makes all designated transparent colors actually become transparent to the video plane. Areas containing transparent colors on the screen show the video plane only. When using enable:

a. If dlevel=255 (see 40.1, vmFade), vmSetTrans enable=0 makes all colors physically opaque regardless of their transparency settings. None of the video plane is visible. However, if dlevel=0, video only is visible. Transparent colors keep their transparency settings and will again be physically transparent after a subsequent vmSetTrans enable=1.

b. Enable can be combined with a color and a state to specify a transparent color and turn transparency on with the same command. The default for enable after a vminit is zero (transparency off).

100.3 Implementation notes. When using **vmSetTrans**, note the following:

a. Using **vmSetTrans** to try to set more than **transcolors** to transparent returns error 51 (Parameter invalid or out of range).

b. Compliant VDI implementations support transparency for at least one color that can be assigned to any logical color. Applications striving for maximum portability should not assume more than one transparent color.

c. **vmGetState color** returns the transparency setting for a single specified color. **vmGetState transcolors** returns the number of logical colors that can be made transparent (see 60.2).

100.4 Return values.

100.4.1 ASCII returns. On success, the return is "OK". On failure, the return is "ERROR n...".

100.4.2 Binary returns. On success, the return is AX = 0. On failure, the return is AX = error number.

100.5 Related commands. The **vmGetState** command requirements should also be reviewed in relation to implementing the **vmSetTrans** command.

100.6 Examples. The following are ASCII and binary examples of the **vmSetTrans** command.

100.6.1 ASCII.

Set color 0 to transparent and enable physical transparency	**vmSetTrans color=0,state=1,enable=1** (returns)"OK"
Set color 5 to opaque	**vmSetTrans color=5,state=0** (returns)"OK"
Set all colors to opaque	**vmSetTrans clear** (returns)"OK"

100.6.2 Binary.

Make color 3 transparent and enable physical transparency	AX	2064	; vmSetTrans decimal ID
	BX	3	; number of parameters
	ES:DI[0]	9	; color decimal ID
	ES:DI[4]	3	; color number
	ES:DI[8]	42	; state decimal ID
	ES:DI[C]	1	; make color 3 transparent
	ES:DI[10]	20	; enable decimal idea
	ES:DI[14]	1	; turn physical transparency on
After return	AX	0	; returns 0 if successful and ; non-zero if not
Make all colors opaque	AX	2064	; vmSetTrans decimal ID
	BX	1	; number of parameters
	ES:DI[0]	8	; clear decimal ID
	ES:DI[4]	Any value	
After return	AX	0	; returns 0 if successful, and ; non-zero if not

110. vmSetVideo COMMAND

110.1 vmSetVideo command summary. vmSetVideo is a core command having a binary token number of 2065 decimal. vmSetVideo sets the video mode and selects the video input source if more than one source is available.

110.2 Command parameters. The vmSetVideo command has two defined parameters: **defsource** and **vmode**. Both are core parameters that must be supported by compliant VDI Management implementations. Table B-14 lists ASCII and binary parameter information for **vmSetVideo**.

110.2.1 Defsource parameter. The defsource parameter selects a video input source in the range 0 through 15 when more than one video source is available. The default at start-up is source zero.

110.2.2 Vmode parameter. The vmode parameter tells the visual-management system which video standard incoming video and the monitor are using. This lets VDI Management use the appropriate timing values for the standard. When using vmode:

a. **Vmode = 1** sets VDI Management for NTSC and **vmode = 2** sets it for PAL. **vmSetVideo vmode = 0** (native) sets the system to the functionality and appearance that the computer would use if it were not an ICW system. This setting turns overlay off without affecting any other parameters relating to overlay.

b. **vmSetVideo vmode** may cause screen disturbances because of the asynchronous rates of the graphics and video signals.

110.3 Implementation notes. When using **vmSetVideo**, note the following:

a. A video source is always selected, but the source number will not necessarily equal the current default player number. For example, logical player zero may be logical video source one. This mapping is done at VDI Management installation/configuration time (see 5.2.1).

b. After a player is selected with vdSet defdevice (see Appendix C), it is activated as a video source using vmSetVideo defsource unless the start-up source (source zero) is already mapped to the device.

110.4 Return values.

110.4.1 ASCII returns. On success, the return is "OK". On failure, the return is "ERROR n...".

110.4.2 Binary returns. On success, the return is AX = 0. On failure, the return is AX = error number.

110.5 Related commands. The vdSet and vmGetState command requirements should also be reviewed in relation to implementing the vmSetVideo command.

110.6 Examples. The following are ASCII and binary examples of the vmSetVideo command.

110.6.1 ASCII.

Set the standard to NTSC	vmSetVideo vmode = 1 (returns)"OK"
Make video source one the default	vmSetVideo defsource = 1 (returns)"OK"

110.6.2 Binary.

Set mode to NTSC	AX	2065	; vmSetVideo decimal command ID
	BX	1	; number of parameters
	ES:DI[0]	53	; vmode decimal ID
	ES:DI[4]	1	; sets mode to NTSC (1)
After return	AX	0	; returns 0 if successful, and ; non-zero if not

APPENDIX B

TABLE B-1. Visual-Management (vm) commands summary.

ASCII command name[1]	Binary interface token number (decimal)	Type [2]
vmFade	2051	Core
vmGetPalette	2053	Core
vmGetState	2054	Core
vmInit	2055	Core
vmSetGraphics	2062	Core
vmSetPalette	2063	Core
vmSetTrans	2064	Core
vmSetVideo	2065	Core

[1] Upper or lower case for command names is not significant.

[2] Compliant implementations must support "Core" commands.

TABLE B-2. Suggested default values for CGA mode 4.

Color (number)	r	g	b
Black (0)	0	0	0
Green (1)	0	153	0
Red (2)	153	0	0
Yellow (3)	153	153	0

TABLE B-3. Suggested default values for CGA mode 6.

Color (number)	r	g	b
Black (0)	0	0	0
White (1)	153	153	153

TABLE B-4. Suggested default values for EGA 16- and 64-color modes.

Color (number)	r	g	b
Black (0)	0	0	0
Blue (1)	0	0	153
Green (2)	0	153	0
Cyan (3)	0	153	153
Red (4)	153	0	0
Magenta (5)	153	0	153
Brown (6)	153	153	0
Light gray (7)	153	153	153
Dark gray (7)	102	102	102
Light blue (8)	102	102	255
Light green (10)	102	255	102
Light cyan (11)	102	255	255
Light red (12)	255	102	102
Light magenta (13)	255	102	255
Yellow (14)	255	255	102
White (15)	255	255	255

TABLE B-5. Suggested default values for VGA 256-color modes.

Color number	r	g	b
0	0	0	0
1	0	0	168
2	0	168	0
3	0	168	168
4	168	0	0
5	168	0	168
6	168	84	0
7	168	168	168
8	84	84	84
9	84	84	252
10	84	252	84
11	84	252	252
12	252	84	84
13	252	84	252
14	252	252	84
15	252	252	252
16	0	0	0
17	20	20	20
18	32	32	32
19	44	44	44
20	56	56	56
21	68	68	68
22	80	80	80
23	96	96	96

TABLE B-5. Suggested default values for VGA 256-color modes - Continued.

Color number	r	g	b
24	112	112	112
25	128	128	128
26	144	144	144
27	160	160	160
28	180	180	180
29	200	200	200
30	224	224	224
31	252	252	252
32	0	0	252
33	64	0	252
34	124	0	252
35	188	0	252
36	252	0	252
37	252	0	188
38	252	0	124
39	252	0	64
40	252	0	0
41	252	64	0
42	252	124	0
43	252	188	0
44	252	252	0
45	188	252	0
46	124	252	0
47	64	252	0

TABLE B-5. Suggested default values for VGA 256-color modes - Continued.

Color number	r	g	b
48	0	252	0
49	0	252	64
50	0	252	124
51	0	252	188
52	0	252	252
53	0	188	252
54	0	124	252
55	0	64	252
56	124	124	252
57	156	124	252
58	188	124	252
59	220	124	252
60	252	124	252
61	252	124	220
62	252	124	188
63	252	124	156
64	252	124	124
65	252	156	124
66	252	188	124
67	252	220	124
68	252	252	124
69	220	252	124
70	188	252	124
71	156	252	124

TABLE B-5. Suggested default values for VGA 256-color modes - Continued.

Color number	r	g	b
72	124	252	124
73	124	252	156
74	124	252	188
75	124	252	220
76	124	252	252
77	124	220	252
78	124	188	252
79	124	156	252
80	180	180	252
81	196	180	252
82	216	180	252
83	232	180	252
84	252	180	252
85	252	180	232
86	252	180	216
87	252	180	196
88	252	180	180
89	252	196	180
90	252	216	180
91	252	232	180
92	252	252	180
93	232	252	180
94	216	252	180
95	196	252	180

TABLE B-5. Suggested default values for VGA 256-color modes - Continued.

Color number	r	g	b
96	180	252	180
97	180	252	196
98	180	252	216
99	180	252	232
100	180	252	252
101	180	232	252
102	180	216	252
103	180	196	252
104	0	0	112
105	28	0	112
106	56	0	112
107	84	0	112
108	112	0	112
109	112	0	84
110	112	0	56
111	112	0	28
112	112	0	0
113	112	28	0
114	112	56	0
115	112	84	0
116	112	112	0
117	84	112	0
118	56	112	0
119	28	112	0

TABLE B-5. Suggested default values for VGA 256-color modes - Continued.

Color number	r	g	b
120	0	112	0
121	0	112	28
122	0	112	56
123	0	112	84
124	0	112	112
125	0	84	112
126	0	56	112
127	0	28	112
128	56	56	112
129	68	56	112
130	84	56	112
131	96	56	112
132	112	56	112
133	112	56	96
134	112	56	84
135	112	56	68
136	112	56	56
137	112	68	56
138	112	84	56
139	112	96	56
140	112	112	56
141	96	112	56
142	84	112	56
143	68	112	56

TABLE B-5. Suggested default values for VGA 256-color modes - Continued.

Color number	r	g	b
144	56	112	56
145	56	112	68
146	56	112	84
147	56	112	96
148	56	112	112
149	56	96	112
150	56	84	112
151	56	68	112
152	80	80	112
153	88	80	112
154	96	80	112
155	104	80	112
156	112	80	112
157	112	80	104
158	112	80	96
159	112	80	88
160	112	80	80
161	112	88	80
162	112	96	80
163	112	104	80
164	112	112	80
165	104	112	80
166	96	112	80
167	88	112	80

TABLE B-5. Suggested default values for VGA 256-color modes - Continued.

Color number	r	g	b
168	80	112	80
169	80	112	88
170	80	112	96
171	80	112	104
172	80	112	112
173	80	104	112
174	80	96	112
175	80	88	112
176	0	0	64
177	16	0	64
178	32	0	64
179	48	0	64
180	64	0	64
181	64	0	48
182	64	0	32
183	64	0	16
184	64	0	0
185	64	16	0
186	64	32	0
187	64	48	0
188	64	64	0
189	48	64	0
190	32	64	0
191	16	64	0

TABLE B-5. Suggested default values for VGA 256-color modes - Continued.

Color number	r	g	b
192	0	64	0
193	0	64	16
194	0	64	32
195	0	64	48
196	0	64	64
197	0	48	64
198	0	32	64
199	0	16	64
200	32	32	64
201	40	32	64
202	48	32	64
203	56	32	64
204	64	32	64
205	64	32	56
206	64	32	48
207	64	32	40
208	64	32	32
209	64	40	32
210	64	48	32
211	64	56	32
212	64	64	32
213	56	64	32
214	48	64	32
215	40	64	32

TABLE B-5. Suggested default values for VGA 256-color modes - Continued.

Color number	r	g	b
216	32	64	32
217	32	64	40
218	32	64	48
219	32	64	56
220	32	64	64
221	32	56	64
222	32	48	64
223	32	40	64
224	44	44	64
225	48	44	64
226	52	44	64
227	60	44	64
228	64	44	64
229	64	44	60
230	64	44	52
231	64	44	48
232	64	44	44
233	64	48	44
234	64	52	44
235	64	60	44
236	64	64	44
237	60	64	44
238	52	64	44
239	48	64	44

TABLE B-5. Suggested default values for VGA 256-color modes - Continued.

Color number	r	g	b
240	44	64	44
241	44	64	48
242	44	64	52
243	44	64	60
244	44	64	64
245	44	60	64
246	44	52	64
247	44	48	64
248	0	0	0
249	0	0	0
250	0	0	0
251	0	0	0
252	0	0	0
253	0	0	0
254	0	0	0
255	0	0	0

TABLE B-6. vmFade parameters.

ASCII Parameters						
Parameter[1]	Core or extended	Associated calling value	Type as ASCII	Associated return value	Type as ASCII	Default if parameter not used
dlevel	Core	Dissolve level, 0-255	Integer	None	N/A	No action
glevel	Core	Graphics level, 0-255	Integer	None	N/A	No action
vlevel	Core	Video level, 0-255	Integer	None	N/A	No action
time	Core	Milliseconds for fade or dissolve	Integer	None	N/A	0
wait	Core	None	N/A	None	N/A	No wait
Binary Parameters						
Parameter[1]	Core or extended	Token number (decimal)	Type	Associated calling value	Associated return value	Default if parameter not used
dlevel	Core	17	Integer	Dissolve level, 0-255	Actual level after rounding if required	No action
glevel	Core	26	Integer	Graphics level, 0-255	Actual level after rounding if required	No action
vlevel	Core	52	Integer	Video level, 0-255	Actual level after rounding if required	No action
time	Core	47	Integer	Milliseconds for fade or dissolve	None	0
wait	Core	54	N/A	None	None	No wait

[1] Exactly one of **dlevel**, **glevel**, or **vlevel** must be specified or an error is returned.

TABLE B-7. vmGetPalette parameters.

ASCII Parameters						
Parameter	Core or extended	Associated calling value	Type as ASCII	Associated return value	Type as ASCII	Default if parameter not used
color[1]	Core	Logical color number	Integer	None	N/A	Causes error
r	Core	None	N/A	Red value, 0-255	Integer	No action
g	Core	None	N/A	Green value, 0-255	Integer	No action
b	Core	None	N/A	Blue value, 0-255	Integer	No action
Binary Parameters						
Parameter	Core or extended	Token number (decimal)	Type	Associated calling value	Associated return value	Default if parameter not used
color[2]	Core	9	Integer	Logical color number	None	Causes error
r	Core	38	Integer	Any value	Red value, 0-255	No action
g	Core	25	Integer	Any value	Green value, 0-255	No action
b	Core	4	Integer	Any value	Blue value, 0-255	No action
length	Core	31	Integer	Number of color array entries [3]	Number of color array entries [3]	No action
array	Core	1	Pointer	Pointer to color array	Pointer to color array	No action

[1] Exactly one color and at least one of r, g, and b are required or an error is returned.

[2] Either exactly one color and at least one of r, g, and b are required; or color, length, and array must be used together; or an error is returned.

[3] Color array entry equals 4 bytes comprised of 3 components plus the reserved byte set to NUL.

TABLE B-8. vmGetState ASCII parameters.

Parameter[1]	Core or extended	Associated calling value	Type as ASCII	Associated return value	Type as ASCII	Default if parameter not used
color	Core	Logical color number	Integer	1 (transparent) | 0 (opaque)	Integer	No action
defsource	Core	None	N/A	Default video source, 0-15	Integer	No action
dlevel	Core	None	N/A	Current level, 0-255	Integer	No action
emulation	Core	None	N/A	1 (on) | 0 (off)	Integer	No action
enable	Core	None	N/A	1 (on) | 0 (off)	Integer	No action
horzpix	Core	None	N/A	Total horizontal pixels in current gmode	Integer	No action
glevel	Core	None	N/A	Current level, 0-255	Integer	No action
gmode	Core	None	N/A	Current graphics mode	Integer	No action
logcolors	Core	None	N/A	Total available	Integer	No action
physcolors	Core	None	N/A	Total available	Integer	No action
transcolors	Core	None	N/A	Total available	Integer	No action
tsources	Core	None	N/A	Total video sources installed, 1-16	Integer	No action
vertpix	Core	None	N/A	Total vertical pixels in current gmode	Integer	No action
vlevel	Core	None	N/A	Current level, 0-255	Integer	No action
vmode	Core	None	N/A	0 (native) | 1 (NTSC) | 2 (PAL)	Integer	No action
width[2]	Extended	None	N/A	Graphics width in nanoseconds	Integer	No action
xoffset	Core	None	N/A	Graphics offset in pixels	Integer	No action
yoffset	Core	None	N/A	Graphics offset in pixels	Integer	No action

[1] At least one parameter is required or an error is returned.

[2] Width is an extended parameter. *Using an unimplemented extended parameter causes error 52 (Parameter invalid for this command)*

TABLE B-9. vmGetState binary parameters.

Parameter[1]	Core or extended	Token number (decimal)	Type	Associated calling value	Associated return value	Default if parameter not used
color	Core	9	Integer	Logical color number	1 (transparent)|0 (opaque)	No action
defsource	Core	13	Integer	Any value	Default video source, 0-15	No action
dlevel	Core	17	Integer	Any value	Current level, 0-255	No action
emulation	Core	19	Integer	Any value	1 (on) | 0 (off)	No action
enable	Core	20	Integer	Any value	1 (on) | 0 (off)	No action
glevel	Core	26	Integer	Any value	Current level, 0-255	No action
gmode	Core	27	Integer	Any value	Current graphics mode	No action
horzpix	Core	28	Integer	Any value	Total horizontal pixels in current gmode	No action
logcolors	Core	32	Integer	Any value	Total available	No action
physcolors	Core	36	Integer	Any value	Total available	No action
transcolors	Core	49	Integer	Any value	Total available	No action
tsources	Core	46	Integer	Any value	Total video sources installed, 1-16	No action
vertpix	Core	50	Integer	Any value	Total vertical pixels in current gmode	No action
vlevel	Core	52	Integer	Any value	Current level, 0-255	No action
vmode	Core	53	Integer	Any value	0 (native) | 1 (NTSC) | 2 (PAL)	No action
width[2]	Extended	55	Real	Any value	Graphics width in nanoseconds	No action
xoffset	Core	60	Integer	Any value	Graphics offset in pixels	No action
yoffset	Core	66	Integer	Any value	Graphics offset in pixels	No action

[1] At least one parameter is required or an error is returned.

[2] Width is an extended parameter. Using an unimplemented extended parameter causes error 52 (Parameter invalid for this command).

TABLE B-10. Parameter values set by vmInit.

Parameter	Value	Command reference
dlevel	255 (hard keying)	vmFade
emulation	1 (on)	vmSetGraphics
enable	0 (off)	vmSetTrans
glevel	255 (full intensity)	vmFade
gmode	Current value	vmSetGraphics
horzpix	Current value	None
logcolors	System limit for gmode	None
physcolors	System default for gmode	None
state	0 (transparency off)	vmSetTrans
transcolors	System limit for gmode	None
width[1]	System default for gmode	vmSetGraphics
vertpix	Current value	None
vlevel	0 (off)	vmFade
vmode	0(native)	vmSetVideo
xoffset	0	vmSetGraphics
yoffset	0	vmSetGraphics

[1] If supported. Width is an extended parameter. Using an unimplemented extended parameter causes error 52 (Parameter invalid for this command).

TABLE B-11. vmSetGraphics parameters.

ASCII Parameters						
Parameter[1]	Core or extended	Associated calling value	Type as ASCII	Associated return value	Type as ASCII	Default if parameter not used
emulation	Core	1 (on) | 0 (off)	Integer	None	N/A	No action
gmode	Core	Mode number[2]	Integer	None	N/A	No action
width[3]	Extended	Width in nanoseconds	Integer	None	N/A	No action
xoffset	Core	Pixels	Integer	None	N/A	No action
yoffset	Core	Pixels	Integer	None	N/A	No action
Binary Parameters						
Parameter[1]	Core or extended	Token number (decimal)	Type	Associated calling value	Associated return value	Default if parameter not used
emulation	Core	19	Integer	1 (on) | 0 (off)	None	No action
gmode	Core	27	Integer	Mode number[2]	None	No action
width[3]	Extended	55	Integer	Width in nanoseconds	None	No action
xoffset	Core	60	Integer	Pixels	None	No action
yoffset	Core	66	Integer	Pixels	None	No action

[1] At least one parameter is required or an error is returned.

[2] In decimal, in MS-DOS as returned by BIOS interrupt 10H, service 0FH (see 4.3.3)

[3] Width is an extended parameter. Using an unimplemented extended parameter causes error 52 (Parameter invalid for this command).

TABLE B-12. vmSetPalette parameters.

ASCII Parameters						
Parameter	Core or extended	Associated calling value	Type as ASCII	Associated return value	Type as ASCII	Default if parameter not used
color[1]	Core	Logical color number	Integer	None	N/A	Causes error
r	Core	Red value, 0-255	Integer	None	N/A	No action
g	Core	Green value, 0-255	Integer	None	N/A	No action
b	Core	Blue value, 0-255	Integer	None	N/A	No action
Binary Parameters						
Parameter	Core or extended	Token number (decimal)	Type	Associated calling value	Associated return value	Default if parameter not used
color[2]	Core	9	Integer	Logical color number	None	Causes error
r	Core	38	Integer	Red value, 0-255	None	No action
g	Core	25	Integer	Green value, 0-255	None	No action
b	Core	4	Integer	Blue value, 0-255	None	No action
length	Core	31	Integer	Number of color array entries[3]	None	No action
array	Core	1	Pointer	Pointer to color array	None	No action

[1] Exactly one color and at least one of r, g, and b are required or an error is returned.

[2] Either exactly one color and at least one of r, g, and b are required; or color, length, and array must be used together without r, g, or b, or an error is returned.

[3] Color array entry equals 4 bytes comprised of 3 components plus a reserved byte set to NUL.

TABLE B-13. vmSetTrans parameters.

ASCII Parameters						
Parameter[1]	Core or extended	Associated calling value	Type as ASCII	Associated return value	Type as ASCII	Default if parameter not used
clear	Core	None	N/A	None	N/A	No action
color	Core	Logical color number	Integer	None	N/A	No action
enable	Core	1 (on) | 0 (off)	Integer	None	N/A	No action
state	Core	1 (on) | 0 (off)	Integer	None	N/A	No action
Binary Parameters						
Parameter[1]	Core or extended	Token number (decimal)	Type	Associated calling value	Associated return value	Default if parameter not used
clear	Core	8	N/A	None	None	No action
color	Core	9	Integer	Logical color number	None	No action
enable	Core	20	Integer	1 (on) | 0 (off)	None	No action
state	Core	42	Integer	1 (on) | 0 (off)	None	No action

[1] Either both color and state, or clear only must be used or an error is returned. Enable can be used alone or with color plus state.

TABLE B-14. vmSetVideo parameters.

ASCII Parameters						
Parameter[1]	Core or extended	Associated calling value	Type as ASCII	Associated return value	Type as ASCII	Default if parameter not used
defsource	Core	Input source, 0-15	Integer	None	N/A	No action
vmode	Core	0 (native) | 1 (NTSC) | 2 (PAL)	Integer	None	N/A	No action
Binary Parameters						
Parameter[1]	Core or extended	Token number (decimal)	Type	Associated calling value	Associated return value	Default if parameter not used
defsource	Core	13	Integer	Input source, 0-15	None	No action
vmode	Core	53	Integer	0 (native) | 1 (NTSC) | 2 (PAL)	None	No action

[1] At least one parameter is required or an error is returned.

APPENDIX B

This Page Intentionally Left Blank.

APPENDIX C

VIDEODISC (vd) COMMANDS FOR ICW PORTABILITY

10. SCOPE

10.1 Scope. This appendix describes commands that control videodisc players. Use these commands to initialize, obtain information about, and control the behavior of videodisc players connected to the system. Table C-1 lists the commands covered in this appendix, their token numbers, and their types.

10.2 Application guidance. This appendix supports implementation of the videodisc (vd) software interface and command requirements prescribed by MIL-STD-1379, Appendix D. Each of the vd commands listed on Table C-1 is described in a separate section of this appendix, beginning with Section 40, **vdGetState** command.

10.2.1 Terms, abbreviations, and acronyms used in this appendix. Key terms, abbreviations, and acronyms used in this appendix are defined as specified in Section 3 of the basic handbook.

20. APPLICABLE DOCUMENTS.

20.1 Government documents.

20.1.1 Specifications, standards, and handbooks. The following specifications, standards, and handbooks form a part of this appendix to the extent specified herein.

STANDARD

MILITARY

MIL-STD-1379 Military Training Programs

(Unless otherwise specified, copies of military specifications, standards and handbooks are available from the Standardization Documents Order Desk, Building 4D, 700 Robbins Avenue, Philadelphia, PA 19111-5094.)

APPENDIX C

30. GENERAL GUIDANCE

30.1 General information and assumptions. The general information and assumptions given in this subsection were used in the definition of the videodisc commands.

30.1.1 CAV and CLV videodisc support. Current technology uses two types of videodiscs: constant angular velocity (CAV) and constant liner velocity (CLV). These vary in the information supplied on the videodisc and the way in which they are read by the player. The individual command descriptions in this appendix identify those commands and parameters that apply to CLV videodiscs only. In addition:

a. Current support for CLV videodiscs is a subset of CAV functions. However, extended commands and parameters to provide more sophisticated CLV support may be added to future revisions of this document.

b. Compliant players support both CAV and CLV videodiscs.

30.1.2 Play and scan speeds. Play speeds are expressed as integer ratios of 1000, which is defined as the normal speed of either 25 frames per second for PAL or 30 frames per second for NTSC. For example, given that 1000 is normal, 2000 is 2 x normal, 500 is 0.5 x normal, and:

a. For players in CAV mode, applications can assume that speed 1000, at least one speed slower than 1000, and at least one speed faster than 1000 are available. For players in CLV mode, applications cannot assume play speeds other than 1000. Paragraph 30.2 explains how VDI Managements round speeds when requested speeds cannot be matched exactly by a player.

b. Scan speeds vary among players. An application should only assume that scan speed is faster than normal speed.

c. Because applications cannot assume that values other than 1000 will be matched exactly, they should not try to calculate videodisc position based on timing and frame speed at any speed other than 1000. Instead, they should use vdGetState frame (see 40.2.8).

30.1.3 Searches and instant jumps. When searching for a specific frame or chapter, players should use instant jumps if possible. If not, the search should always be at the fastest possible speed. Blanking during searches is automatic and, therefore, is not under VDI Management control.

30.1.4 Fields, frames, and chapters. All frame numbers assume a standard format of two fields per frame. The vd command set does not support accessing individual fields.

The vd command set assumes that frame numbers are always available from CAV and never from CLV, and that chapter numbers may be available from either.

30.2 Rounding methods for player speeds. Player speeds are represented by integers, with 1000 representing normal speed. Values less than 1000 represent speeds below normal, and values greater than 1000 represent speeds above normal. A value other than 1000 calls for a speed in frames per second (fps) that equals the product of the speed and the default number of frames per second divided by 1000. On a NTSC system for example, a speed of 500 specifies a rate of 0.5 x 30 fps or 15 fps. Additionally:

a. Videodisc players are limited to a finite range of speeds. If a requested speed is not 1000, VDI Management uses a rounding algorithm to translate from the specified speed to a player-supported speed. The algorithm rounds to the nearest supported speed, except that values are never rounded to 0 or 1000, except for players in CLV mode as described later in this appendix. This method lets applications guarantee use of the fastest fast and slowest slow speeds available.

   (1) Table C-2 shows how speeds are rounded for the Sony LDP-2000 videodisc player.

   (2) Table C-3 shows how speeds are rounded for the Pioneer LD-4200 videodisc player.

   (3) Speed requests of zero are errors.

b. For players in CAV mode, applications can assume that normal speed, 1000, at least one speed slower than 1000, and at least one speed faster than 1000 are available. Given this availability and the rounding algorithm, which never rounds to 0 or 1000, Table C-4 lists speed parameter values for specifying several convenient speeds without knowing the exact speeds available from a given player.

c. For players in CLV mode, applications cannot assume play speeds other than normal speed. For players that support normal speed only, all speeds other than 0 are rounded to 1000. However, if the player supports multiple speeds in CLV mode, VDI Management applies normal rounding rules.

d. After requesting a play speed, a query for that speed returns the actual speed of the videodisc player. Actual speed may differ from the requested speed because of rounding.

30.3 Multisided and multidisc applications. Many ICW applications use both sides of a videodisc or multiple disc sides. Such applications need a way to ensure that the proper

side of a disc, or the proper disc is inserted in the player. There are at least three (3) ways to do this.

30.3.1 Reference frame display. The first and simplest method for ensuring that the correct disc or disc side is inserted in the player is to display a reference frame that contains a key characteristic and ask the user to verify that the characteristic is present. If not, the application asks the user to change the side or disc, then tries again.

30.3.2 Picture stops. A second method for verifying the disc or side involves picture stops at unique frame numbers. Picture stops can be requested when the videodisc is pressed or mastered. For example:

a. Assume that the desired side or disc contains a unique picture stop at frame 125. Also assume that the videodisc service group has been initialized, and the user has been instructed to insert the proper disc. The application can:

(1) Issue **vdSet audio1=0,audio2=0,video=0** to protect the user from the following steps. (Section 100 describes the **vdSet** command.)

(2) Issue **vdPlay from=115** to start an open-ended play at frame 115 and before the picture stop. (Section 70 describes the **vdPlay** command.)

(3) Wait for a few seconds to be sure the player will pass frame 125 if no picture stop is present. (An alternative approach would have the application poll for frame numbers to see if it gets frame 125 more than once.)

(4) Issue **vdStill** to stop the player. (Section 120 describes the **vdStill** command.)

(5) Issue **vdGetState frame** to retrieve the frame number at which the player stopped. (Section 40 describes **vdGetState**.)

(6) Examine the returned frame number to determine whether or not the player stopped at frame 125, or a higher frame number.

(7) If the player stopped at frame 125, the application knows that the side or disc containing the unique picture stop is inserted in the player. If the player stopped at a higher frame number, the application knows that the wrong side or disc is inserted in the player and should prompt the user to insert the correct side or disc and try again.

b. This method is portable under the ICW portability practices because support for all involved commands and parameters is required by MIL-STD-1379, Appendix

D; and, as near as can be determined, all videodisc players support picture stops. However, note that this procedure did not use a bounded play sequence such as **vdPlay from=120,to=130** to determine whether the player stopped at frame 125 or 130. Some videodisc players do not detect picture stops during bounded plays.

c. Compliant videodisc players support detection of picture stops in unbounded plays, as indicated in MIL-STD-1379, Appendix D.

30.3.3 Chapter number search. The chapter number search method involves searching for a chapter number with a known, unique frame number. The application would issue a vdSearch chapter command, then examine the return to see if error 216 (Invalid chapter number) occurred. (Section 90 describes the vdSearch command and its parameters.) When using this method:

a. If no error is returned, the application queries for the frame number with **vdGetState frame** to verify that the chapter starts at the unique frame number.

b. Applications should avoid using this method alone, because **chapter** is an extended parameter. Therefore, this method is only portable between VDI Management implementations that support the optional **chapter** parameter.

40. vdGetState COMMAND

40.1 vdGetState command summary. **vdGetState** is a core command having a binary token number of 3078 decimal. **vdGetState** returns information about the videodisc player specified by the **device** parameter, or the default player when a device number is not specified.

40.2 Command parameters. The vdGetState command has sixteen defined parameters: **audio1**, **audio2**, cdisplay, chapter, defdevice, device, disctype, door, frame, **idxdisplay**, **motion**, **speed**, **spin**, tdevices and video. Twelve are core parameters that must be supported by compliant VDI Management implementations. There are four extended parameters; support of extended parameters is optional (see 4.3.2.6). Table C-5 and Table C-6 list ASCII and binary parameter information, respectively, for **vdGetState**.

40.2.1 Audio1 and audio2 parameters. The **audio1** and **audio2** parameters return one if the respective audio channel is on and zero if it is off.

40.2.2 Cdisplay parameter. The **cdisplay** parameter returns one if the player's chapter number display is on and zero if it is off. Using cdisplay with videodiscs that do not have chapter numbers returns error 88 (Unable to return requested information). **Cdisplay** is an extended parameter. Using an unimplemented extended parameter causes error 52 (Parameter invalid for this command).

40.2.3 Chapter parameter. The **chapter** parameter returns the current videodisc chapter number. **vdGetState chapter** returns error 86 (Device not ready) if the videodisc is not spinning normally and error 88 (Unable to return requested information) if the videodisc does not have chapter numbers. Chapter is an extended parameter. Using an unimplemented extended parameter causes error 52 (Parameter invalid for this command).

40.2.4 Defdevice parameter. The **defdevice** parameter returns the default logical player number set by vdSet defdevice or zero, the default device at start up. VDI Management directs all videodisc commands to this player unless a command contains a **device** parameter directing it to a different player.

40.2.5 Device parameter. The **device** parameter directs vdGetState to the specified logical player number regardless of the default player number set by vdSet defdevice. Because **device** only affects the command with which it is associated, the parameter does not affect the return value for defdevice when the two parameters are used together. When using this parameter:

a. Specifying **device** with no other parameter returns error 49 (Insufficient parameters).

b. Specifying a nonexistent or uninstalled player returns error 160 (Invalid device number).

c. Specifying an uninitialized player returns error 81 (Device not initialized).

40.2.6 Disctype parameter. The **disctype** parameter returns one if the videodisc is a CLV disc and zero if it is a CAV videodisc.

40.2.7 Door parameter. The **door** parameter returns one if the player door is open and zero if it is closed. VDI implementers should implement **door** for a player that supports reporting the door's status even if the player does not support opening and closing the door from an application. In addition:

a. If an implementation supports the **door** parameter but a player does not support reporting its status, VDI Management returns error 88 (Unable to return requested information).

b. Door is an extended parameter. Using an unimplemented extended parameter causes error 52 (Parameter invalid for this command).

40.2.8 Frame parameter. The **frame** parameter returns the current frame number of the videodisc player. **vdGetState frame** returns error 86 (Device not ready) if the videodisc is not spinning normally. Frame returns error 88 (Unable to return requested information) for CLV videodiscs.

40.2.9 Idxdisplay parameter. The **idxdisplay** parameter returns one if player's frame number (CAV) or time (CLV) display is on and zero if it is off.

40.2.10 Motion parameter. The **motion** parameter returns the state of a background play or scan. If the laser is reading the videodisc during a play or scan sequence -- either backward or forward -- **motion** returns one; otherwise, it returns zero. Additionally:

a. The primary intent of **vdGetState motion** is to let developers determine when an asynchronous background **vdPlay**, **vdScan**, or **vdSearch** is in progress. If **vdGetState motion** returns zero (false) during a **vdPlay** that requires a seek to a **from** (starting) frame, problems arise in determining when a play sequence has ended. Therefore, **motion** returns one (true) during the entire **vdPlay** sequence. For example, if a **vdPlay** includes both a **from** and a **to** (ending) frame , **vdGetState motion** returns one from the time the **vdPlay** returns application control until the **to** frame is reached or until an error occurs.

b. Similarly, a **vdSearch** issued without the wait parameter will return control to the application before the player reaches the target frame. If **vdSearch** is followed by **vdGetState motion**, the **motion** parameter returns one while the player is seeking

and zero when the target frame is reached. Finally, if the extended **chapter** parameter of the **vdPlay** and **vdSearch** commands is implemented, **vdGetState motion** return one as soon as the seek to the start of the chapter begins.

c. Developers can synchronize with the arrival at a from frame and the resulting display of image data without using the **motion** parameter. If no to frame is specified, **vdPlay** can include the **wait** parameter. If both the from and to parameters are needed, the **wait** parameter cannot be used with a single **vdPlay** because the command would not return until the to frame had been reached. However, the application can use either of two ways to synchronize to the start of image display. The application can poll for a window of frames surrounding the **from** frame number with **vdGetState frame**, or it can split the vdPlay into two commands. The following listings show both approaches.

(1) Polling for a from frame, then determining when a play sequence has ended.

```
vdPlay from {from frame} to {to frame}

do
        {
        vdGetState frame {frame number}
        }
while not (({from frame} - 30) << {frame number} << ({from frame} +
30))

do
        {
        something that happens at the start of video
        }
end

do
        {
        vdGetState motion {motion flag}
        }
while ({motion flag} equals 1)
```

(2) Using two **vdPlay** commands to synchronize with a **from** frame.

```
vdPlay from {from frame} wait

do
        {
```

```
        something that happens at the start of video
        }
end

vdPlay to {to frame}

do
        {
        vdGetState motion {motion flag}
        }
while ({motion flag} equals 1)
```

40.2.11 Remote parameter. The remote parameter returns one if the player's remote control unit is on and zero if it is off. Additionally:

a. If a VDI implementation supports the remote parameter but the player does not support a remote control unit, remote returns zero.

b. Remote is an extended parameter. Using an unimplemented extended parameter causes error 52 (Parameter invalid for this command).

40.2.12 Speed parameter. The speed parameter returns the actual player speed multiplied by 1000 in the play mode, or 999999 if the player is in scan mode. A return of zero indicates the player is parked or on a still frame.

40.2.13 Spin parameter. The spin parameter returns zero if the player is parked or in a transition from spun up to spun down. The spin parameter returns one if the videodisc is spinning and the player is ready to accept motion commands or in a transition from spun down to spun up.

40.2.14 Tdevices parameter. The tdevices parameter returns the total number of logical players for which VDI Management was configured during installation. If only one player is connected, it is numbered logical player zero, and tdevices returns one.

40.2.15 Video parameter. The video parameter returns one if the player's video channel is on and zero if it is off.

40.2.16 Parameters resulting in errors. If a parameter causes an error, vdGetState returns immediately with the error message. The command does not return partial responses for other parameters that did not cause errors.

40.3 Implementation notes. When using vdGetState, note the following:

a. vdGetState can be successfully issued any time VDI Management can accept commands, regardless of the current state of the player. However, the player's state can affect the ability to return specific parameter values, and could cause errors. For example, vdGetState frame returns error 88 (Unable to return requested information) if the player is parked.

b. Trying to queue vdGetState causes error 177 (Command cannot be queued) at the time of the attempt.

40.4 Return values.

40.4.1 ASCII returns. On success, the return is a comma-separated list of values for requested parameters as described in 40.2, above. On failure, the return is "ERROR n...".

40.4.2 Binary returns. On success, the return is AX = 0. Values associated with requested parameters are 32-bit values of the types given in binary parameter Table C-6. On failure, the return is AX = error number. Any return values in the parameter block addressed by ES:DI are undefined and should be ignored.

40.5 Related commands. The syGetState, vdInit, vdSet, vmGetState, and xyGetState command requirements should also be reviewed in relation to implementing the vdGetState command.

40.6 Examples. The following are ASCII and binary examples of the vdGetState command.

40.6.1 ASCII.

Get status of player 2 motion flag	vdGetState device=2, motion (returns) "1"	; player 2 is in ; play or scan ; mode
Get information on door and spin state for current player	vdGetState door,spin (returns) "1,0"	; door is open and ; player is spun ; down.
Get whether disc index display is on for current player.	vdGetState idxdisplay (returns) "1"	; videodisc index ; display is on
Get whether player 1 video is on and the currently selected player	vdGetState device=1,video,defdevice (returns) "1,2"	; Player 1 video ; is on, and ; default player ; is logical ; number 2

40.6.2 Binary.

Get status of motion flag	AX	3078	; vdGetState decimal ID
	BX	1	; number of parameters
	ES:DI[0]	35	; motion decimal ID
	ES:DI[4]	Any value	; place holder for return value
After return	AX	0	; *returns 0 if successful and* ; non-zero if not.
	ES:DI[4]	1	; value for motion, player is ; playing or scanning.

50. vdInit COMMAND

50.1 vdInit command summary. vdInit is a core command having a binary token number of 3079 decimal. **vdInit** initializes videodisc hardware and the vd software service group, placing both in a known state. **vdInit** must be issued for each attached player that will be used by the application. This command interrupts any other player motion command that did not include a wait parameter, in which case, the application is not able to issue **vdInit** until the motion command is complete. In addition:

a. **vdInit** is a synchronous command. It does not return control to the application until it has succeeded or detected an error condition. To keep disturbances to a minimum, VDI Management should turn video and audio off at the player, spin up the videodisc, then turn video and audio back on.

b. The resulting display after vdInit varies with videodisc type. With CAV videodiscs, video remains visible with the player frozen on the first available frame. With most CLV videodiscs, the player automatically blanks video.

50.2 Command parameters. The vdInit command has one defined parameter: **device**. This is a core parameter that must be supported by compliant VDI Management implementations. Table C-7 lists **vdInit** ASCII and binary parameter information.

50.2.1 Device parameter. The **device** parameter specifies the logical player number to be initialized. If **device** is omitted, **vdInit** either initializes logical device zero or reinitializes the default player set by **vdSet defdevice**. When using **device**:

a. A player must have been previously initialized with **vdInit** for **vdSet defdevice** to set it to the default. If **vdSet** has not been used and the **device** parameter is omitted, the default player is defined to be logical device zero. On subsequent calls that include a **device** parameter, **vdInit** does not change the default device if a **device** other than the default is specified.

b. If, on the first call to **vdInit**, a **device** other than zero is specified, **vdSet** must be used if that device is the desired default. For example, assume that the first call to **vdInit** is **vdInit device=1**. This does not set the default device to one. An application must issue **vdSet defdevice=1** to do this.

c. To change the default to a device that has not yet been initialized, use **vdInit device=n**, where "n" is the desired logical device number, followed by **vdSet defdevice=n**. After a device has been initialized, **vdSet defdevice** alone may be used to change the default.

d. Specifying a nonexistent or uninstalled player causes error 160 (Invalid device number).

50.2.2 Conditions set by vdInit. vdInit sets the videodisc parameters listed in Table C-8 to the specified values.

50.3 Implementation notes. When using vdInit, note the following:

a. vdInit can be successfully issued any time the specified device or, without a specified device, the default player (either player zero or the player set by vdSet defdevice) can accept motion commands (except, see 50.3d).

b. vdGetState tdevices returns the total number of players for which VDI Management was installed. This command can be used after the first vdInit to determine the number of additional devices that can be initialized.

c. If the player supports a character generator, vdInit turns it off.

d. Trying to queue vdInit causes error 177 (Command cannot be queued) at the time of the attempt.

e. With systems that do not support the door parameter, VDI Management returns error 80 (Initialization error) if an application issues vdInit with the player door open. Therefore, it is good programming practice to prompt the user to insert the videodisc and close the door before issuing vdInit.

f. Spinning the videodisc up also updates the disctype parameter, and sets cdisplay to undefined for videodiscs that do not support chapter numbers.

g. If vdInit returns an error, the parameters listed in Table C-8 have undefined values.

50.4 Return values.

50.4.1 ASCII returns. On success, the return is "OK". On failure, the return is "ERROR...n".

50.4.2 Binary returns. On success, the return is AX = 0. On failure, the return is AX = error number.

50.5 Related commands. The sylnit, vdGetState, vdSet, vmInit, and xylnit command requirements should also be reviewed in relation to implementing the vdInit command.

50.6 Examples. The following are ASCII and binary examples of the vdInit command.

50.6.1 ASCII.

Initialize player 0 and vd service group	vdInit (returns) "OK"	; first time command is ; issued
Initialize player 1	vdInit device=1 (returns)"OK"	

50.6.2 Binary.

Initialize player 0 and vd service group	AX	3079	; vdInit decimal ID
	BX	0	; number of parameters
After return	AX	0	; returns zero if successful, and ; non-zero if not.
Initialize player 1	AX	3079	; vdInit decimal value
	BX	1	; number of parameters
	ES:DI[0]	14	; device decimal ID
	ES:DI[4]	1	; logical player number for ; device
After return	AX	0	; returns 0 if successful, and ; non-zero if not

60. vdPassThru COMMAND

60.1 vdPassThru command summary. vdPassThru is a core command having a binary token number of 3080 decimal. **vdPassThru** communicates directly with a player, bypassing the standard videodisc service group commands and parameters. It is furnished to allow access to specific player features that are not supported by other VDI videodisc commands. **vdPassThru** passes a string of bytes to the player and waits for the player's response for a specified or default time. If a response is issued, VDI Management returns it to the application. When using **vdPassThru**:

a. Some videodisc players may have more than one command set that can be used to transmit commands to the player. The application sets the appropriate command mode when using **vdPassThru**.

b. When an application issues **vdPassThru**, VDI Management stops normal communications using preset parameters to let the application change player communications parameters as necessary. VDI Management does not resume normal communications until it receives a command from the vd group other than **vdPassThru**. VDI Management then resets the command mode and communications parameters for the player to the normal settings.

CAUTION

vdPassThru allows nonportable access to special features of videodisc players. It is included as a convenience to developers and implementers who want to use the IMA/DoD command set for portable ICW applications and do not want to switch to a different environment for applications that require access to nonportable player functions not furnished by other vd service group commands. Therefore, although the command is required in compliant applications, it is supplied for convenience only and SHOULD NOT be used for portable ICW applications. An application that uses **vdPassThru** cannot be compliant unless it properly handles a failure of a player to act properly on the passed-through command string.

60.2 Command parameters. The vdPassThru command has three defined parameters: **device**, **pmsg** and **time**. All are core parameters that must be supported by compliant VDI Management implementations. Table C-9 lists ASCII and binary parameter information for **vdPassThru**.

60.2.1 Device parameter. The **device** parameter directs vdPassThru to the specified logical player number regardless of the default player number set by vdSet defdevice. Specifying a nonexistent or uninstalled player causes error 160 (Invalid device number). Specifying an uninitialized player causes error 81 (Device not initialized).

60.2.2 Pmsg parameter. The pmsg parameter denotes a command string to be passed through to the player without modification by VDI Management. In addition:

a. The binary interface passes a pointer to a null-terminated player message string. Each byte in the string can take the value 1 through 255.

b. The binary interface data block for command and response strings allocated by the application and pointed to by the pmsg parameter must be a minimum of 33 bytes. VDI Management uses this memory for the player response as well as the player command. Because the response may be up to 32 bytes plus NULL, allocating less than 33 bytes could result in the binary interface response string over-writing application memory.

c. For the ASCII interface, the digit string consists of pairs of hexadecimal digits. The string can not contain spaces (ASCII 20H) because the space character is a delimiter and would denote the end of the string. Also:

(1) Any non-delimiter characters other than the digits "0" through "9", and the letters "A" though "F" and "a" through "f" cause error 51 (Parameter value invalid or out of range). A string containing an odd number of digits also causes this error.

(2) The inadvertent insertion of a delimiter, such as a space, is likely to cause error 52 (Parameter invalid for this command) because VDI Management will interpret following characters as the start of a new parameter name which is highly unlikely to be valid.

60.2.3 Time parameter. The time parameter specifies a timeout value in milliseconds. VDI Management waits for time milliseconds for a player response. If the player does not respond in time milliseconds, VDI Management returns control to the application. Additionally, when using this parameter:

a. If no time is specified, the default timeout value is implementation- dependent. An application may specify a time of 0 milliseconds for players that typically do not issue responses to commands. Values are likely to be approximate but very close to specified values in actual implementations because of limitations in computer timing.

b. A failure by the player to respond within the specified or default timeout period results in a null response string. A timeout does not cause an error.

60.3 Return values.

60.3.1 ASCII returns. On success, the return is the player response as a string of up to 64 upper-case hexadecimal digits (0-9, A-F; 32 bytes of data) terminated by CR/LF, or an empty string (CR/LF only) if a timeout occurs. On failure, the return is "ERROR n...".

60.3.2 Binary returns. On success, the return is AX = 0. Value associated with the **pmsg** parameter is a 32-bit pointer to a player response string of up to 32 bytes + NULL. **vdPassThru** does not change the **pmsg** value passed back to the application. The response string may be an empty string (NULL only) if a timeout occurs. On failure, the return is AX = error number. Any return values in the parameter block addressed by ES:DI are undefined and should be ignored.

60.4 Examples. The following are ASCII and binary examples of the **vdPassThru** command.

60.4.1 ASCII.

Send "THIS IS A COMMAND" to the player and return "THIS IS A RESPONSE"	vdPassThru pmsg = 5448495320495320412043 4F4D4D414E44	; command string is not quoted and ; contains no spaces.
	(returns) "5448495320495320412052455350 4F4E5345"	; response string is not quoted.

60.4.2 Binary.

Send a command string to player 2	AX	3080	; vdPassThru decimal ID
	BX	2	; number of parameters
	ES:DI[0]	14	; device decimal ID
	ES:DI[4]	1	; send message to player 1
	ES:DI[8]	37	; pmsg decimal ID
	ES:DI[C]	pointer	; pointer to player message ; string
After return			
	AX	0	; returns 0 if successful, and ; non-zero if not
	ES:DI[C]	pointer	; pointer to player response ; string (Same pointer value as ; the passed value)

70.0 vdPlay COMMAND

70.1 vdPlay command summary. vdPlay is a core command having a binary token number of 3081 decimal. **vdPlay** executes videodisc play sequences. The sequences may include starting frames, ending frames, chapters, directions, and speeds in various combinations. The application can instruct VDI Management to return control immediately or when the play sequence is complete. This command interrupts any other player motion command that did not include a wait parameter, in which case, the application will not be able to issue **vdPlay** until the motion command is complete.

70.2 Command parameters. The vdPlay command has seven defined parameters: **chapter**, **device**, **direction**, **from**, **speed**, **to**, and **wait**. All parameters are core parameters that must be supported by compliant VDI Management implementations, except **chapter** which is an extended parameter. Implementation support for extended parameters is optional (see 4.3.2.6). Table C-10 lists ASCII and binary parameter information for **vdPlay**.

70.2.1 No parameters. **vdPlay** issued with no parameters causes the player to start playing forward from the current frame at a speed of 1000 (see 70.2.6) and continue until interrupted by a subsequent vdInit, vdPlay, vdScan, vdSearch, vdSet **spin**=0, vdStep, or **vdStill** command; or until the player reaches the end of the videodisc.

70.2.2 Chapter parameter. The **chapter** parameter specifies a chapter number to play from beginning to end. When used with a **speed** parameter, the chapter plays at the specified speed. Adding **wait** causes VDI Management to wait until the chapter has been played to return application control. Also:

a. Specifying a **chapter** for a videodisc without chapter numbers causes error 208, (Action not supported by disc). Specifying an illegal **chapter** number causes error 216 (Invalid chapter number).

b. **Chapter** is an extended parameter: implementation is optional. Using an unimplemented extended parameter causes error 52 (Parameter invalid for this command).

70.2.2.1 Compatible parameters. The **chapter** parameter can be issued with **device**, **speed**, and **wait** parameters. Using **chapter** with any other parameters cause error 50 (Parameters cannot be used together).

70.2.3 Device parameter. The **device** parameter directs vdPlay to the specified logical player number regardless of the default player number set by vdSet **defdevice**. Specifying a nonexistent or uninstalled player causes error 160 (Invalid device number). Specifying an uninitialized player causes error 81 (Device not initialized).

70.2.3.1 Compatible parameters. The **device** parameter is compatible with all other **vdPlay** parameters, assuming other parameters are compatible with each other.

70.2.4 Direction parameter. The **direction** parameter sets the direction of motion (1 = forward, 0 = backward) for play sequences that do not include to frames. When using the **direction** parameter:

a. Specifying a **direction** with no **from** frame starts a play sequence in the specified **direction** from the current frame at an optional **speed**. If a **from** frame is specified, the player searches to the specified frame, then begins play in the specified **direction**.

b. Specifying a **direction** with a **to** frame causes error 50 (Parameters cannot be used together) because the direction required to reach the **to** frame is predetermined either by the relative position of the current frame or, if specified, the relative position of the **from** frame. Therefore, a **direction** used with a **to** frame is redundant if it agrees with the predetermined direction, and conflicting if it opposes the predetermined direction.

70.2.4.1 Compatible parameters. The **direction** parameter is compatible with the **from**, **device**, **speed**, and **wait** parameters. Using the direction parameter with other **vdPlay** parameters causes error 50 (Parameters cannot be used together).

70.2.5 From parameter. The **from** parameter specifies the starting frame number for a play sequence. The player immediately searches or jumps to the specified frame with video off, turns video on, and executes the play sequence at an optional **speed**. The play executes either to an optional **to** frame or in an optional **direction**. Note that **vdPlay from = 1000,to = 1000** is exactly the same as **vdSearch frame = 1000**. In addition:

a. Using the **from** parameter without a **to** parameter starts an unbounded play. The play sequence continues until interrupted by a subsequent **vdInit**, **vdPlay**, **vdScan**, **vdSearch**, **vdSet spin = down**, **vdStep**, or **vdStill** command, or until the player reaches the edge of the videodisc.

b. Specifying a **from** frame for a CLV videodisc causes error 208 (Action not supported by disc). Specifying an illegal frame number causes error 215 (Invalid frame number).

70.2.5.1 Compatible parameters. The **from** parameter is compatible with the **device**, either **direction** or **to**, **speed**, and **wait** parameters. Using the **from** parameter with other **vdPlay** parameters or with illegal combinations of otherwise compatible parameters causes error 50 (Parameters cannot be used together).

70.2.6 Speed parameter. The speed parameter specifies the speed of play with 1000 equaling normal. VDI Management maps requested speeds as closely as possible to available player speeds (see 30.1.2). In addition:

a. Because actual speeds may vary from requested speeds, the binary interface changes the speed value passed in the parameter block to the actual speed set by VDI Management. Issuing **vdGetState speed** returns the actual speed after any necessary rounding. For CAV mode, speeds are never rounded to 0 or 1000 (see 30.1.2).

b. Specifying a speed less than or equal to zero causes error 51 (Parameter value invalid or out of range).

70.2.6.1 Compatible parameters. The **speed** parameter is compatible with the **device**, either **direction** or to, from, chapter, and wait parameters. However, neither the **from** or to parameter should be used in conjunction with **chapter** (see 70.2.2). Using the **speed** parameter with other parameters or illegal combinations of compatible parameters causes error 50 (Parameters cannot be used together).

70.2.7 To parameter. The to parameter specifies the ending frame number for a play sequence. When the player reaches the to frame, the player automatically enters still mode, freezing on and displaying the frame. Also:

a. The to parameter has lower priority than the **from** parameter. For example, **vdPlay from=100,to=1000** and **vdPlay to=1000,from=100** both search to frame 100, then play to frame 1000.

b. Specifying a to frame for a CLV videodisc caused error 208 (Action not supported by disc). Specifying an illegal frame number causes error 215 (Invalid frame number).

70.2.7.1 Compatible parameters. The to parameter is compatible with the **device**, **from**, **speed**, and **wait** parameters. Using to with other parameters causes error 50 (Parameters cannot be used together).

70.2.8 Wait parameter. The effect of the **wait** parameter depends on the parameters that accompany it. Table C-11 lists when **vdPlay wait** returns application control based on accompanying parameters. If the player returns an error, the command will return when the error state is detected instead of at the time given in Table C-11. Additionally:

a. Without **wait**, VDI Management returns control as soon as it determines that the command is legal. No error checking is done to determine if the player actually accepts the command or acts on it properly. Therefore, **syCheckError** should be

used to determine if the player entered an error state while either accepting or trying to execute the command.

b. **syCheckError** may be needed to detect certain error states that occur after **vdPlay wait**. For example, **vdPlay wait,from=1000** returns when the **from** frame has been reached. **syCheckError** is required to detect any error state that occurs after the player has reached frame 1000.

c. Without **wait**, a subsequent vdInit, vdPlay, vdScan, **vdSearch**, **vdSet spin=0**, **vdStep** or **vdStill** command immediately interrupts an executing play sequence, even if the play sequence specifies a target **to** frame or a chapter.

70.2.8.1 Compatible parameters. The **wait** parameter is compatible with all other **vdPlay** parameters, assuming the other parameters are compatible with each other.

70.3 Implementation notes. *When using* **vdPlay**, *note the following:*

a. **vdPlay** can be successfully issued any time **spin=1** (up) as set by vdSet.

b. **vdGetState motion** can be used to find out whether the player is currently executing a play sequence. This could be used, for example, with a vdPlay to... with no **wait** parameter to determine whether or not the to frame has been reached.

c. All parameters apply to the current **vdPlay** they were issued with only. For example, **vdPlay to=1000,speed=500** does not set the speed to a default of 500: a subsequent vdPlay without a speed will play at speed 1000, not 500.

70.4 Return values.

70.4.1 ASCII returns. On success, the return is "OK". On failure, the return is "ERROR n...".

70.4.2 Binary returns. On success, the return is AX = 0. Value associated with **speed** parameter is a 32-bit integer that gives the actual speed that will be set after rounding, if rounding is required. On failure, the return is AX = error number.

70.5 Related commands. The syCheckError, **vdGetState**, **vdScan**, **vdSearch**, **vdStep**, and **vdStill** command requirements should also be reviewed in relation to implementing the **vdPlay** command.

70.6 Examples. The following are ASCII and binary examples of the **vdPlay** command.

70.6.1 ASCII.

Play forward at normal speed from current frame.	**vdPlay** (returns)"OK"	
Play backward from frame 1000 at the slowest possible speed.	**vdPlay** **speed=1,direction=0,from=1000** (returns)"OK"	
Play from current frame to 2000, do not return until it is reached.	**vdPlay to=2000,wait** (returns)"OK"	
Play backward to frame 1.	**vdPlay to=1,direction=0** (returns)"ERROR 50"	; Parameters cannot be ; used together.
Play backward from 200 to 100	**vdPlay from=200,to=100** (returns)"OK"	
Play backward from the current frame	**vdPlay direction=0** (returns)"OK"	
Play all of chapter 2 at normal speed	**vdPlay chapter=2** (returns)"OK"	

70.6.2 Binary.

Play from frame 200 to 500	AX	3081	; vdPlay decimal ID
	BX	2	; number of parameters
	ES:DI[0]	24	; from decimal ID
	ES:DI[4]	200	; starting frame number
	ES:DI[8]	48	; to decimal ID
	ES:DI[C]	500	; ending frame number
After return	AX	0	; returns zero if successful, and ; non-zero if not.

80. vdScan COMMAND

80.1 vdScan command summary. vdScan is a core command having a binary token number of 3083 decimal. **vdScan** places the default or specified player in scan mode in an optional **direction**. The player plays at the maximum possible speed. The command continues until interrupted by a subsequent **vdInit**, **vdPlay**, **vdScan**, **vdSearch**, **vdSet spin=0**, **vdStep**, or **vdStill** command, or until the player reaches the edge of the videodisc. This command interrupts any other player motion command that did not include a **wait** parameter, in which case, the application is not able to issue **vdScan** until the motion command is complete.

80.2 Command parameters. The vdScan command has three defined parameters: **device**, **direction**, and **wait**. All are core parameters that must be supported by compliant VDI Management implementations. Table C-12 lists **vdScan** ASCII and binary parameter information.

80.2.1 vdScan with no parameters. **vdScan** issued with no parameters starts scanning forward from the current frame.

80.2.2 Device parameter. The **device** parameter directs vdScan to the specified logical player number regardless of the default player number set by **vdSet defdevice**. Specifying a nonexistent or uninstalled player causes error 160 (Invalid device number). Specifying an uninitialized player causes error 81 (Device not initialized).

80.2.3 Direction parameter. The **direction** parameter sets the direction of motion for the scan, either one (forward) or zero (backward).

80.2.4 Wait parameter. The **wait** parameter causes VDI Management to wait until it has confirmed that the player is in scan mode to return application control. Without **wait**, VDI Management returns control as soon as it determines that the command is legal. No error checking is done to determine if the player actually accepts the command or acts on it properly. Therefore, **syCheckError** should be used to determine if the player entered an error state while either accepting or trying to execute the command.

80.3 Implementation notes. When using **vdScan**, note that the command is typically used during application development only. This is because **vdScan** often results in displaying parts of frames and does not accept parameters to limit the area of the videodisc that is scanned.

80.4 Return values.

80.4.1 ASCII returns. On success, the return is "OK". On failure, the return is "ERROR n...".

80.4.2 Binary returns. On success, the return is AX = 0. On failure, the return is AX = error number.

80.5 Related commands. The vdPlay, vdInit, vdScan, vdSearch, vdSet, vdStep, and vdStill commands should also be reviewed in relation to implementing the vdScan command.

80.6 Examples. The following are ASCII and binary examples of the vdScan command.

80.6.1 ASCII.

Scan forward from the current frame	vdScan (returns)"OK"
Scan backward on player 1, do not return until scan mode is confirmed	vdScan direction=0,device=1,wait (returns)"OK"

80.6.2 Binary.

Scan backward on the default player	AX	3083	; vdScan decimal ID
	BX	2	; number of parameters
	ES:DI[0]	15	; direction decimal ID
	ES:DI[4]	0	; play direction (backward)
After return			
	AX	0	; returns zero if successful, and ; non-zero if not.

90. vdSearch COMMAND

90.1 vdSearch command summary. vdSearch is a core command having a binary token number of 3084 decimal. **vdSearch** causes the player to turn video off, immediately search for the specified frame number or the first frame of the specified chapter number, and freeze. This command interrupts any other player motion command that did not include a **wait** parameter, in which case, the application is not able to issue **vdSearch** until the motion command is complete. The resulting display after **vdSearch** varies with videodisc type. For example:

a. With CAV videodiscs, video remains visible.

b. With most CLV videodiscs, **vdSearch** is equivalent to searching to the start of a chapter followed by a pause command. Typically, a CLV pause command automatically blanks video.

90.2 Command parameters. The vdSearch command has four defined parameters: **chapter**, **device**, **frame**, and **wait**. Device, frame, and **wait** are core parameters that must be supported by compliant VDI Management implementations. **Chapter** is an extended parameter; VDI implementation support of extended parameters is optional (see 4.3.2.6). Table C-13 lists ASCII and binary parameter information for **vdSearch**.

90.2.1 Chapter parameter. The **chapter** parameter specifies a chapter number to search to. The player displays the first frame of the specified chapter (CAV) or pauses and blanks video (CLV). Also:

a. Specifying a **chapter** for a videodisc without chapter numbers causes error 208 (Action not supported by disc). Specifying an illegal chapter number causes error 216 (Invalid chapter number).

b. **Chapter** is an extended parameter. Using an unimplemented extended parameter causes error 52 (Parameter invalid for this command).

90.2.2 Device parameter. The **device** parameter directs **vdSearch** to the specified logical player number regardless of the default player number set by **vdSet defdevice**. Specifying a nonexistent or uninstalled player causes error 160 (Invalid device number); specifying an uninitialized player causes error 81 (Device not initialized).

90.2.3 Frame parameter. The **frame** parameter specifies a frame number to search to. The player displays the specified frame. Specifying a frame for a CLV videodisc causes error 208 (Action not supported by disc). Specifying an illegal frame number causes error 215 (Invalid frame number).

90.2.4 Wait parameter. The wait parameter causes VDI Management to wait until the specified chapter or frame has been reached before returning application control. Without wait, VDI Management returns control as soon as it determines that the command is legal. No error checking is done to determine if the player actually accepts the command or acts on it properly. Therefore, syCheckError should be used to determine if the player entered an error state while either accepting or trying to execute the command.

90.3 Return values.

90.3.1 ASCII returns. On success, the return is "OK". On failure, the return is "ERROR n...n".

90.3.2 Binary returns. On success, the return is AX = 0. On failure, the return is AX = error number.

90.4 Related commands. The vdPlay, vdSet, vdStep, and vdStill command requirements should also be reviewed in relation to implementing the vdSearch command.

90.5 Examples. The following are ASCII and binary examples of the vdSearch command.

90.5.1 ASCII.

Search for frame 23476	vdSearch frame=23476 (returns)"OK"
Search for chapter 5 on player 1, do not return until the chapter has been reached	vdSearch chapter=5,device=1,wait (returns)"OK"

90.5.2 Binary.

Search for frame 10356	AX	3084	; vdSearch decimal ID
	BX	1	; number of parameters
	ES:DI[0]	23	; frame decimal ID
	ES:DI[4]	10356	; frame number to search for
After return	AX	0	; returns 0 if successful, and ; non-zero if not

100. vdSet COMMAND

100.1 vdSet command summary. **vdSet** is a core command having a binary token number of 3085 decimal. **vdSet** sets the default logical player number and other player conditions, including the state of the audio and video channels, the index and chapter number displays, the disc spin/park status, whether the door is open or closed, and whether the user remote control is on or off.

100.2 Command parameters. The **vdSet** command has eleven defined parameters: **audio1**, **audio2**, **cdisplay**, **defdevice**, **device**, **door**, **idxdisplay**, **remote**, **spin**, **video**, and **wait**. The **cdisplay**, **door**, and **remote** parameters are extended; implementation of extended parameters is optional (see 4.3.2.6). All other parameters are core parameters that must be supported by compliant VDI Management implementations. Table C-14 and Table C-15 lists **vdSet** ASCII and binary parameter information, respectively.

100.2.1 Audio1 and audio2 parameters. The **audio1** and **audio2** parameters enable (1) and disable (0) the player's stereo outputs. Setting both **audio1** and **audio2** to zero turns off all player audio. Note that many players automatically route the output of an enabled audio channel to a disabled channel. For example, if **audio1** = 0 and **audio2** = 1, the player may automatically route the output of **audio2** to **audio1**.

100.2.2 Cdisplay parameter. The **cdisplay** parameter enables and disables the player's chapter number display. This display is typically a character generator within the videodisc player that displays chapter number information as part of the video signal. In addition:

a. **Cdisplay** is an extended parameter. Using an unimplemented extended parameter causes error 52 (Parameter invalid for this command).

b. Simultaneous display of chapter numbers, if implemented, and the position index (see 100.2.6) is not a requirement. The position and formats of these displays is typically player dependent and should, therefore, be avoided in finished applications.

100.2.3 Defdevice parameter. The **defdevice** parameter sets the default logical player number. VDI Management directs all videodisc commands to this player number unless a command contains a **device** parameter (see 100.2.4) directing it to a different player number.

100.2.4 Device parameter. The **device** parameter directs **vdSet** to the specified logical player number regardless of the default player number set by **vdSet defdevice**. Specifying a nonexistent or uninstalled player causes error 160 (Invalid device number). Specifying an uninitialized player causes error 81 (Device not initialized).

100.2.5 Door parameter. The **door** parameter opens and closes the videodisc player door. If the player does not support this function, the parameter returns error 87 (Action not supported by device). Additionally:

a. VDI Management developers should implement **door** for a player that supports reporting the door's status even if the player does not support opening and closing the door from an application.

b. **Door** is an extended parameter. Using an unimplemented extended parameter causes error 52 (Parameter invalid for this command).

100.2.6 Idxdisplay parameter. The **idxdisplay** parameter enables and disables the player's position index display. The resulting display is in frame numbers for CAV videodiscs and time for CLV videodiscs. This display is typically a character generator within the videodisc player that displays videodisc position as part of the video signal (see 100.2.2b).

100.2.7 Remote parameter. The **remote** parameter turns the hand held remote on and off. **Remote**=0 (off) gives the application software complete control over the videodisc player. If the player does not support this function, **remote** returns error 87 (Action not supported by device). **Remote** is an extended parameter. Using an unimplemented extended parameter causes error 52 (Parameter invalid for this command).

100.2.8 Spin parameter. The **spin** parameter spins the disc up and down. In addition:

a. **Spin**=1 (up) causes the player to spin up and still on frame 1 (or the first available frame). Spinning the videodisc up also updates the **disctype** parameter and sets **cdisplay** to undefined for videodiscs that do not support chapter numbers.

b. **Spin**=0 (down) causes the player to spin down immediately, interrupting any player motion command not accompanied by the **wait** parameter, in which case, the application is not able to issue **vdSet spin**=0 until the motion command is complete.

100.2.9 Video parameter. The **video** parameter enables and disables the player's video output channel.

100.2.10 Wait parameter. With the **wait** parameter, **vdSet** does not return application control until the specified settings have been acknowledged or a player error state is detected. Without **wait**, VDI Management returns control as soon as it determines that the command is legal. Also:

a. Without wait, no error checking is done to determine if the player actually accepts the command or acts on it properly. Therefore, syCheckError should be used to determine if the player entered an error state while either accepting or trying to execute the command.

b. Specifying wait with no other parameter results in error 49 (insufficient parameters).

100.3 Implementation notes. When using vdSet, note that applications can issue vdSet successfully any time the player can accept commands, except that vdSet door=1 (open) can only be issued following vdSet spin=0 (down) and after the player has actually completed the spin down sequence.

100.4 Return values.

100.4.1 ASCII returns. On success, the return is "OK". On failure, the return is "ERROR n...".

100.4.2 Binary returns. On success, the return is AX = 0. On failure, the return is Ax = error number.

100.5 Related commands. The syCheckError and vdGetState command requirements should also be reviewed in relation to implementing the vdSet command.

100.6 Examples. The following are ASCII and binary examples of the vdSet command.

100.6.1 ASCII.

Turn off both number displays	vdSet idxdisplay=0,cdisplay=0 (returns)"OK"
Disable hand-held remote control on player 1	vdSet remote=0,device=1 (returns)"OK"
Make logical player 1 the default	vdSet defdevice=1 (returns)"OK"
Turn on all player outputs	vdSet video=1,audio1=1,audio2=1 (returns)"OK"
Disable audio channel 1	vdSet audio1=0 (returns)"OK"

100.6.2 Binary.

Turn on player index display	AX	3085	; vdSet decimal ID
	BX	1	; number of parameters
	ES:DI[0]	29	; idxdisplay decimal ID
	ES:DI[4]	1	; set value to 1 (on)
After return	AX	0	; returns 0 if successful, and ; non-zero if not.
Set video on, audio channel 1 on, audio channel 2 off	AX	3085	; vdSet decimal ID
	BX	3	; number of parameters
	ES:DI[0]	51	; video decimal ID
	ES:DI[4]	1	; set value to 1 (on)
	ES:DI[8]	2	; audio1 decimal ID
	ES:DI[C]	1	; set value to 1 (on)
	ES:DI[10]	3	; audio2 decimal ID
	ES:DI[14]	0	; set value to 0 (off)
After return	AX	0	; returns 0 if successful, and ; non-zero if not

110. vdStep COMMAND

110.1 vdStep command summary. vdStep is a core command having a binary token number of 3090 decimal. **vdStep** causes the videodisc player to move forward or backward one frame at a time in a specified **direction** without blanking the screen. The command does not return application control until the step is complete. This command interrupts any other player motion command that did not include a **wait** parameter, in which case, the application is not able to issue vdPlay until the motion command is complete.

110.2 Command parameters. The vdStep command has two defined parameters; **device** and **direction**. Both are core parameters that must be supported by compliant VDI Management implementations. Table C-16 lists ASCII and binary parameter information for **vdStep**.

110.2.1 No parameters. With no parameters, **vdStep** steps forward one frame and freezes.

110.2.2 Device parameter. The **device** parameter directs vdStep to the specified logical player number regardless of the default player number set by **vdSet defdevice**. Specifying a nonexistent or uninstalled player causes error 160 (Invalid device number). Specifying an uninitialized player causes error 81 (Device not initialized).

110.2.3 Direction parameter. The **direction** parameter sets the direction of motion for the step, either one (forward) or zero (backward).

110.3 Implementation notes. When using **vdStep** in applications, it can be successfully issued any time that **spin**=1 (up) as set by vdSet. However, issuing vdStep while a vdPlay sequence is in progress is not recommended because of the difficulty in determining which frames will be displayed.

110.4 Return values.

110.4.1 ASCII returns. On success, the return is "OK". On failure, the return is "ERROR n...".

110.4.2 Binary returns. On success, the return is AX = 0. On failure, the return is AX = error number.

110.5 Related commands. **syCheckError**, **vdPlay**, and **vdSet** command requirements should also be reviewed in relation to implementing the **vdStep** command.

110.6 Examples. The following are ASCII and binary examples of the **vdStep** command.

110.6.1 ASCII.

Step forward 1 frame	vdStep (returns)"OK"
Step backward 1 frame	vdStep direction=0 (returns)"OK"

110.6.2 Binary.

Step forward one frame	AX	3090	; vdStep decimal command ID
	BX	0	; number of parameters
After return	AX	0	; returns 0 if successful, and ; non-zero if not

120. vdStill COMMAND

120.1 vdStill command summary. vdStill is a core command having a binary token number of 3091 decimal. **vdStill** causes the videodisc player to immediately stop on the current frame and sets the **motion** parameter returned by **vdGetState** to zero. In addition:

a. This command interrupts any other player motion command that did not include a **wait** parameter, in which case, the application will not be able to issue **vdStill** until the motion command is complete.

b. The resulting display after vdStill varies with videodisc type. With CAV videodiscs, video remains visible. With most CLV videodiscs, vdStill is equivalent to a pause command and the player automatically blanks video.

120.2 Command parameters. The vdStill command has one defined parameter; **device**. Device is a core parameter that must be supported by compliant VDI Management implementations. Table C-17 lists ASCII and binary parameter information for **vdStill**.

120.2.1 Device parameter. The **device** parameter directs vdStill to the specified logical player number regardless of the default player number set by **vdSet defdevice**. Specifying a nonexistent or uninstalled player causes error 160 (Invalid device number): specifying an uninitialized player causes error 81 (Device not initialized).

120.3 Return values.

120.3.1 ASCII returns. On success, the return is "OK". On failure, the return is "ERROR n...".

120.3.2 Binary returns. On success, the return is AX = 0. On failure, the return is AX = error number.

120.4 Related commands. **vdPlay**, **vdScan**, and **vdSet** command requirements should also be reviewed in relation to implementing the **vdStill** command.

120.5 Examples. The following are ASCII and binary examples of the VdStill command.

120.5.1 ASCII.

Stop player motion	vdStill (returns) "OK"

120.5.2 Binary.

Stop player motion	AX	3091	; vdStill decimal ID
	BX	0	; number of parameters
After return			
	AX	0	; returns 0 if successful, and ; non-zero if not

TABLE C-1. Videodisc (vd) commands summary.

ASCII command name[1]	Binary interface token number (decimal)	Type[2]
vdGetState	3078	Core
vdInit	3079	Core
vdPassThru	3080	Core
vdPlay	3081	Core
vdScan	3083	Core
vdSearch	3084	Core
vdSet	3085	Core
vdStep	3090	Core
vdStill	3091	Core

[1] Upper or lower case for command names is not significant.

[2] Compliant implementations must support "Core" commands.

TABLE C-2. Effects of rounding on speed parameters for Sony LDP-2000.

Requested speed	Actual speed as multiple of normal
0.0	Error
1-999	0.2
1000	1.0
1000 and up	3

TABLE C-3. Effects of rounding on speed parameters for Pioneer 4200.

Requested speed	Actual speed as multiple of normal
0	Error
1-149	0.1
150-349	0.2
350-649	0.5
650-999	0.8
1000	1
1001-2499	2
2500-3499	3
3500 and up	4

TABLE C-4. Example speed parameter values for boundary player speeds.

Speed parameter value	Resulting player speed
0	Error
1	Slowest available speed
999	Fastest speed that is slower than normal
1001	Slowest speed that is faster than normal
999999	Fastest available speed

TABLE C-5. vdGetState ASCII parameters.

Parameter[1]	Core or extended	Associated calling value	Type as ASCII	Associated return value	Type as ASCII	Default if parameter not used
audio1	Core	None	N/A	1 (on) | 0 (off)	Integer	No action
audio2	Core	None	N/A	1 (on) | 0 (off)	Integer	No action
cdisplay[2]	Extended	None	N/A	1 (on) | 0 (off)	Integer	No action
chapter[2]	Extended	None	N/A	Current chapter number	Integer	No action
defdevice	Core	None	N/A	Default player, 0-15	Integer	No action
device[3]	Core	Logical player, 0-15	Integer	None	N/A	Default player
disctype	Core	None	N/A	1 (CLV) | 0 (CAV)	Integer	No action
door[2]	Extended	None	N/A	1 (open) | 0 (closed)	Integer	No action
frame[4]	Core	None	N/A	Current frame number	Integer	No action
idxdisplay	Core	None	N/A	1 (on) | 0 (off)	Integer	No action
motion	Core	None	N/A	1 (on) | 0 (off)	Integer	No action
remote[2]	Extended	None	N/A	1 (on) | 0 (off)	Integer	No action
speed	Core	None	N/A	Current player speed or 999999 if scanning	Integer	No action
spin	Core	None	N/A	1 (up) | 0 (down)	Integer	No action
tdevices	Core	None	N/A	Total devices installed, 1-16	Integer	No action
video	Core	None	N/A	1 (on) | 0 (off)	Integer	No action

[1] At least one parameter must be specified or an error is returned.

[2] If supported. This is an extended parameter. Using an unimplemented extended parameter causes error 52 (Parameter invalid for this command).

[3] If device is specified, at least one other parameter must also be specified.

[4] Supported for CAV videodisc only. All other parameters apply to both CAV and CLV videodiscs.

TABLE C-6. vdGetState binary parameters.

Parameter[1]	Core or extended	Token number (decimal)	Type	Associated calling value	Associated return value	Default if parameter not used
audio1	Core	2	Integer	Any value	1 (on) | 0 (off)	No action
audio2	Core	3	Integer	Any value	1 (on) | 0 (off)	No action
cdisplay[2]	Extended	6	Integer	Any value	1 (on) | 0 (off)	No action
chapter[2]	Extended	7	Integer	Any value	Current chapter number	No action
defdevice	Core	12	Integer	Any value	Default player, 0-15	No action
device[3]	Core	14	Integer	Logical player, 0-15	None	Default player
disctype	Core	16	Integer	Any value	1 (CLV) | 0 (CAV)	No action
door[2]	Extended	18	Integer	Any value	1 (open) | 0 (closed)	No action
frame[4]	Core	23	Integer	Any value	Current frame number	No action
idxdisplay	Core	29	Integer	Any value	1 (on) | 0 (off)	No action
motion	Core	35	Integer	Any value	1 (on) | 0 (off)	No action
remote[2]	Extended	39	Integer	Any value	1 (on) | 0 (off)	No action
speed	Core	40	Integer	Any value	Current player speed or 999999 if scanning	No action
spin	Core	41	Integer	Any value	1 (up) | 0 (down)	No action
tdevices	Core	45	Integer	Any value	Total devices installed, 1-16	No action
video	Core	51	Integer	Any value	1 (on) | 0 (off)	No action

[1] At least one parameter must be specified or an error is returned.

[2] If supported. This is an extended parameter. Using an unimplemented extended parameter causes error 52 (Parameter invalid for this command).

[3] If device is specified, at least one other parameter must also be specified.

[4] Supported for CAV videodisc only. All other parameters apply to both CAV and CLV videodiscs.

TABLE C-7. vdInit parameters.

ASCII Parameters						
Parameter	Core or extended	Associated calling value	Type as ASCII	Associated return value	Type as ASCII	Default if parameter not used
device[1]	Core	Logical player, 0-15	Integer	None	N/A	Default player
Binary Parameters						
Parameter	Core or extended	Token number (decimal)	Type	Associated calling value	Associated return value	Default if parameter not used
device[1]	Core	14	Integer	Logical player, 0-15	None	Default player

[1] This parameter applies to both CAV and CLV videodisc, as does vdInit without parameters.

TABLE C-8. Parameter values set by vdInit.

Parameter	Value	Command reference
audio1	1 (on)	vdSet
audio2	1 (on)	vdSet
cdisplay[1]	0 (off) or undefined	vdSet
door[1]	0 (closed)	vdSet
frame	First available on disc	vdPlay, vdSet
idxdisplay	0 (off)	vdSet
motion	0 (off)	vdGetState
remote[1]	0 (off)	vdSet
spin	1 (up)	vdSet
tdevices	Total installed, 1-16	none
video	1 (on)	vdSet

[1] If supported. This is an extended parameter. Using an unimplemented extended parameter causes error 52 (Parameter invalid for this command).

TABLE C-9. vdPassThru parameters.

ASCII Parameters						
Parameter[1]	Core or extended	Associated calling value	Type as ASCII	Associated return value	Type as ASCII	Default if parameter not used
device	Core	Logical player, 0-15	Integer	None	N/A	Default player
pmsg[2]	Core	Unquoted string of hexadecimal digits	Text	None	N/A	Causes error
time	Core	Timeout in milliseconds	Integer	None	N/A	Default time
Binary Parameters						
Parameter[1]	Core or extended	Token number (decimal)	Type	Associated calling value	Associated return value	Default if parameter not used
device	Core	14	Integer	Logical player, 0-15	None	Default player
pmsg[2, 3]	Core	37	Pointer	Pointer to player command string	Pointer to player response string	Causes error
time	Core	47	Integer	Timeout in milliseconds	None	Default time

[1] All parameters apply to both CAV and CLV videodisc players.

[2] *Pmsg parameter* must be used with the vdPassThru command or an error is returned.

[3] The memory pointed to by pmsg must be a minimum of 33 bytes.

TABLE C-10. vdPlay parameters.

ASCII Parameters						
Parameter[1]	Core or extended	Associated calling value	Type as ASCII	Associated return value	Type as ASCII	Default if parameter not used
chapter[2]	Extended	Chapter number	Integer	None	N/A	No action
device	Core	Logical player, 0-15	Integer	None	N/A	Default player
direction[3]	Core	1 (fwd) | 0 (back)	Integer	None	N/A	1
from[3]	Core	Starting frame number	Integer	None	N/A	Current frame
speed	Core	Play speed, >0	Integer	None	N/A	1000
to[3]	Core	Ending	Integer	None	N/A	Disc limit
wait	Core	None	N/A	None	N/A	No wait
Binary Parameters						
Parameter[1]	Core or extended	Token number (decimal)	Type	Associated calling value	Associated return value	Default if parameter not used
chapter[2]	Extended	7	Integer	Chapter number	None	No action
device	Core	14	Integer	Logical player, 0-15	None	Default player
direction[3]	Core	15	Integer	1(fwd) | 0(back)	None	1
from[3]	Core	24	Integer	Starting frame number	None	Current frame
speed	Core	40	Integer	Play speed, >0	Actual speed after rounding if required	1000
to[3]	Core	48	Integer	Ending frame number	None	Disc limit
wait	Core	54	N/A	None	None	No wait

[1] The vdPlay command may be issued without any parameters. Paragraph 70.2 describes improper usage of vdPlay parameters.

[2] Using an unimplemented extended parameter causes error 52 (Parameter invalid for this command).

[3] Parameter is supported for CAV videodisc players only. All other parameters apply to both CAV and CLV videodisc players, as does vdPlay issued with no parameters.

TABLE C-11. Effects of the wait parameter on vdPlay.

Additional parameter	Returns control when?
none	After the default player is playing normally
chapter	After the player has played the specified chapter
device only	After the specified player is playing normally
from with any other legal parameters except to	After the player has searched to the specified from frame
speed only	After the player is playing normally at the specified speed
to with any other legal parameters	After the player has reached the specified to frame

TABLE C-12. vdScan parameters.

ASCII Parameters						
Parameter[1]	Core or extended	Associated calling value	Type as ASCII	Associated return value	Type as ASCII	Default if parameter not used
device	Core	Logical player, 0-15	Integer	None	N/A	Default player
direction[2]	Core	1 (fwd) | 0 (back)	Integer	None	N/A	1
wait	Core	None	N/A	None	N/A	No wait
Binary Parameters						
Parameter[1]	Core or extended	Token number (decimal)	Type	Associated calling value	Associated return value	Default if parameter not used
device	Core	14	Integer	Logical player, 0-15	None	Default player
direction[2]	Core	15	Integer	1 (fwd) | 0 (back)	None	1
wait	Core	54	N/A	None	None	No wait.

[1] The vdScan command may be issued without any parameters.

[2] Parameter supported for CAV videodiscs only. All other parameters apply to both CAV and CLV videodiscs, as does the vdScan command without any parameters.

TABLE C-13. vdSearch parameters.

ASCII Parameters						
Parameter[1]	Core or extended	Associated calling value	Type as ASCII	Associated return value	Type as ASCII	Default if parameter not used
chapter	Core	Chapter number	Integer	None	N/A	No action
device	Core	Logical player, 0-15	Integer	None	N/A	Default player
frame[2]	Core	Frame number	Integer	None	N/A	No action
wait	Core	None	N/A	None	N/A	No wait
Binary Parameters						
Parameter[1]	Core or extended	Token number (decimal)	Type	Associated calling value	Associated return value	Default if parameter not used
chapter	Extended	7	Integer	Chapter number	None	No action
device	Core	14	Integer	Logical player, 0-15	None	Default player
frame[2]	Core	3	Integer	Frame number	None	No action
wait	Core	54	N/A	None	None	No wait

[1] The vdSearch command must include either the frame or chapter parameter or an error is returned. If device or wait is specified, at least one other parameter must also be specified.

[2] Parameter is supported for CAV videodiscs only. All other parameters apply to both CAV and CLV videodiscs.

TABLE C-14. vdSet ASCII parameters.

Parameter[1]	Core or extended	Associated calling value	Type as ASCII	Associated return value	Type as ASCII	Default if parameter not used
audio1	Core	1 (on) | 0 (off)	Integer	None	N/A	No action
audio2	Core	1 (on) | 0 (off)	Integer	None	N/A	No action
cdisplay[2]	Extended	1 (on) | 0 (off)	Integer	None	N/A	No action
defdevice	Core	Logical player, 0-15	Integer	None	N/A	No action
device[3]	Core	Logical player, 0-15	Integer	None	N/A	Default player
door[2]	Extended	1 (open) | 0 (closed)	Integer	None	N/A	No action
idxdisplay	Core	1 (on) | 0 (off)	Integer	None	N/A	No action
remote[2]	Extended	1 (on) | 0 (off)	Integer	None	N/A	No action
spin	Core	1 (up) | 0 (down)	Integer	None	N/A	No action
video	Core	1 (on) | 0 (off)	Integer	None	N/A	No action
wait[3]	Core	None	N/A	None	N/A	No wait

[1] At least one parameter is required or an error is returned. All parameters apply to both CAV and CLV videodiscs.

[2] If supported. Using an unimplemented extended parameter causes error 52 (Parameter invalid for this command).

[3] If device or wait is specified, at least one additional parameter must also be specified.

TABLE C-15. vdSet binary parameters.

Parameter[1]	Core or extended	Token number (decimal)	Type	Associated calling value	Associated return value	Default if parameter not used
audio1	Core	2	Integer	1 (on) | 0 (off)	None	No action
audio2	Core	3	Integer	1 (on) | 0 (off)	None	No action
cdisplay[2]	Extended	6	Integer	1 (on) | 0 (off)	None	No action
defdevice	Core	12	Integer	Logical player, 0-15	None	No action
device[3]	Core	14	Integer	Logical player, 0-15	None	Default player
door[2]	Extended	18	Integer	1 (open) | 0 (closed)	None	No action
idxdisplay	Core	29	Integer	1 (on) | 0 (off)	None	No action
remote[2]	Extended	39	Integer	1 (on) | 0 (off)	None	No action
spin	Core	41	Integer	1 (up) | 0 (down)	None	No action
video	Core	51	Integer	1 (on) | 0 (off)	None	No action
wait[3]	Core	54	N/A	None	None	No wait

[1] At least one parameter is required or an error is returned. *All parameters apply to both CAV and CLV videodiscs.*

[2] If supported. Using an unimplemented extended parameter causes error 52 (Parameter invalid for this command).

[3] If device or wait is specified, at least one additional parameter must also be specified.

TABLE C-16. vdStep parameters.

ASCII Parameters						
Parameter[1]	Core or extended	Associated calling value	Type as ASCII	Associated return value	Type as ASCII	Default if parameter not used
device	Core	Logical player, 0-15	Integer	None	N/A	Default player
direction	Core	1 (fwd) | 0 (back)	Integer	None	N/A	1
Binary Parameters						
Parameter[1]	Core or extended	Token number (decimal)	Type	Associated calling value	Associated return value	Default if parameter not used
device	Core	14	Integer	Logical player, 0-15	None	Default player
direction	Core	15	Integer	1 (fwd) | 0 (back)	None	1

[1] The vdStep command, with or without parameters, applies to CAV videodiscs only. vdStep can be issued with no parameters.

TABLE C-17. vdStill parameters.

ASCII Parameters						
Parameter[1]	Core or extended	Associated calling value	Type as ASCII	Associated return value	Type as ASCII	Default if parameter not used
device[2]	Core	Logical player, 0-15	Integer	None	N/A	Default player
Binary Parameters						
Parameter[1]	Core or extended	Token number (decimal)	Type	Associated calling value	Associated return value	Default if parameter not used
device[2]	Core	14	Integer	Logical player, 0-15	None	Default player

[1] The vdStill command applies to both CAV and CLV videodiscs and may be issued with no parameter.

[2] The device parameter applies to both CAV and CLV videodiscs.

APPENDIX C

This Page Intentionally Left Blank

APPENDIX D

XY-INPUT (xy) COMMANDS FOR ICW PORTABILITY

10. SCOPE

10.1 Scope. This appendix describes commands that relate to XY-input devices such as mice, touch screens, and light pens. These commands provide a uniform way to obtain information from these devices and define coordinate spaces. Table D-1 lists the commands covered in this section, their token numbers, and their types.

10.2 Application guidance. This appendix is written to support implementation of the software interface and command requirements prescribed by MIL-STD-1379, Appendix D that relate to overall software operation in ICW applications. Each of the xy commands listed in Table D-1 are addressed in a separate section of this appendix, beginning with Section 40, xyGetInput command.

10.2.1 Terms, abbreviations, and acronyms used in this appendix. Key terms, abbreviations, and acronyms used in this appendix are defined as specified in Section 3 of the basic handbook.

20. APPLICABLE DOCUMENTS.

20.1 Government documents.

20.1.1 Specifications, standards, and handbooks. The following specifications, standards, and handbooks form a part of this appendix to the extent specified herein.

STANDARD

MILITARY

MIL-STD-1379 Military Training Programs

(Unless otherwise specified, copies of military specifications, standards and handbooks are available from the Standardization Documents Order Desk, Building 4D, 700 Robbins Avenue, Philadelphia, PA 19111-5094.)

APPENDIX D

30. GENERAL GUIDANCE

30.1 General information and assumptions. The general information and assumptions in this section were used in the definition of the XY-input commands.

30.1.1 Device mapping. Typically, each physical XY-input device is treated independently and mapped to a unique logical device number. However, VDI implementers may choose to support multiple physical devices as a single logical device by mapping the devices to a single logical device number. If so, VDI Management must correct the raw values returned by the physical devices so that multiple devices return the same value to the application for the same screen position. Return values are based on the application-coordinate space established with the **xySet** command. Mapping of multiple devices to a single logical device allows a user to use, for example, a mouse and a touch screen that both appear to be the same device from the application's viewpoint. In addition:

a. All mapping must be done when VDI Management is installed. Devices cannot be remapped at run-time and mapping is not under application control.

b. Mapping the keyboard or cursor keypad to an XY-input device is optional. How such support is provided is an implementation issue and is not considered by the ICW Portability Practices.

c. Note that the xy service group keeps sets of all parameters that can be returned by **xyGetInput** and **xyGetState** for each logical device. This function is described in the appendix sections for these commands.

30.1.2 Handling the graphics plane and cursor. Well-behaved applications should not turn off the graphics plane when they need selection and coordinate input. The plane must be active for a device such as a mouse to display a cursor for making menu selections and doing similar tasks. Additionally:

a. Applications should limit active XY-input areas to the active graphics plane, even though some XY-input devices such as touch screens allow input beyond the limits of active graphics. Again, this is necessary for devices such as mice that rely on the graphics plane for cursor display.

b. For VDI Management implementations that support the cursor parameter, the application can determine if a device supports a graphics cursor with the **xyGetState** command. If the device does support a cursor, the application should turn it on for XY input.

30.1.3 Coordinate space mapping. The alignment of specific XY-coordinate values versus graphics is a VDI Management implementation issue. However, the minimum and

maximum values for X and Y always map to the edges of the active graphics area. For example:

a. If the minimum value of X is 0 and the maximum is 10, these values map to the left and right edges of the active graphics (these values are typically 0 and 319, or 0 and 639), respectively. If the minimum and maximum values are -100 and +100, they still map to the left and right edges of active graphics.

b. Note that coordinates are not restricted to the largest X value always mapping to the right side of the screen and the largest Y value to the bottom of the screen. For example, **xyset xmin=100,xmax=0** results in the left edge of the screen mapping to 100 and the right edge to 0 (see 70. **xyset** command).

30.1.3.1 Clipping values. Clipping values for X and Y cannot lie outside of the minimum and maximum values. Trying to set clipping values outside of the minimum and maximum values causes an error. In addition:

a. For relative positioning devices such as mice, VDI Management ignores changes in position that take the cursor outside of the clipping area.

b. For absolute positioning devices such as touch screens, VDI Management ignores button presses outside of the clipping area.

c. Application developers should note that the behavior differs between these two device classes and should consider testing applications against both classes.

30.1.3.2 Coordinate space calibration. Calibrating the XY-coordinate space to the active graphics area is a VDI Management implementation issue. Typically, if a device such as a touch screen requires calibration, the device comes with software to support calibration at installation.

30.1.4 Buttons. In the context of the XY-command set, a button is any device that allows signaling the application that a choice has been made. A button press may consist of touching a finger to a touch screen or pressing a physical button on a mouse. In addition:

a. The command set supports devices with multiple buttons. However, applications should assume single-button devices for maximum portability.

b. The command set supports reporting whether or not a button has been pressed only. It does not distinguish touchdown, liftoff, or intensity (Z dimension). Supporting these variations is an application issue and is nonportable.

c. To be considered compliant, an XY-input device must have at least one switch of some sort that can function as a button. For example, a touch screen should be treated as a one-button device. When a finger is in contact with the touch screen, the button is pressed. When the touch screen is not being touched, the button is not pressed.

30.2 Stream-mode and point-mode devices. XY-input devices fall into two broad categories based on how they make positional information available: stream-mode devices and point-mode. Some devices support one mode only, while others support both depending on configuration. VDI Management developers should also consider:

a. In stream mode, devices make position and selection information available on a continuous basis. Software can ask for and receive current information at any time. In point mode, devices make position and selection information available only when a button is being pressed.

b. Stream-mode devices can be forced into point mode by restricting their functionality. However, such reduced functionality would place unwarranted restrictions on application design. Therefore, VDI Management treats all XY-input devices as stream-mode devices.

c. To treat both true stream-mode devices and point-mode devices as stream-mode devices, the reported coordinates will be one of the following:

(1) the current coordinates from a true stream-mode device,

(2) the coordinates at the time the button was last pressed for a point-mode device, or

(3) the minimum X and Y values (typically 0,0) for a point-mode device for which no button has been pressed since the device was initialized.

40. xyGetInput COMMAND

40.1 xyGetInput command summary. **xyGetInput** is a core command having a binary token number of 4100 decimal. **xyGetInput** returns the current position and button status of the XY-input device.

40.2 Command parameters. The **xyGetInput** command has four defined parameters: **buttons, device, xpos,** and **ypos.** All are core parameters that must be supported by compliant VDI Management implementations. Table D-2 lists ASCII and binary parameter information for **xyGetInput.**

40.2.1 Buttons parameter. The **button** parameter returns the state of all buttons as a bit field. Each bit in the bit field can have two states: zero (open) or one (closed). A device can have up to 32 buttons numbered zero (0) through 31, and:

a. The binary interface returns a 4-byte bit field. The least significant bit (bit 0) of the least significant byte (byte 0) corresponds to button zero, the next bit to button one, and so on. For example, if an input device had three buttons with states of closed, open, closed, the binary interface would return 00000101B in the low byte.

b. The ASCII interface returns the same bit field as an integer value. For the three buttons in the example above, the ASCII interfaces would return "5" (4 + 0 + 1).

c. Some touch screens may be able to detect a touch only when the first contact between finger and touch screen is made. If so, **syGetInput buttons** returns one if the screen was touched since the last call to **syGetInput** or **xyInit,** and zero if the screen was not touched.

40.2.2 Device parameter. The **device** parameter directs **xyGetInput** to the specified logical XY-input device regardless of the default device number set by **xyset defdevice.** Specifying **device** with no other parameter returns error 49 (Insufficient parameters). Specifying a nonexistent or uninstalled device returns error 160 (Invalid device number). Specifying an uninitialized device causes error 81 (Device not initialized).

40.2.3 Xpos and ypos parameters. The **xpos** and **ypos** parameters return the current XY coordinates of the input device according to the scale set by **xyset.** When using these parameters:

a. If the coordinate system is changed by any combination of **xySet xmin, ymin, xmax, ymax,** VDI Management automatically converts **xpos** and **ypos** values so that the cursor stays at the same position on the screen. **xyGetInput xpos ypos** returns the new parameter values.

b. If the clipping area is changed by any combination of **xySet xminclip, yminclip, xmaxclip, ymaxclip** and the cursor lies outside the new clipping region, VDI Management automatically converts **xpos** and **ypos** values to move the cursor to the edge of the clipping area nearest to the previous cursor position. **xyGetInput xpos ypos** returns the new parameter values.

40.3 Implementation notes. When using **xyGetInput**, note the following:

a. **xyGetState tbuttons** returns the number of buttons available on a device.

b. Trying to queue **xyGetInput** causes error 177 (Command cannot be queued) at the time of the attempt.

c. If multiple physical XY-input devices are mapped to the same logical XY-input device and two or more of these devices generate absolute coordinates, **xyGetinput xpos ypos** returns the coordinates for the most recently active device — the last device that was moved or clicked.

d. If multiple physical XY-input devices are mapped to the same logical device, **xyGetInput buttons** should do a logical "or" of the button states for all physical devices. Note that **xyGetInput buttons** cannot distinguish the situation where the same button numbers on two or more devices are simultaneously closed.

40.4 Return values.

40.4.1 ASCII returns. On success, the return is a comma-separated list of values for requested parameters as described in 40.2, above. On failure, the return is "ERROR n...".

40.4.2 Binary returns. On success, the return is AX = 0. Values associated with requested parameters are 32-bit values of the types given in the binary parameter portion of Table D-2. On failure, the return is AX = error number. Any return values in the parameter block addressed by ES:DI are undefined and should be ignored.

40.5 Related commands. The **xyGetState** and **xySet** command requirements should also be reviewed in relation to implementing the **xyGetInput** command.

40.6 Examples. The following are ASCII and binary examples of the **xyGetInput** command.

40.6.1 ASCII.

Get current X position	**xyGetInput xpos** (returns) "43"	; the current X ; coordinate is 43

Get current XY positions and state of buttons	xyGetInput xpos,ypos,buttons (returns)"43,110,1"	; XY coordinates are 43,110 ; and button 1 is closed
Get XY positions for input device 2 are	xyGetInput device=2,xpos,ypos (returns)"128,145"	; device 2 XY coordinates ; 128,145

40.6.2 Binary.

Get current XY and buttons values	AX	4100	; xyGetInput decimal ID
	BX	3	; number of parameters
	ES:DI[0]	61	; xpos decimal ID
	ES:DI[4]	Any value	; place holder for value on ; return
	ES:DI[8]	67	; ypos decimal ID
	ES:DI[C]	Any value	; place holder for value on ; return
	ES:DI[10]	5	; buttons decimal ID
	ES:DI[14]	Any value	; place holder for value on ; return
After return	AX	0	; returns 0 if successful, and ; non-zero if not
	ES:DI[4]	X position	; X position value
	ES:DI[C]	Y position	; Y position value
	ES:DI[14]	Bit field	; button status bit field

50. xyGetState COMMAND

50.1 xyGetState command summary. **xyGetState** is a core command having a binary token number of 4102 decimal. **xyGetState** returns information about the current values of the coordinate space, and available devices and capabilities. VDI Management maintains a copy of device-specific parameters including coordinates for each logical - device.

50.2 Command parameters. The xyGetState command has thirteen defined parameters: **cursor, defdevice, device, tbuttons, tdevices, xmax, xmaxclip, xmin, xminclip, ymax, ymaxclip, ymin,** and **yminclip**. All are core parameters that must be supported by compliant VDI Management implementations, except **cursor** which is an extended parameter. Implementation support of extended parameters is optional (see 4.3.2.6). However, VDI Management developers should consider implementing the **cursor** parameter for maximum portability. Tables D-3 and D-4 list ASCII and binary parameter information, respectively, for the xyGetState command.

50.2.1 Cursor parameter. The **cursor** parameter returns one (1) if the graphics cursor is visible. **Cursor** returns zero (0) if the input device supports a cursor that is not visible or the device does not support a cursor, in which case, the cursor must always be off. **Cursor** is an extended parameter. Using an unimplemented extended parameter causes error 52 (Parameter invalid for this command).

50.2.2 Defdevice parameter. The **defdevice** parameter returns the logical number of the default XY-input device set by **xySet defdevice**. VDI management directs all XY commands to this device unless a command includes a **device** parameter (see 50.2.3) directing it to a different input device.

50.2.3 Device parameter. The **device** parameter directs xyGetState to the specified logical device number regardless of the default device number set by **xySet defdevice**. Because **device** affects the command with which it is associated only, the parameter does not affect the return value for **defdevice** when the two parameters are used together. In addition:

a. Specifying **device** with no other parameter returns error 49 (Insufficient parameters).

b. Specifying a nonexistent or uninstalled device returns error 160 (Invalid device number); specifying an uninitialized device causes error 81 (Device not initialized).

50.2.4 Tbuttons parameter. The **tbuttons** parameter returns the total number of buttons available for the default or specified XY-input device.

50.2.5 Tdevices parameter. The **tdevices** parameter returns the total number of logical XY-input devices for which VDI Management was configured at installation. If only one device is installed, it is numbered zero and **tdevices** returns one. This parameter alerts the application to systems that have more than one available input device -- both a mouse and touch screen, for example.

50.2.6 Xmin, ymin, xmax, and ymax parameters. The **xmin**, **ymin**, **xmax**, and **ymax** parameters return the current scaling of the XY-coordinate system. The **xmin** and **ymin** values are the coordinates corresponding to the physical location of the upper left corner of the active graphics area. The **xmax** and **ymax** values correspond to the lower right corner of the active graphics area. VDI Management scales absolute positioning information to the space defined by these parameters.

50.2.7 Xminclip, yminclip, xmaxclip, and yminclip parameters. The **xminclip**, **yminclip**, **xmaxclip**, and **ymaxclip** parameters return the area within the XY-coordinate space within which changes of position are reported.

50.2.8 Parameters resulting in errors. If a parameter causes an error, **xyGetState** returns immediately with the error message. The command does not return partial responses for other parameters that did not cause errors.

50.3 Implementation notes. When using **xyGetState**, note that trying to queue **xyGetState** causes error 177 (Command cannot be queued) at the time of the attempt.

50.4 Return values.

50.4.1 ASCII returns. On success, the return is a comma-separated list of values for the requested parameters as described in 50.2. On failure, the return is "ERROR n...".

50.4.2 Binary returns. On success, the return is AX = 0. Values associated with requested parameters are 32-bit values of the types given in the binary parameter Table D-4. On failure, the return is AX = error number. Any return values in the parameter block addressed by ES:DI are undefined and should be ignored.

50.5 Related commands. The **syGetState**, **vdGetState**, **vmGetState**, **xyGetInput**, **xyInit**, and **xySet** command requirements should be reviewed in relation to implementing the **xyGetState** command.

50.6 Examples. The following are ASCII and binary examples of the **xyGetState** command.

50.6.1 ASCII.

Get total devices, and default device	xyGetState tdevices,defdevice (returns) "2,1"	; two devices - number 1 is ; selected
Get current XY-coordinate space	xyGetState xmin,ymin, xmax,ymax (returns) "0,0,639,199"	; x values will be between ; 0 and 639, and y values ; between 0 and 199
Get available buttons for device 2	xyGetState tbuttons,device=2 (returns) "3"	; three buttons available

50.6.2 Binary.

Determine if cursor is on	AX	4102	; xyGetState decimal ID
	BX	1	; number of parameters
	ES:DI[0]	11	; cursor decimal ID
	ES:DI[4]	Any value	; place holder for value on ; return
After return	AX	0	; returns 0 if successful, and ; non-zero if not
	ES:DI [4]	1	; graphics cursor is on ;(0 = off)

60. xyInit COMMAND

60.1 xyInit command summary. xyInit is a core command having a binary token number of 4103 decimal. xyInit initializes XY-input hardware and the xy service group, placing both in a known state. xyInit must be issued for each attached XY-input device that will be used by the application.

60.2 Command parameters. The xyInit command has one defined parameter: device. Device is a core parameter that must be supported by compliant VDI Management implementations. Table D-5 lists ASCII and binary parameter information for xyInit.

60.2.1 Device parameter. The device parameter specifies the logical number of the XY-input device to be initialized. If device is omitted, xyInit initializes logical device zero, or re-initializes the default device set by xySet defdevice. In addition:

a. A device must have been previously initialized with xyInit for xySet defdevice to set it to the default. If xySet has not been used and the device parameter is omitted, the default device is defined to be logical device zero. On subsequent calls that include a device parameter, xyInit does not change the default device if a device other than the default is specified.

b. If, on the first call to xyInit, a device other than zero is specified, xySet must be used if that device is the desired default. For example, assume that the first call to xyInit is xyInit device=1. This command does not set the default device to one; an application must issue xySet defdevice=1 to do this.

c. To change the default to a device that has not yet been initialized, use xyInit device=n, where "n" is the desired logical device number, followed by xySet defdevice=n. After a device has been initialized, xySet defdevice alone may be used to change the default.

d. Specifying a nonexistent or uninstalled device causes error 160 (Invalid device number).

60.2.2 Conditions set by xyInit. xyInit sets the parameters listed in Table D-6 to the values specified in the table.

60.2.3 Effects of xyInit on the cursor parameter. The extended parameter cursor, if supported, turns the graphics cursor on and off. xyInit should turn the cursor on if VDI Management supports the parameter AND the XY-input device supports a cursor. This is functionally equivalent to sySet cursor=1. In addition:

a. Turning the cursor on by default will let applications developed on a system with an input device that does not support a cursor run without modification on a

system with an input device, such as a mouse, that requires a cursor. However, setting **cursor** = 1 cannot be a compliance requirement because a VDI implementation could support the parameter, but the input device for a particular system running that implementation might not support a cursor. In this case, **syInit** should set **cursor** = 0.

b. If an application needs to know about the **cursor** parameter, it should issue **syGetState cursor** after **xyInit**. VDI Management should return error 52 (Parameter invalid for this command) if **cursor** is not supported, zero if cursor is supported but the input device does not support a cursor, or one if cursor is supported and the input device does support a cursor.

60.3 Implementation notes. When using **xyInit**, note the following:

a. **xyGetState tdevices** returns the total number of logical XY-input devices for which VDI Management was installed. This command can be used after the first **xyInit** to determine the number of additional devices to initialize.

b. Trying to queue **xyInit** causes error 177 (Command cannot be queued) at the time of the attempt.

60.4 Return values.

60.4.1 ASCII returns. On success, the return is "OK". On failure, the return is "ERROR n...".

60.4.2 Binary returns. On success, the return is AX = 0. On failure, the return is AX = error number.

60.5 Related commands. The syInit, vdInit, vmInit, xyGetState, and xySet command requirements should be reviewed in relation to implementing the **xyInit** command.

60.6 Examples. The following are ASCII and binary examples of the **xyInit** command.

60.6.1 ASCII.

Initialize device 0 and xy service group	**xyInit** (returns) "OK"	; first time command is issued
Initialize device 1	**xyInit device** = 1 (returns) "OK"	; device 1 successfully initialized

60.6.2 Binary.

Initialize device 0	AX	4103	; xyInit decimal ID
and xy service group	BX	0	; number of parameters
After return	AX	0	; returns 0 if successful, and ; non-zero if not

70. xySet COMMAND

70.1 xySet command Summary. **xySet** is a core command having a binary token number of 4109 decimal. **xySet** defines the XY-coordinate space, sets the default input device, turns the cursor on and off, and sets the current XY coordinates. Regardless of the graphics mode, each parameter defined by **xySet** stays in effect for either the current or specified device until reset with **xySet** or **xyInit**.

70.2 Command parameters. The **xySet** command has thirteen defined parameters: **cursor**, **defdevice**, **device**, **xmax**, **xmaxclip**, **xmin**, **xminclip**, **xpos**, **ymax**, **ymaxclip**, **ymin**, **yminclip**, and **ypos**. All are core parameters that must be supported by compliant VDI Management implementations, except **cursor** which is an extended parameter. Implementation support for extended parameters is optional (see 4.3.2.6). However, see 50.2 above. Tables D-7 and D-8 list ASCII and binary parameter information, respectively, for **xySet**.

70.2.1 Cursor parameter. The **cursor** parameter enables and disables a graphics cursor, if one is available. For example, if the device is a mouse, **xySet cursor = 1** enables the cursor and makes it visible. Updating the position of such a cursor is a background function, and:

a. If the input device does not support a cursor, turning the cursor on returns error 87 (Action not supported by device).

b. **Cursor** is an extended parameter. Using an unimplemented extended parameter causes error 52 (Parameter invalid for this command).

70.2.2 Defdevice parameter. The **defdevice** parameter specifies the default input device to be used when more than one input device is available. In addition:

a. Specifying a nonexistent or uninstalled device returns error 160 (Invalid device number); specifying an uninitialized device causes error 81 (Device not initialized).

b. **xyGetState tdevices** returns the number of logical input devices that can be selected by **xySet defdevice**, assuming all devices for which VDI Management was installed are available.

70.2.3 Device parameter. The **device** parameter directs **xySet** to the specified logical device number regardless of the default input device set by **xySet defdevice** (see 70.2.2). Additionally:

a. Specifying **device** with no other parameter returns error 49 (Insufficient parameters).

b. Specifying a nonexistent or uninstalled device returns error 160 (Invalid device number); specifying an uninitialized device causes error 81 (Device not initialized).

70.2.4 Xmin, ymin, xmax, and ymax parameters. The xmin, ymin, xmax, and ymax parameters set the scaling of the XY-coordinate space to the physical screen. The values correspond to the upper left and lower right corners of the active graphics display area. Legal values range from -32768 through +32767. Regardless of the values, xmin and xmax always map to the left and right edges of the screen, respectively; ymin and ymax always map to the top and bottom edges. Also consider:

a. Specifying a minimum or maximum coordinate value that would force the current corresponding clipping value (see 70.2.5) to lie outside of the coordinate system causes error 51 (Parameter value invalid or out of range). In this situation, clipping values do not automatically adjust to minimum and maximum coordinate values.

b. Xmin values greater than xmax values, and ymin values greater than ymax values are allowed. For example, xySet xmin=100,xmax=0 results in the left edge of the screen mapping to 100 and the right edge to 0.

70.2.5 Xminclip, yminclip, xmaxclip, and ymaxclip parameters. The xminclip, yminclip, xmaxclip, and ymaxclip parameters define a constrained area within the coordinate space for reporting movement. Coordinates values are returned only within the defined clipping area. The clipping area is initially defined to be the same as xmin, ymin, xmax, and ymax. Specifying a clipping value outside the scaling of the XY-coordinate system (see 70.2.4) returns error 51 (Parameter value invalid or out of range).

70.2.6 Xpos and ypos parameters. The xpos and ypos parameters set the XY coordinates to a specific location. These parameters are especially useful for initially positioning the XY-input device. An xpos or ypos value outside of the clipping values sets the coordinate to the limit of the respective clipping range. In addition:

a. If the coordinate system is changed by any combination of xySet xmin, ymin, xmax, ymax, VDI Management automatically converts the xpos and ypos values so that the cursor stays at the same position on the screen. xyGetInput xpos,ypos returns the new parameter values.

b. If the clipping area is changed by any combination of xySet xminclip, yminclip, xmaxclip, ymaxclip and the cursor lies outside of the new clipping area, VDI Management automatically converts the xpos and ypos values to move the cursor to the edge of the clipping area nearest the previous cursor position. xyGetInput xpos,ypos returns the new parameter values.

70.3 Return values.

70.3.1 ASCII returns. On success, the return is "OK". On failure, the return is "ERROR n...".

70.3.2 Binary returns. On success, the return is AX = 0. On failure, the return is AX = error number.

70.4 Related commands. The xyGetInput, xyGetState, and xyInit command requirements should also be reviewed in relation to implementing the **xySet** command.

70.5 Examples. The following are ASCII and binary examples of the **xySet** command.

70.5.1 ASCII.

Set XY position	xySet xpos=40,ypos=50 (returns) "OK"	; XY position is 40,50
Turn device 2 cursor on	xySet device=2,cursor=1 (returns) "OK"	
Match coordinate space to VGA max	xySet xmin=0,ymin=0,xmax=639,ymax=479 (returns) "OK"	
Set maximum clipping values	xySet xmaxclip=200,ymaxclip=200 (returns) "OK"	

70.5.2 Binary examples.

Turn on XY cursor for the default device	AX	4109	; xySet decimal ID
	BX	1	; number of parameters
	ES:DI[0]	11	; cursor decimal ID
	ES:DI[4]	1	; turn cursor on
After return	AX	0	; Returns 0 if successful, and ; non-zero if not

APPENDIX D

TABLE D-1. XY-Input (xy) commands summary.

ASCII command name[1]	Binary interface token number (decimal)	Type[2]
xyGetInput	4100	Core
xyGetState	4102	Core
xyInit	4103	Core
xySet	4109	Core

[1] Upper or lower case for command names is not significant.

[2] Compliant VDI Management implementations must support Core commands.

TABLE D-2. xyGetInput parameters.

ASCII Parameters						
Parameter[1]	Core or extended	Associated calling value	Type as ASCII	Associated return value	Type as ASCII	Default if parameter not used
buttons	Core	None	N/A	Integer sum of bit field, 1 (closed) | 0 (open)	Integer	No action
device[2]	Core	Logical input device, 0-15	Integer	None	N/A	default device
xpos	Core	None	N/A	Current X value	Integer	No action
ypos	Core	None	N/A	Current Y value	Integer	No action
Binary Parameters						
Parameter[1]	Core or extended	Token number (decimal)	Type	Associated calling value	Associated return value	Default if parameter not used
buttons	Core	5	Integer	Any value	Bit field, 1 (closed) | 0 (open)	No action
device[2]	Core	14	Integer	Logical input device, 0-15	None	Default device
xpos	Core	61	Integer	Any value	Current X value	No action
ypos	Core	67	Integer	Any value	Current Y value	No action

[1] At least one parameter is required or an error is returned.

[2] If device is specified, at least one other parameter must also be specified.

TABLE D-3. xyGetState ASCII parameters.

Parameter[1]	Core or extended	Associated calling value	Type as ASCII	Associated return value	Type as ASCII	Default if parameter not used
cursor[2]	Extended	None	N/A	1 (on) | 0 (off)	Integer	No action
defdevice	Core	None	N/A	Default input device, 0-15	Integer	No action
device[3]	Core	Logical input device, 0-15	Integer	None	N/A	Default device
tbuttons	Core	None	N/A	Total available for device	Integer	No action
tdevices	Core	None	N/A	Total installed, 1-16	Integer	No action
xmax	Core	None	N/A	Maximum possible X value	Integer	No action
xmaxclip	Core	None	N/A	Current maximum X value	Integer	No action
xmin	Core	None	N/A	Minimum possible X value	Integer	No action
xminclip	Core	None	N/A	Current minimum X value	Integer	No action
ymax	Core	None	N/A	Maximum possible Y value	Integer	No action
ymaxclip	Core	None	N/A	Current maximum Y value	Integer	No action
ymin	Core	None	N/A	Minimum possible Y value	Integer	No action
yminclip	Core	None	N/A	Current minimum Y value	Integer	No action

[1] At least one parameter is required or an error is returned.

[2] Cursor is an extended parameter. Using an unimplemented extended parameter causes error 52 (Parameter invalid for this command).

[3] If device is specified, at least one other parameter must also be specified.

TABLE D-4. xyGetState binary parameters.

Parameter[1]	Core or extended	Token number (decimal)	Type	Associated calling value	Associated return value	Default if parameter not used
cursor[2]	Extended	11	Integer	Any value	1 (on) | 0 (off)	No action
defdevice	Core	12	Integer	Any value	Default input device, 0-15	No action
device[3]	Core	14	Integer	Logical input device, 0-15	None	Default device
tbuttons	Core	44	Integer	Any value	Total available for device	No action
tdevices	Core	45	Integer	Any value	Total installed for, 1-16	No action
xmax	Core	56	Integer	Any value	Maximum possible X value	No action
xmaxclip	Core	57	Integer	Any value	Current maximum X value	No action
xmin	Core	58	Integer	Any value	Minimum possible X value	No action
xminclip	Core	59	Integer	Any value	Current minimum X value	No action
ymax	Core	62	Integer	Any value	Maximum possible Y value	No action
ymaxclip	Core	63	Integer	Any value	Current maximum Y value	No action
ymin	Core	64	Integer	Any value	Minimum possible Y value	No action
yminclip	Core	65	Integer	Any value	Current minimum Y value	No action

[1] At least one parameter is required or an error is returned.

[2] Cursor is an extended parameter. Using an unimplemented extended parameter causes error 52 (Parameter invalid for this command).

[3] If device is specified, at least one other parameter must also be specified.

TABLE D-5. xyInit parameters.

ASCII Parameters						
Parameter[1]	Core or extended	Associated calling value	Type as ASCII	Associated return value	Type as ASCII	Default if parameter not used
device	Core	Logical input device, 0-15	Integer	None	N/A	Default device
Binary Parameters						
Parameter[1]	Core or extended	Token number (decimal)	Type	Associated calling value	Associated return value	Default if parameter not used
device	Core	14	Integer	Logical input device, 0-15	None	Default device

[1] The xyInit command may be issued without any parameters.

TABLE D-6. Parameter values set by xyInit.

Parameter	Value	Command reference
tbuttons	Total available for device being initialized	none
tdevices	Total installed, 1-16	none
xmax	639	xySet
xmaxclip	639	xySet
xmin	0	xySet
xminclip	0	xySet
xpos	0	xySet
ymax	199	xySet
ymaxclip	199	xySet
ymin	0	xySet
yminclip	0	xySet
ypos	0	xySet

TABLE D-7. xySet ASCII parameters.

Parameter[1]	Core or extended	Associated calling value	Type as ASCII	Associated return value	Type as ASCII	Default if parameter not used
cursor[2]	Extended	1 (on) | 0 (off)	Integer	None	N/A	No action
defdevice	Core	Logical input device, 0-15	Integer	None	N/A	No action
device[3]	Core	Logical input device, 0-15	Integer	None	N/A	Default device
xmax	Core	Maximum possible X value	Integer	None	N/A	No action
xmaxclip	Core	Current maximum X value	Integer	None	N/A	No action
xmin	Core	Minimum possible X value	Integer	None	N/A	No action
xminclip	Core	Current minimum X value	Integer	None	N/A	No action
xpos	Core	X position	Integer	None	N/A	No action
ymax	Core	Maximum possible Y value	Integer	None	N/A	No action
ymaxclip	Core	Current maximum Y value	Integer	None	N/A	No action
ymin	Core	Minimum possible Y value	Integer	None	N/A	No action
yminclip	Core	Current minimum Y value	Integer	None	N/A	No action
ypos	Core	Y position	Integer	None	N/A	No action

[1] At least one parameter is required or an error is returned.

[2] Cursor is an extended parameter. Using an unimplemented extended parameter causes error 52 (Parameter invalid for this command).

[3] If device is specified, at least one other parameter must also be specified.

TABLE D-8. xySet Binary parameters.

Parameter[1]	Core or extended	Token number (decimal)	Type	Associated calling value	Associated return value	Default if parameter not used
cursor[2]	Extended	11	Integer	1 (on) | 0 (off)	None	No action
defdevice	Core	12	Integer	Logical input device, 0-15	None	No action
device[3]	Core	14	Integer	Logical input device, 0-15	None	Default device
xmax	Core	56	Integer	Maximum possible X value	None	No action
xmaxclip	Core	57	Integer	Current maximum X value	None	No action
xmin	Core	58	Integer	Minimum possible X value	None	No action
xminclip	Core	59	Integer	Current minimum X value	None	No action
xpos	Core	61	Integer	X position	None	No action
ymax	Core	62	Integer	Maximum possible Y value	None	No action
ymaxclip	Core	63	Integer	Current maximum Y value	None	No action
ymin	Core	64	Integer	Minimum possible Y value	None	No action
yminclip	Core	65	Integer	Current minimum Y value	None	No action
ypos	Core	67	Integer	Y position	None	No action

[1] At least one parameter is required or an error is returned.

[2] Cursor is an extended parameter. Using an unimplemented extended parameter causes error 52 (Parameter invalid for this command).

[3] If device is specified, at least one other parameter must also be specified.

This Page Intentionally Left Blank

APPENDIX E

DIGITAL AUDIO (da) COMMANDS FOR ICW PORTABILITY

10. SCOPE

10.1 Scope. This appendix describes commands that control digital audio devices. These commands will be used to initialize, obtain information about, and control the behavior of digital audio devices installed in or connected to the ICW training system.

10.2 Application guidance. The commands, command parameters, and ASCII and binary interface requirements for the digital audio service group are not yet defined. The service group is identified for future compatibility. This appendix is reserved to support and implement the digital audio service group requirements once they are defined in MIL-STD-1379, Appendix D.

(THIS APPENDIX IS RESERVED)

APPENDIX E

This Page Intentionally Left Blank.

APPENDIX F

AUDIO MANAGEMENT (am) COMMANDS FOR ICW PORTABILITY

10. SCOPE

10.1 Scope. This appendix describes commands that relate to the system audio management functions. These commands will be used to initialize, obtain information about, and control the signal routing and operational modes of audio systems installed in or connected to the ICW training system.

10.2 Application guidance. The commands, command parameters, and ASCII and binary interface requirements for the audio management (am) service group are not yet defined. The service group is identified for future compatibility. This appendix is reserved to support and implement audio management service group requirements once they are defined in MIL-STD-1379, Appendix D.

(THIS APPENDIX IS RESERVED)

APPENDIX F

This Page Intentionally Left Blank.

APPENDIX G

DEFAULT POSITIONS OF ICW GRAPHICS

10. SCOPE

10.1 Scope. This appendix describes how to determine the size and position of graphics relative to background video. To ensure the compatibility of hardware, VDI Management software, and ICW applications, the active graphics screen for a given application should always have the same position relative to the active video, and be of the same size. However, the proper position of graphics can vary with the video standard (NTSC versus PAL), the graphics mode, and the adapter type (VGA versus CGA and EGA). This appendix describes a method for determining horizontal and vertical graphics position for both NTSC and PAL video standards.

10.1.1 Terms, abbreviations, and acronyms used in this appendix. Key terms, abbreviations, and acronyms used in this appendix are defined as specified in Section 3 of the basic handbook.

20. APPLICABLE DOCUMENTS.

20.1 Non-Government publications. The following documents form a part of this document to the extent specified herein.

ELECTRONIC INDUSTRIES ASSOCIATION (EIA)

EIA RS-170	Monochrome Television Studio Facilities, Electrical Performance Standards

(Application for copies should be addressed to the Electronic Industries Association (EIA), 2001 Eye Street, NW, Washington, DC 20006.)

INTERNATIONAL RADIO CONSULTATIVE COMMITTEE

Volume XI-PART 1-82 Broadcasting Service (Television)

(CCIR 470-1, Television Systems, is a part of the document listed above). Application for copies of the document should be addressed to the International Telecommunication Union, Place DES NATIONS, CH-1216, Geneva, 20 Switzerland.)

(Non-Government standards and other publications are normally available from the organizations that prepare or distribute the documents. These documents also may be available in or through libraries or other informational services.)

APPENDIX G

30. GENERAL GUIDANCE

30.1 Introduction. Although exact registration and graphics screen sizes — within one or two pixels or lines — may require user calibration using a position reference frame, proper registration can be calculated with reasonable accuracy. The following paragraphs explain how to determine horizontal and vertical graphics positions for both NTSC and PAL video.

30.1.1 General application. The recommended positions in this appendix are guidelines for a nominal video image because no absolute specification exists for the size and position of the background video and because the position can vary in post-production generation of videodiscs. If exact positioning is critical, the videodisc for the application should include a reference frame for user calibration at run time. This requires VDI Management implementations to support dynamic repositioning of graphics.

30.1.2 Graphics registration. Proper registration requires accurately setting the graphics width as well as the origin. Correctly setting the origin but using the wrong width results in improper registration on the right side of the video. Generating the graphics clock so that exactly 912 clock cycles equal 1 horizontal period assures proper width. If an overlay method does not guarantee this relationship, implementers must provide a way to adjust the graphics width and the reference frame must provide both left and right registration information.

30.1.3 Graphics display. Figure G-1 shows a simplified display screen including video, graphics, sync signals, and blanking intervals. For simplicity, the figure shows separate horizontal and vertical sync signals. Although these signals may be thought of as separate for determining graphics positions, they may be combined into a composite signal in actual monitors.

30.1.4 Graphics values. The values used in the figures and calculations for horizontal and vertical positioning are based on accepted definitions for NTSC and PAL video. The corresponding standards are EIA RS-170 and CCIR 470-1, respectively.

30.2 Special considerations for VGA graphics. VGA graphics require special consideration for two reasons. The first deals with the differences in background video signals and vertical timing. The second deals with differences in active graphics sizes for the same video modes when considering VGA graphics versus CGA and EGA graphics.

30.2.1 Differences in signals and timing. CGA and EGA graphics overlay systems typically use standard, 15 KHz background video. Accepted video standards for these systems dictate blanking interval widths, and the starts of horizontal and vertical sync.

a. CGA and EGA graphics systems can use the starts of horizontal and vertical sync as absolute references for graphics positioning. These systems can also rely on

constant horizontal signal width and vertical timing for a given video standard, either NTSC or PAL.

b. VGA systems typically use scan-altered, non-15 KHz modes. The background video for VGA may not include horizontal or vertical sync, and blanking interval widths may vary. Therefore, graphics positioning must use the nominal start of active video as a reference instead of the horizontal and vertical sync pulse. Graphics positioning must be relative to the active video rather than the total video including blanking.

30.2.2 Differences in the size of active graphics. By increasing pixel width, VGA graphics cover the entire width of active video, leaving borders only at the top and bottom of the active graphics. However, a CGA or EGA adapter used to display graphics in the same mode leaves a border around all edges of the active graphics. For example, VGA mode 6 (640 X 200) graphics cover the entire width of the active video, while a CGA or EGA mode 6 (640 X 200) graphics will leave a visible video border around all edges of the active graphics. Therefore, compliant VDI Management implementations for VGA overlay systems must support both graphics that map to the left and right edges of active video and, for those overlay modes that are CGA/EGA-compatible, emulations of true CGA and EGA systems that leave a border around all edges of active graphics.

APPENDIX G

40. HORIZONTAL POSITIONS

40.1 Horizontal positions. This paragraph describes how to determine the start and end of active graphics relative to a horizontal video signal.

a. The horizontal position of graphics relative to video can be expressed as a proportion of the horizontal video signal width, which is abbreviated H. Using proportions simplifies determining positions for scan-altered systems.

b. Because horizontal sync is the timing reference from which all 15 KHz video horizontal components are measured, true CGA and EGA graphics adapters use the start of horizontal sync as a reference. Then, graphics positions can be expressed as proportions of total H. Because VGA video and graphics may not include a horizontal sync signal and the width of the blanking intervals may vary, VGA graphics modes that emulate CGA and EGA modes are measured from the nominal start of active video, and are expressed as a proportion of active H.

40.1.1 General assumptions. The following general assumptions are used in determining the horizontal positions of active graphics relative to NTSC and PAL video.

a. The optimal position for active graphics is centered horizontally in the active video. The basis for this assumption is nominal common practice. However, variations in graphics positioning due to monitor centering adjustments cannot be accounted for in this appendix.

b. One horizontal line of graphics is 912 pixels in length. The active graphics consist of a 640-pixel window. The basis for this is that CGA standards define a 640-pixel active graphics window in a 912-pixel horizontal line. The graphics are about 85 percent of the total displayable window to allow for monitor over-scan. This has become a de facto standard that is also used by EGA graphics. CGA- and EGA-based overlay systems generally follow this standard.

c. Because color graphics standards are based on a line length of 912 pixels, 1/912 H or approximately 0.0011 H is the finest positioning resolution available. This is based upon the original IBM CGA implementation criteria.

d. Graphics with either a 320-pixel active area or a 640-pixel active area cover exactly the same horizontal area. Therefore, starting and ending positions based on calculations using 640 pixels are also valid for 320-pixel graphics. The pixel width for 320-pixel graphics modes is exactly twice the width for 640-pixel graphics, and the number of pixels for 320-pixel graphics is exactly one half the number of pixels for 640-pixel modes.

40.2 NTSC Video. Figure G-2 shows the timing for one line of 15 KHz NTSC video including the position of CGA and EGA 640- or 320-pixel graphics. The timing values in the figure are either taken directly or derived from the accepted standard for NTSC video. The calculations in this section are based on the values shown in the figure.

40.2.1 Position of true CGA and EGA graphics. To determine the correct position of true CGA and EGA graphics over video as a fraction of a horizontal, 15.734 KHz, NTSC video signal using the timing shown in Figure G-2, use the following equations and assumptions for 640-pixel graphics. Note that the resulting positions also apply for 320-pixel graphics.

a. The general equation for the starting position of true CGA and EGA graphics centered in active video as a proportion of total H using the start of sync as a reference can be expressed as:

$$GH_{start} = \frac{AV_{start} + \left(\frac{(P_{active} - P_{displayed}) \times P_{width}}{2}\right)}{H_{total}} \qquad \text{Equation 1}$$

b. Similarly, the general equation for determining the ending position of true CGA and EGA graphics as a proportion of total H can be expressed as:

$$GH_{end} = \frac{AV_{start} + \left(\frac{(P_{active} - P_{displayed}) \times P_{width}}{2}\right) + (P_{displayed} \times P_{width})}{H_{Total}} \qquad \text{Equation 2}$$

Where: GH stands for "graphics horizontal" and for NTSC video and 640-pixel resolutions:

(1) The nominal start of active video from Figure G-2 is:

$$AV_{start} = 9.4\ \mu s$$

(2) From Figure G-2 and the de facto standard (see 40.3.1), the total number of pixels corresponding to the width of active video is:

$$P_{active} = \frac{52.656\mu s}{63.556\mu s} \times 912\ pixels = 756\ pixels$$

(3) From the de facto standard (see 40.3.1), the total number of displayed pixels is:

$$P_{displayed} = 640 \text{ pixels}$$

(4) From Figure G-2, the total width of the video signal is:

$$H_{total} = 63.556\ \mu s$$

(5) The width of one pixel is:

$$P_{width} = \frac{63.556 \mu s}{912} = 0.06969 \mu s$$

c. Solving equation 1 for the starting position of 640-pixel graphics yields:

$$GH_{start} = \frac{9.4 + \left(\frac{(756 - 640) \times 0.06969}{2}\right)}{63.556} = 0.2115\ H_{total}$$

d. Solving equation 2 for the ending position of 640-pixel graphics yields:

$$GH_{end} = \frac{9.4 + \left(\frac{(756 - 640) \times 0.06969}{2}\right) + (640 \times 0.06969)}{63.556} = 0.9133\ H_{total}$$

e. For NTSC video, these values equate to left and right border widths of 4.042 microseconds and a start of active graphics at approximately 13.4 microseconds after the start of horizontal sync. The latter value can be reliably verified with an accurate oscilloscope.

40.2.2 Position of VGA graphics emulating CGA and EGA modes. True VGA graphics map to the edges of active video and require no calculations. However, the starting and ending positions of VGA graphics emulating CGA and EGA graphics must be calculated. To determine the correct position of such graphics over video as a fraction of a horizontal, 15.734 KHz, NTSC video signal given the timing shown in Figure G-2, use the following equations and assumptions for 640-pixel graphics. The resulting positions are also true for 320-pixel graphics.

APPENDIX G

a. The general equation for the starting position of VGA emulating CGA and EGA graphics centered in active video as a proportion of active H using the nominal start of video as a reference can be expressed as:

$$GH_{start} = \frac{\left(\frac{(P_{active} - P_{displayed}) \times P_{width}}{2}\right)}{H_{active}} \qquad \text{Equation 3}$$

b. Similarly, the general equation for determining the ending position of VGA graphics emulating CGA and EGA graphics as a proportion of active H can be expressed as:

$$GH_{end} = \frac{\left(\frac{(P_{active} - P_{displayed}) \times P_{width}}{2}\right) + (P_{displayed} \times P_{width})}{H_{active}} \qquad \text{Equation 4}$$

c. These equations are identical to equations 1 and 2 in the previous paragraph - except that AV_{start} is now equal to zero and therefore dropped from the equations and that H_{total} has been changed to H_{active} where, from Figure G-2:

H_{active} = 52.656 microseconds

d. Solving equation 3 for the starting position of 640-pixel graphics yields:

$$GH_{start} = \frac{\left(\frac{(756 - 640) \times 0.06969}{2}\right)}{52.656} = 0.0768\ H_{active}$$

e. Solving equation 4 for the ending position of 640-pixel graphics yields:

$$GH_{end} = \frac{\left(\frac{(756 - 640) \times 0.06969}{2}\right) + (640 \times 0.06969)}{52.656} = 0.924\ H_{active}$$

40.3 <u>PAL Video</u>. Figure G-3 shows the timing for one line of 15 KHz PAL video including the position of CGA and EGA 640- and 320-pixel graphics. The general equations used to calculate graphics positions for PAL are identical to those for NTSC. However, the values used for the equation variables differ because of differences in horizontal timing. For convenience, all equations and variable values are repeated in the

following sections. The timing values in the figure are either taken directly or derived from the accepted standard for PAL video. The calculations in this section are based on the values shown in the figure.

40.3.1 Position of true CGA and EGA graphics. To determine the correct position of true CGA and EGA graphics over video as a fraction of a horizontal, 15.625 KHz, PAL video signal given the timing shown in Figure G-3, use the following equations and assumptions for 640-pixel graphics. The resulting positions apply equally to 320-pixel graphics.

a. The general equation for the starting position of true CGA and EGA graphics that is centered in active video as a proportion of total H, and using the start of horizontal sync as a reference can be expressed as:

$$GH_{start} = \frac{AV_{start} + \left(\frac{(P_{active} - P_{displayed}) \times P_{width}}{2}\right)}{H_{total}} \qquad \text{Equation 5}$$

b. Similarly, the general equation for determining the ending position of true CGA and EGA graphics as a proportion total H width can be expressed as:

$$GH_{end} = \frac{AV_{start} + \left(\frac{(P_{active} - P_{displayed}) \times P_{width}}{2}\right) + (P_{displayed} \times P_{width})}{H_{Total}} \qquad \text{Equation 6}$$

c. For PAL video and 640-pixel resolutions:

(1) From Figure G-3, the nominal start of active video is:

AV_{start} = 10.5 microseconds.

(2) From Figure G-3 and the de facto standard (see 40.3.1), the total number of pixels corresponding to the width of active video is:

$$P_{active} = \left(\frac{51.95\mu s}{64.0\mu s}\right) \times 912\ pixels \approx 740\ pixels$$

(3) From the de facto standard, the total number of displayed pixels is:

$P_{displayed}$ = 640 pixels.

(4) From Figure G-3, the total width of the video signal is:

H_{total} = 64.0 microseconds.

(5) The width of one pixel is:

$$P_{width} = \left(\frac{64.0\ \mu s}{912}\right) = 0.07018\mu s$$

d. Solving equation 5 for the starting position of 640-pixel graphics yields:

$$GH_{start} = \frac{10.5 + \left(\frac{(740 - 640) \times 0.07018}{2}\right)}{64.0} = 0.2189H_{total}$$

e. Solving equation 6 for the ending position of 640-pixel graphics yields:

$$GH_{end} = \frac{10.5 + \left(\frac{(740 - 640) \times 0.07018}{2}\right) + (640 \times 0.07018)}{64.0} = 0.9207\ H_{total}$$

f. For PAL video these values equate to left and right border widths of 3.509 microseconds and a start of active graphics at approximately 14.0 microseconds after the start of horizontal sync. The latter value can be reliably verified with an accurate oscilloscope.

40.3.2 Position of VGA graphics emulating CGA and EGA modes. True VGA graphics map to the edges of active video and require no calculations. However, the starting and ending positions of VGA graphics emulating CGA and EGA graphics must be calculated. To determine the correct position of such graphics over video as a fraction of a horizontal, 15.625 KHz, PAL video signal given the timing shown in Figure G-3, use the following equations and assumptions for 640-pixel graphics. The resulting positions hold for 320-pixel graphics.

a. The general equation for the starting position of VGA emulating CGA and EGA graphics centered in active video as a proportion of active H, and using the nominal start of video as a reference can be expressed as:

$$GH_{start} = \frac{\left(\frac{(P_{active} - P_{displayed}) \times P_{width}}{2}\right)}{H_{active}} \qquad \text{Equation 7}$$

b. Similarly, the general equation for determining the ending position of VGA graphics emulating CGA and EGA graphics as a proportion of active video can be expressed as:

$$GH_{end} = \frac{\left(\frac{(P_{active} - P_{displayed}) \times P_{width}}{2}\right) + (P_{displayed} \times P_{width})}{H_{active}} \qquad \text{Equation 8}$$

c. These equations are identical to equations 5 and 6 in 40.3.3.1, except that AV_{start} is now equal to zero and therefore dropped from the equations and that H_{total} has been changed to H_{active} where, from Figure G-3:

H_{active} = 51.95 microseconds.

d. Solving equation 7 for the starting position of 640-pixel graphics yields:

$$GH_{start} = \frac{\left(\frac{(740 - 640) \times 0.07018}{2}\right)}{51.95} = 0.0675\ H_{active}$$

e. Solving equation 8 for the ending position of 640-pixel graphics yields:

$$GH_{end} = \frac{\left(\frac{(740 - 640) \times 0.07018}{2}\right) + (640 \times 0.07018)}{51.95} = 0.9321 H_{active}$$

50. VERTICAL POSITIONS

50.1 Vertical positions. This paragraph describes how to determine the start and end of active graphics relative to vertical video timing. The vertical position of graphics can be expressed both in lines and as a proportion of vertical timing, which is abbreviated V. Using proportions simplifies determining positions for scan-altered systems.

50.1.1 General assumptions. The following general assumptions are used in determining the vertical positions of active graphics relative to NTSC and PAL video.

a. The optimal position for active graphics is centered vertically in the active video. The basis for this assumption is nominal common practice. However, note that variations in graphics positioning due to monitor centering adjustments cannot be accounted for in this appendix.

b. Vertical resolution is expressed relative to fields, not frames. The VGA 480-line mode converts to 240 lines for single-field computations.

50.2 NTSC Video. Figure G-4 shows the timing for 15 KHz NTSC video including the position of 200-line graphics. The timing values in the figure are either taken directly or derived from the accepted standard for NTSC video. The calculations in this paragraph are based on the values shown in the figure.

50.2.1 Position of graphics in lines relative to vertical sync. Use the following equations and assumptions to determine the correct position of graphics over video for a 15.734 KHz, NTSC video signal. Vertical position is described as the number of lines relative to the start of vertical sync given the timing shown in Figure G-4. The calculations can easily be modified for different numbers of lines.

a. The NTSC standard specifies a vertical blanking interval of 21 lines with error limits of +0 and -1 lines, and a total vertical field of 262.5 lines. The calculations below assume a blanking interval of 21 lines, an active video height of 241.5 lines, and a nominal start of active video at 18 lines from the start of vertical sync.

b. The general equation for the starting position of graphics centered in active video as the number of lines relative to the start of vertical sync can be expressed as:

$$GV_{start} = AV_{start} + \left(\frac{(V_{active} - G_{active})}{2}\right) \qquad \text{Equation 9}$$

c. Similarly, the general equation for the ending position of graphics as the number of lines relative to the start of vertical sync can be expressed as:

$$GV_{end} = AV_{start} + \left(\frac{(V_{active} - G_{active})}{2}\right) + G_{active} \qquad \text{Equation 10}$$

d. GV stands for "graphics vertical." For NTSC video:

(1) The nominal start of active video from Figure G-4 is:

AV_{start} = 18 lines.

(2) The number of active video lines from Figure G-4 is:

V_{active} = 241.5 lines.

(3) The number of active graphics lines is either:

G_{active} = 200 lines

for 640 X 200 and 320 X 200 graphics, or:

G_{active} = 240 lines

for 640 X 480 graphics (see 50.1.1b).

50.2.1.1 Starting and ending positions for 200-line graphics.

a. Solving equation 9 for the start of 200-line graphics yields:

$$GV_{start} = 18 + \left(\frac{241.5 - 200}{2}\right) = 39 \textit{ lines}$$

b. Solving equation 10 for the end of 200-line graphics yields:

$$GV_{end} = 18 + \left(\frac{241.5 - 200}{2}\right) + 200 = 239 \textit{ lines}$$

50.2.1.2 Starting and ending positions for 240-line (640 X 480) graphics.

a. Solving equation 9 for the start of 240-line graphics yields:

$$GV_{start} = 18 + \left(\frac{241.5 - 240}{2}\right) = 18.75 \textit{ lines}$$

b. Solving equation 10 for the end of 240-line graphics yields:

$$GV_{end} = 18 + \left(\frac{241.5 - 240}{2}\right) + 240 = 258.75 \textit{ lines}$$

c. Given the derived values above, the VGA 240-line mode actually leaves a border of 1 line and 0.5 lines. To avoid screen disturbance, hardware implementers may want to blank these borders. Although this is not a compliance requirement, it is recommended.

50.2.2 Position of graphics as a proportion of total video. To calculate the position of graphics as a proportion of total vertical timing, simply change equations 9 and 10 to yield:

$$GV_{start} = \frac{AV_{start} + \left(\frac{(V_{active} - G_{active})}{2}\right)}{V_{total}} \qquad \text{Equation 11}$$

$$GV_{end} = \frac{AV_{start} + \left(\frac{(V_{active} - G_{active})}{2}\right) + G_{active}}{V_{total}} \qquad \text{Equation 12}$$

All terms are defined in the previous section except V_{total}, which from Figure G-4 for NTSC video is:

$$V_{total} = 262.5 \text{ lines}$$

50.2.2.1 Starting and ending positions for 200-line graphics. Borrowing from the previous section, the starting and ending positions for 200-line graphics as proportions of total V are simply:

$$GV_{start} = \frac{39}{262.5} = 0.1486\ V_{total}$$

$$GV_{end} = \frac{239}{262.5} = 0.9105\ V_{total}$$

50.2.2.2 Starting and ending positions for 240-line (640 X 480) graphics. Borrowing from the previous paragraph, the starting and ending positions for 240-line graphics as proportions of total V are simply:

$$GV_{start} = \frac{18.75}{262.5} = 0.0714\ V_{total}$$

$$GV_{end} = \frac{258.75}{262.5} = 0.9857\ V_{total}$$

50.2.3 Position of graphics as a proportion of active video. To calculate the position of graphics as a proportion of active vertical timing using the nominal start of video as a reference, simply change equations 11 and 12 to yield:

$$GV_{start} = \frac{\left(\frac{(V_{active} - G_{active})}{2}\right)}{V_{active}} \qquad \text{Equation 13}$$

$$GV_{end} = \frac{\left(\frac{(V_{active} - G_{active})}{2}\right) + G_{active}}{V_{active}} \qquad \text{Equation 14}$$

All terms are defined in previous paragraphs except V_{active}, which is for NTSC video from Figure G-4:

V_{active} = 241.5 lines.

50.2.3.1 Starting and ending positions for 200-line graphics. Borrowing from previous paragraphs, the starting and ending positions for 200-line graphics as proportions of active V are simply:

$$GV_{start} = \frac{\left(\frac{(241.5 - 200)}{2}\right)}{241.5} = 0.0859\ V_{active}$$

$$GV_{end} = \frac{\left(\frac{(241.5 - 200)}{2}\right) + 200}{241.5} = 0.9141\ V_{active}$$

50.2.3.2 Starting and ending positions for 240-line (640 X 480) graphics. Borrowing from previous paragraphs, the starting and ending positions for 240-line graphics as proportions of active V are:

$$GV_{start} = \frac{\left(\frac{(241.5 - 240)}{2}\right)}{241.5} = 0.0031\ V_{active}$$

$$GV_{end} = \frac{\left(\frac{(241.5 - 240)}{2}\right) + 240}{241.5} = 0.9969\ V_{active}$$

50.3 PAL Video. Figure G-5 shows the timing for a 15 KHz PAL video signal, including the position of 200-line graphics.

a. The timing values in the figure are either taken directly or derived from the accepted standard for PAL video. The calculations in this section are based on the values shown in the figure.

b. The general equations used to calculate graphics positions for PAL are identical to those for NTSC. However, the values used for the equation variables differ because of differences in vertical timing. For convenience, all equations and variable values are repeated in the following sections.

50.3.1 Position of graphics in lines relative to vertical sync. To determine the correct position of graphics over video as the number of lines relative to vertical sync for a 15.625 KHz, PAL video signal, and given the timing shown in Figure G-5, use the following equations and assumptions. Note that the calculations can easily be modified for different numbers of graphics lines.

a. The general equation for the starting position of graphics centered in active video as the number of lines relative to the start of vertical sync can be expressed as:

$$GV_{start} = AV_{start} + \left(\frac{(V_{active} - G_{active})}{2}\right) \qquad \text{Equation 15}$$

b. Similarly, the general equation for the ending position of graphics as the number of lines relative to the start of vertical sync can be expressed as:

$$GV_{end} = AV_{start} + \left(\frac{(V_{active} - G_{active})}{2}\right) + G_{active} \qquad \text{Equation 16}$$

c. GV stands for "graphics vertical." For PAL video:

(1) From Figure G-5, the nominal start of active video is:

AV_{start} = 22.5 lines

(2) Also from Figure G-5, the number of active video lines is:

V_{active} = 287.5 lines

(3) The number of active graphics lines for a 640 X 200 or 320 X 200 graphics is:

G_{active} = 200 lines

(4) The number of active graphics lines for a 640 X 480 graphics and various special PAL modes is:

G_{active} = 240 lines

50.3.1.1 Starting and ending positions for 200-line graphics.

a. Solving equation 15 for the start of 200-line graphics yields:

$$GV_{start} = 22.5 + \left(\frac{287.5 - 200}{2}\right) = 66 \ lines$$

b. Solving equation 16 for the end of 200-line graphics yields:

$$GV_{end} = 22.5 + \left(\frac{287.5 - 200}{2}\right) + 200 = 266 \ lines$$

50.3.1.2 Starting and ending positions for 240-line graphics.

a. Solving equation 15 for the start of 240-line graphics yields:

$$GV_{start} = 22.5 + \left(\frac{287.5 - 240}{2}\right) = 46 \ lines$$

b. Solving equation 16 for the end of 240-line graphics yields:

$$GV_{end} = 22.5 + \left(\frac{287.5 - 240}{2}\right) + 240 = 286 \ lines$$

50.3.2 Position of graphics as a proportion of total video. All terms are defined in the previous paragraph except V_{total}, which for PAL video is:

V_{total} = 312.5 lines

To calculate the position of graphics as a proportion of total vertical timing, simply change equations 15 and 16 to yield:

$$GV_{start} = \frac{AV_{start} + \left(\frac{(V_{active} - G_{active})}{2}\right)}{V_{total}} \qquad \text{Equation 17}$$

$$GV_{end} = \frac{AV_{start} + \left(\frac{(V_{active} - G_{active})}{2}\right) + G_{active}}{V_{total}}$$ Equation 18

50.3.2.1 Starting and ending positions for 200-line graphics. Borrowing from the previous paragraph, the starting and ending positions for 200-line graphics as proportions of total V timing are simply:

$$GV_{start} = \frac{66}{312.5} = 0.2112\ V_{total}$$

$$GV_{end} = \frac{266}{312.5} = 0.8512\ V_{total}$$

50.3.2.2 Starting and ending positions for 240-line graphics. Borrowing from the previous paragraph, the starting and ending positions for 240-line graphics as proportions of total V are simply:

$$GV_{start} = \frac{46}{312.5} = 0.1472\ V_{total}$$

$$GV_{end} = \frac{286}{312.5} = 0.9152\ V_{total}$$

50.3.3 Position of graphics as a proportion of active video. To calculate the position of graphics as a proportion of active vertical timing using the nominal start of video as a reference, simply change equations 17 and 18 to yield the following equations. All terms are defined in previous paragraphs, except V_{active}. For PAL video: V_{active} = 287.5 lines.

$$GV_{start} = \frac{\left(\frac{(V_{active} - G_{active})}{2}\right)}{V_{Active}} \qquad \text{Equation 19}$$

$$GV_{end} = \frac{\left(\frac{(V_{active} - G_{active})}{2}\right) + G_{active}}{V_{active}} \qquad \text{Equation 20}$$

50.3.3.1 Starting and ending positions for 200-line graphics. Borrowing from previous paragraphs, the starting and ending positions for 240-line graphics as proportions of active V are:

$$GV_{start} = \frac{\left(\frac{(287.5 - 200)}{2}\right)}{287.5} = 0.1522\ V_{active}$$

$$GV_{end} = \frac{\left(\frac{(287.5 - 200)}{2}\right) + 200}{287.5} = 0.8478\ V_{active}$$

50.3.3.2 Starting and ending positions for 240-line graphics. Borrowing from previous sections, the starting and ending positions for 240-line graphics as proportions of active V are:

$$GV_{start} = \frac{\left(\frac{(287.5 - 240)}{2}\right)}{287.5} = 0.0826\ V_{active}$$

$$GV_{end} = \frac{\left(\frac{(287.5 - 240)}{2}\right) + 240}{287.5} = 0.9174\ V_{active}$$

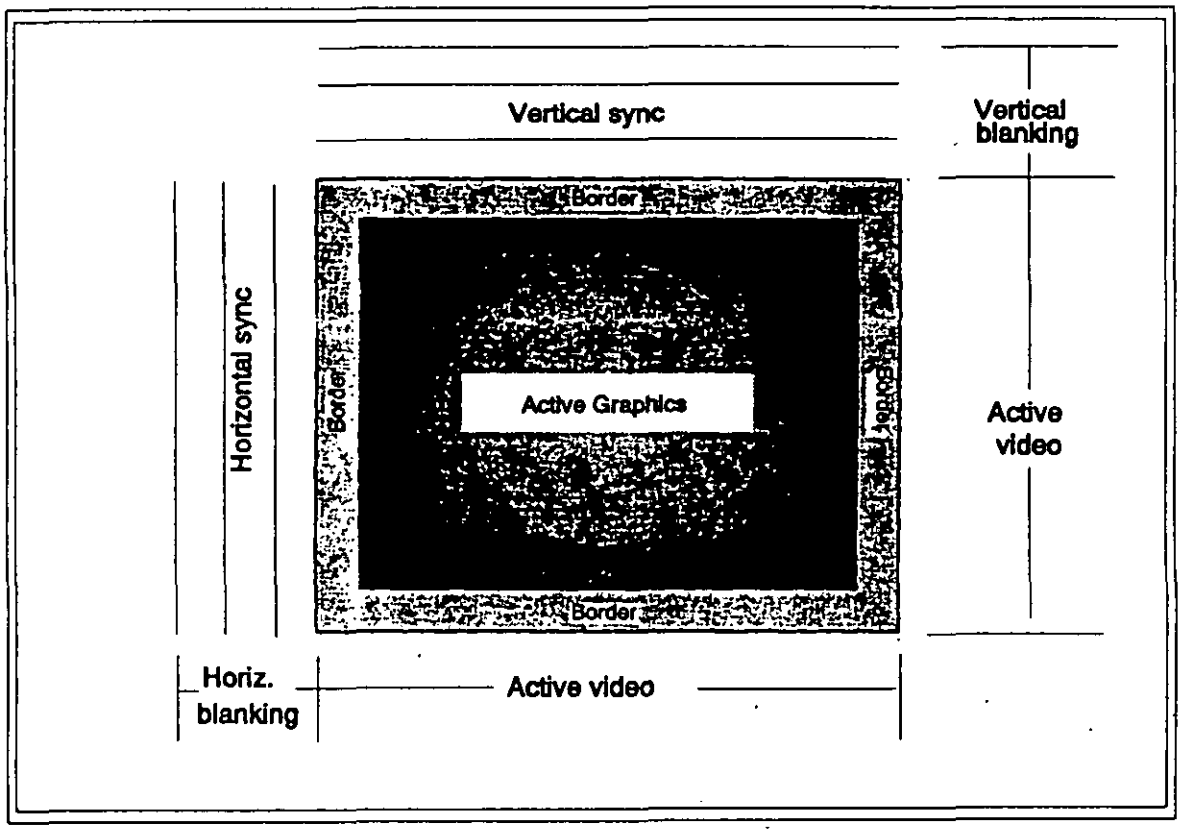

FIGURE G-1. Simplified diagram of an overlay display using CGA or EGA graphics.

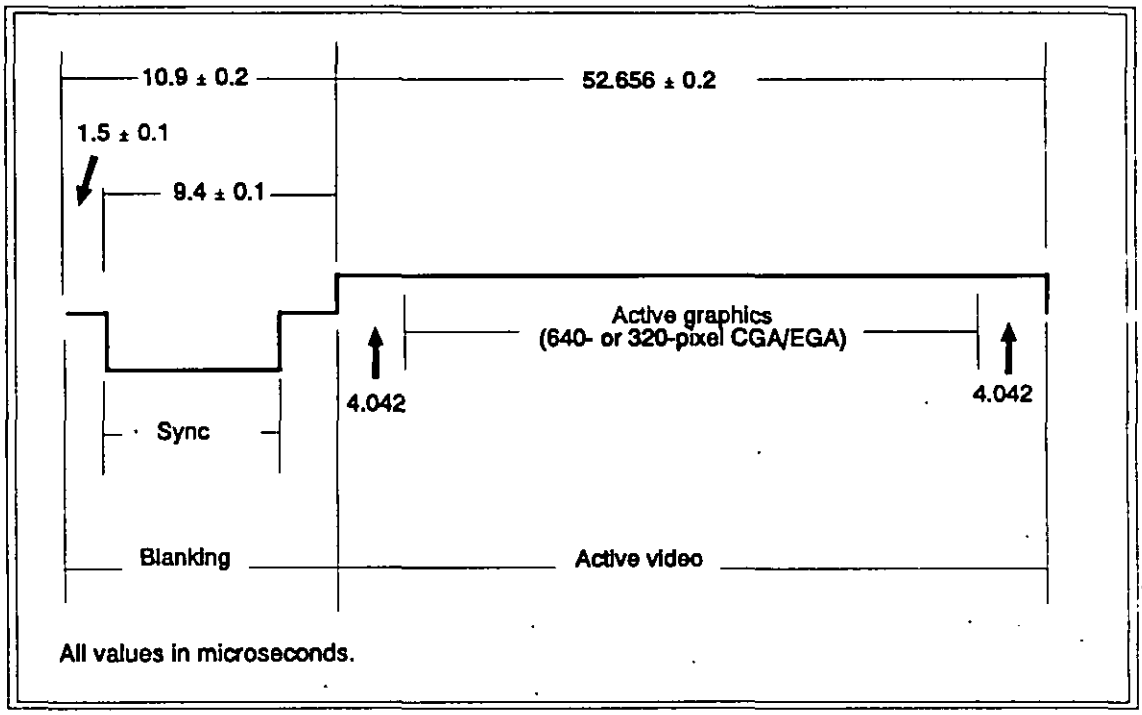

FIGURE G-2. One horizontal line of NTSC video with 640- or 320-pixel overlay graphics.

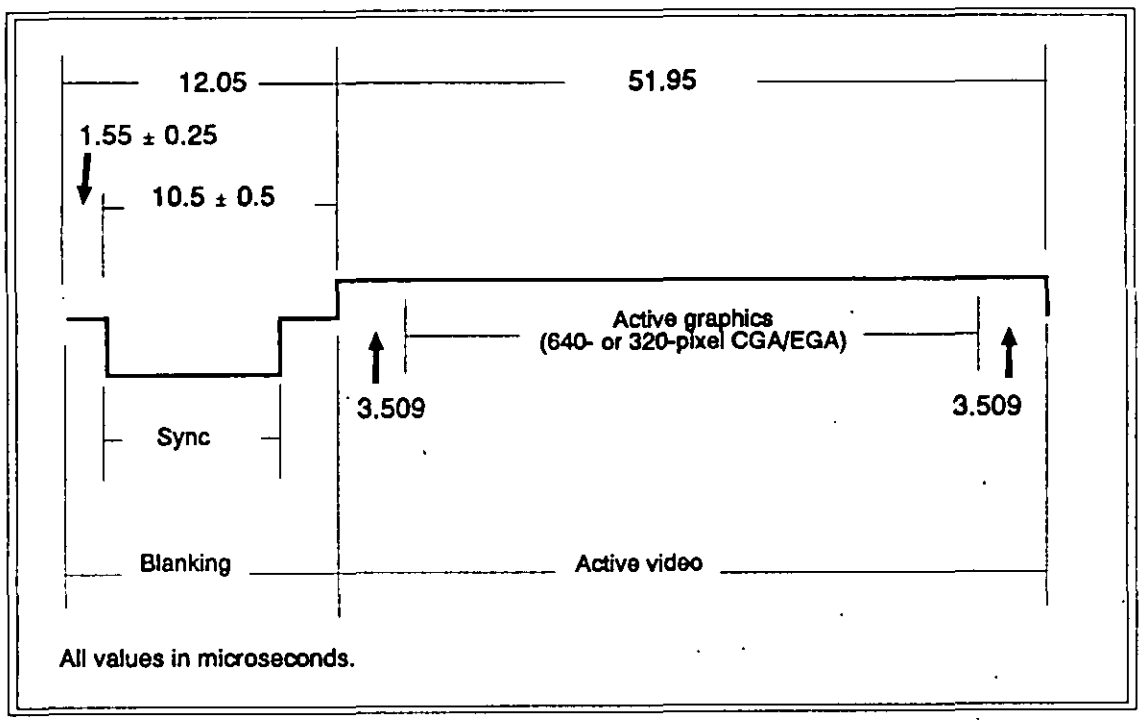

FIGURE G-3. One horizontal line of PAL video with 640- and 320-pixel overlay graphics.

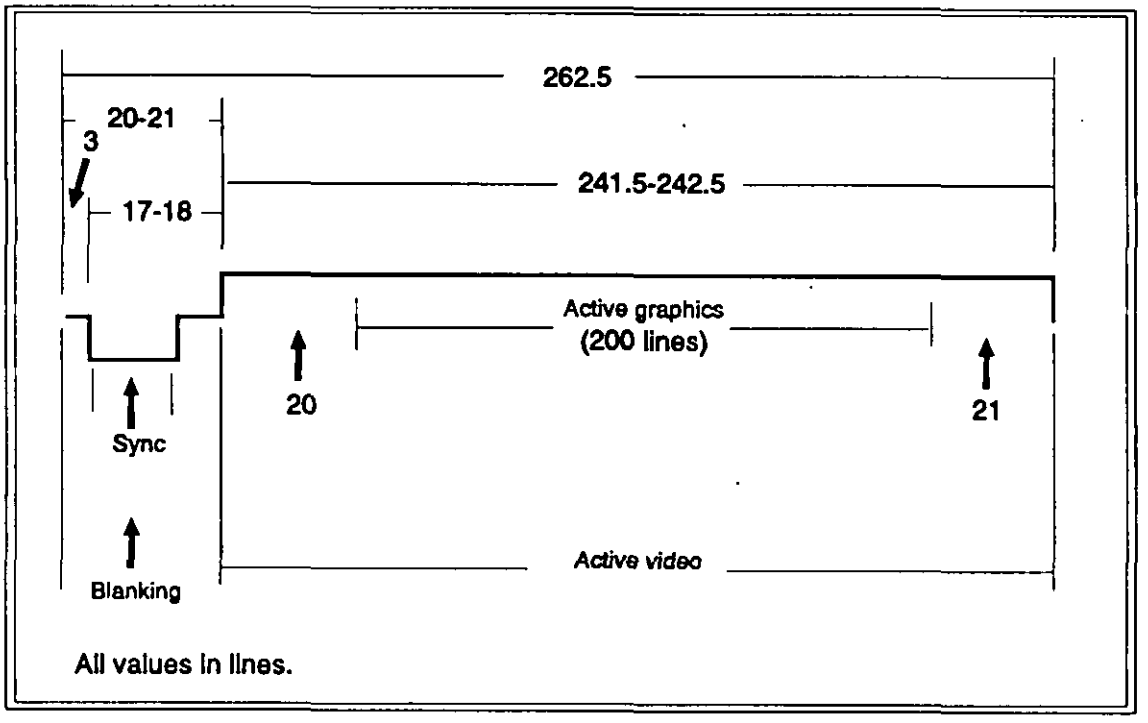

FIGURE G-4. NTSC vertical timing with 200-line overlay graphics.

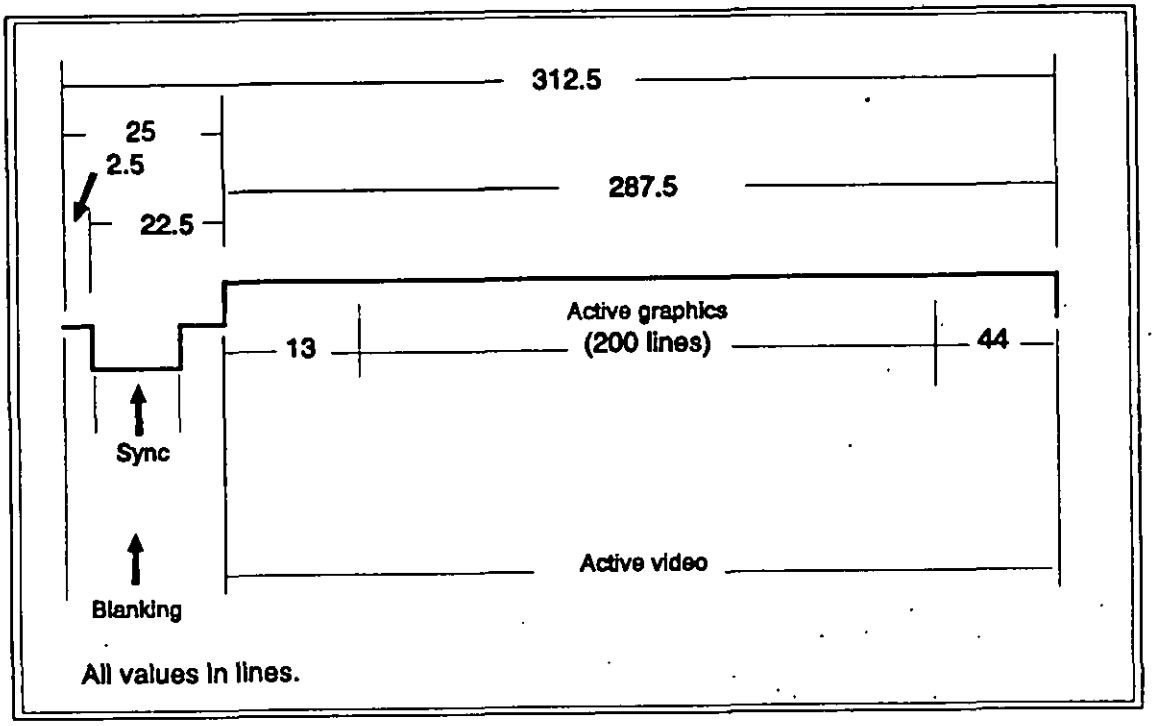

FIGURE G-5. PAL vertical timing with 200-line overlay graphics.

This Page Intentionally Left Blank.

APPENDIX H

ICW PORTABILITY PRACTICES ERROR HANDLING

10. SCOPE

10.1 Scope. This appendix describes error numbers and associated error messages used in the ICW portability practices (see 5.5). Error handling procedures in this appendix support the DoD Software Interface and Command Requirements for Interactive Courseware and Authoring Systems prescribed by Appendix D, MIL-STD-1379.

10.1.1 Terms, abbreviations, and acronyms used in this appendix. Key terms, abbreviations, and acronyms used in this appendix are defined as specified in Section 3 of the basic handbook.

20. APPLICABLE DOCUMENTS

20.1 Government documents.

20.1.1 Specifications, standards, and handbooks. The following specifications, standards, and handbooks form a part of this appendix to the extent specified herein.

STANDARD

MILITARY

MIL-STD-1379 Military Training Programs

(Unless otherwise specified, copies of military specifications, standards and handbooks are available from the Standardization Documents Order Desk, Building 4D, 700 Robbins Avenue, Philadelphia, PA 19111-5094.)

APPENDIX H

30. ERROR HANDLING

30.1 Introduction. When an application issues a command, VDI Management may be unable to carry out the requested action, or return the requested information. This causes an error. This appendix describes the error codes and definitions that VDI Management should return to the application.

30.2 Error listings. The following sections of this appendix describe error numbers and summary messages. These sections list the error codes in numerical order, and briefly explain error codes and messages. The sections group related errors for convenience and do not imply a corresponding programming structure in VDI Management implementations. However, implementations should use the error numbers as they are defined for portability.

30.3 Return message format. Error returns by the ASCII interface are all upper case and do not include the error text or a terminal period, even though these are used in the headings to the error descriptions that follow in this appendix. An error return consists of the word "ERROR" followed by a space, followed by the error number in decimal digits and a terminal CR/LF. For example, the error in Section 40.1 below, "Error 1 - SERVICE GROUP NOT INSTALLED," would be returned as "ERROR 1". (A return of "OK" indicates success.) The text for the error number can be retrieved with the extended syErrorMsg command, if supported. For example, issuing syErrorMsg errno = 1 would return "SERVICE GROUP NOT INSTALLED" (see Section 40 in Appendix A). The binary interface returns error numbers in the microprocessor's AX register (AX = 0 indicates success). Again, the extended syErrorMsg command can be used to retrieve the text for the error number.

40. COMMAND PROBLEMS.

40.1 Error 1 - SERVICE GROUP NOT INSTALLED. The VDI Management implementation supports the service group that contains the command, but the service group is not installed. For example, an application issued an xy command on a system that is not configured for XY-input devices.

40.2 Error 2 - UNKNOWN COMMAND. The command does not exist. Compliant VDI implementations should not return this error in response to any core command for a supported service group.

40.3 Error 3 - SYSTEM NOT INITIALIZED. The command was issued before the application issued syInit, or syStop was followed by a command other than syInit.

40.4 Error 15 - GENERAL COMMAND ERROR. A command error occurred that is not listed above or about which no information is available.

APPENDIX H

50. ASCII INTERFACE PROBLEMS.

50.1 Error 16 - BAD COMMAND SYNTAX. The parser encountered a fatal syntax error that could not be further diagnosed. For example, a command string contained a control code. Error 16 should not be used in place of parameter-problems errors.

50.2 Error 17 - COMMAND TOO LONG. The command was longer than 255 characters and should be ignored in its entirety. The terminal carriage return counts toward the total command length, but redundant delimiters do not.

50.3 Error 18 - RESPONSE TOO LONG. The response to an information request including the terminal CR/LF would be longer than 255 characters. VDI Management does not return partially filled information requests. For example, the application used an **xxGetState** command to request too much information.

50.4 Error 19 - DEVICE DRIVER READ BEFORE WRITE. The application tried to read a response from the device driver before it had written at least one command to it. This is not allowed and indicates a problem with the application's initialization code. This error can occur only immediately after VDI Management has been installed, or after a well-behaved application issued **syStop** before exiting.

50.5 Error 31 - GENERAL ASCII INTERFACE ERROR. An ASCII interface error has occurred that is not listed above or about which no information is available.

APPENDIX H

60. BINARY INTERFACE PROBLEMS

60.1 Error 32 - INVALID PARAMETER COUNT. On 80X86-based systems, the BX register contains an invalid or out of range parameter count. For example, the application passed a negative value in BX.

60.2 Error 33 - INVALID PARAMETER PACKET ADDRESS. On 80X86-based systems, the ES:DI register pair contains an invalid address for a parameter packet.

60.3 Error 34 - INVALID POINTER IN PARAMETER PACKET. The parameter packet contains a null or invalid pointer.

60.4 Error 47 - GENERAL BINARY INTERFACE ERROR. A binary interface error occurred that is not listed above or about which no information is available.

70. PARAMETER PROBLEMS

70.1 Error 48 - RESERVED. This error number is reserved for future use. The original error, Unknown parameter, was deleted because it required too much overhead. Error 52 should be used in place of "Unknown parameter."

70.2 Error 49 - INSUFFICIENT PARAMETERS. The command required one of the following:

a. a specific parameter that was missing,

b. at least one parameter from a specific group of parameters and was issued without that parameter, or

c. at least one parameter that could have been any parameter in its list and was issued with no parameters.

70.3 Error 50 - PARAMETERS CANNOT BE USED TOGETHER. The command included two or more parameters that cannot be used together. For example, a vdPlay command included both a direction and a to parameter.

70.4 Error 51 - PARAMETER VALUE INVALID OR OUT OF RANGE. The command included an incorrect parameter value. For example, a parameter value that must be in the range 0–255 was negative or greater than 255. This error can also result from the combined effects of two or more parameters, or from exceeding limits set by another parameter. For example, with the **vmSetPalette** command, the sum of the **color** and **length** parameters must be less than or equal to **logcolors**. **Logcolors** is the maximum number of available logical colors that can be retrieved using the vmGetState command.

70.5 Error 52 - PARAMETER INVALID FOR THIS COMMAND. The command included an invalid parameter label. For example, the application issued **sylnit** with a **color** parameter. This label could be also be an unknown parameter, or a value supplied for a parameter that does not take a value and, therefore, was interpreted as a label. Compliant VDI implementations should not return this error in response to any core parameter that is valid for a core command.

70.6 Error 53 - MISSING PARAMETER VALUE. The command failed to include a value for a parameter that requires one. The parser reached either the end of the command string or another parameter label when a parameter value was expected. This is an ASCII interface error only.

70.7 Error 54 - PARAMETER USED MORE THAN ONCE. The command included the same parameter more than once.

70.8 Error 79 - GENERAL PARAMETER ERROR. A parameter error occurred that is not listed above, or about which no information is available.

80. HARDWARE PROBLEMS

80.1 Error 80 - INITIALIZATION ERROR. The system could not initialize an attached device. The application can find out which device by examining the failed command.

80.2 Error 81 - DEVICE NOT INITIALIZED. The application tried to use either an uninitialized device or an uninitialized service group.

80.3 Error 82 - COMMUNICATIONS TIMEOUT. A timeout occurred while VDI Management was communicating with a peripheral device. Either the device did not produce an expected message within a predetermined timeout period, or the computer was unable to send a message to the device because signal control lines were in an inappropriate state. For example, this error would result from a cable being unplugged after a device has been initialized.

80.4 Error 83 - COMMUNICATIONS ERROR. An error occurred during communications. For example, repeated parity errors that cannot be cleared during asynchronous serial communications may cause this error.

80.5 Error 84 - DEVICE REPORTS ERROR. A peripheral device sent a message indicating that an error occurred that it cannot clear.

80.6 Error 85 - DEVICE CANCELED REQUEST. A peripheral device sent a message indicating that it has unilaterally canceled a requested service.

80.7 Error 86 - DEVICE NOT READY. A peripheral device sent a message indicating that it cannot be made operational.

80.8 Error 87 - ACTION NOT SUPPORTED BY DEVICE. A peripheral device sent a message indicating that it cannot do a requested action. This error indicates either an installation problem or the inappropriate use of the vdPassThru command. Compliant systems should not normally generate this error.

80.9 Error 88 - UNABLE TO RETURN REQUESTED INFORMATION. A hardware device could not generate information requested by a command. For example, a CLV videodisc could not report a frame number.

80.10 Error 111 - GENERAL HARDWARE ERROR. A hardware error occurred that is not listed above or about which no information is available.

90. SYSTEM RESOURCES (Some compliant systems may not be able to return some of the errors listed in this group.)

90.1 Error 112 - INSUFFICIENT MEMORY. VDI Management could not access enough memory to perform the requested service.

90.2 Error 113 - NEEDED HARDWARE INTERRUPT IN USE. VDI Management requires the use of a specific hardware interrupt that is already in use, or one of a range of interrupts and all are in use.

90.3 Error 114 - NEEDED SOFTWARE INTERRUPT IN USE. VDI Management requires the use of a specific software interrupt that is already in use, or one of a range of interrupts and all are in use.

90.4 Error 115 - NEEDED DMA CHANNEL NOT AVAILABLE. VDI Management requires the use of a specific DMA channel that is already in use, or requires the use of any DMA channel and all are in use.

90.5 Error 116 - NEEDED TIMER NOT AVAILABLE. VDI Management requires the use of a timer resource that is not available.

90.6 Error 127 - GENERAL RESOURCES ERROR. VDI Management requires additional system resources that are not listed above, or about which no information is available.

100. FILING SYSTEM PROBLEMS

100.1 Error 128 - INVALID FILENAME. The command used a filename that was invalid for the operating system. VDI Management should return error 132 when a valid file name can not be opened.

100.2 Error 129 - INVALID PATH. The command used a path name that was invalid for the operating system.

100.3 Error 130 - INVALID DRIVE. The command specified a drive that is not recognized by the operating system.

100.4 Error 131 - INVALID FILE NUMBER. The command used a file number that was not recognized by the operating system.

100.5 Error 132 - CANNOT OPEN OR CREATE FILE. The operating system could not open or create a requested file.

100.6 Error 133 - CANNOT CLOSE FILE. The operating system could not close a requested file.

100.7 Error 134 - FILE ALREADY OPEN. The command tried to open a file that was already open.

100.8 Error 135 - FILE ALREADY EXISTS. The command tried to create a file that already exists.

100.9 Error 136 - FILE DOES NOT EXIST. The command tried to access a file that does not exist.

100.10 Error 137 - FILE ACCESS DENIED. The command was denied access to a requested file. For example, a file with a locked status on a network file server would cause this error.

100.11 Error 138 - FILE SEEK ERROR. The command tried to use a nonexistent piece of a file. For example, a command tried to access byte 9000 of a 5-KB file.

100.12 Error 139 - TOO MANY OPEN FILES. The operating system has run out of file handles because too many files are open. Either the application should open fewer files, or the user should change the operating system installation to allow more files to be open simultaneously.

100.13 Error 140 - DISK FULL. The command tried to write to a full disk. The user should delete some files or change to a different disk before trying to run the application again.

100.14 Error 141 - DISK READ ERROR. A data error occurred while reading the disk.

100.15 Error 142 - DISK WRITE ERROR. A data error occurred while writing to the disk.

100.16 Error 159 - GENERAL FILING-SYSTEM ERROR. A filing-system error occurred that is not listed above or about which no information is available.

110. MISCELLANEOUS PROBLEMS

110.1 Error 160 - INVALID DEVICE NUMBER. The command specified an invalid device or source number. This error results from using an invalid number for a device or source parameter or from trying to change the default device or source to an invalid number.

110.2 Error 161 - BUFFER OVERFLOW. An internal VDI Management buffer overflowed. This indicates an internal VDI Management failure and should be brought to the attention of the system vendor.

110.3 Error 162 - INTERNAL CALCULATION ERROR. An error such as divide by zero occurred during a numeric calculation within VDI Management. This indicates an internal VDI Management failure and should be brought to the attention of the VDI Management developer.

110.4 Error 163 - COPY PROTECTION ERROR. A copy protected version of VDI Management has declined to run because its protection scheme has been violated. Normally, DoD data rights specified in ICW and ICW Training System contracts prohibit copy protection schemes. Legitimate users should discuss this problem with the appropriate contracting officer and VDI Management developer.

110.5 Error 164 - INTERFACE BUSY. The ASCII or binary interface was busy and could not accept the command, and the application should reissue the command. This error is returned only when an application calls VDI Management and a previous call to VDI Management has not yet returned. This error is possible only if an application issues VDI Management calls from within an interrupt routine.

110.6 Error 165 - INVALID INTERRUPT NUMBER. The environment variable IVINT was set to an interrupt number outside the range from 60H through 66H.

110.7 Error 173 - GENERAL INTERNAL ERROR. An internal VDI Management error occurred that is not listed above or about which no information is available. This indicates an internal VDI Management failure and should be brought to the attention of the VDI Management developer.

110.8 Error 174 - GENERAL OPERATING SYSTEM ERROR. The operating system reported an error unrelated to the filing system and not specific to any particular aspect of VDI Management. VDI Management should try to recover from this error before returning it to the application.

110.9 Error 175 - GENERAL ERROR. An error occurred that is not listed elsewhere and about which no information at all is available. This error differs from error 111 - General hardware error. Error 111 guarantees that a hardware error has occurred while

this error can be caused by any unknown failure including unknown hardware, VDI Management, and application failures. VDI Management developers should not use this error number before carefully considering the use of a more appropriate and informative error code.

120. SYSTEM GROUP PROBLEMS

120.1 Error 176 - QUEUE FULL. The application tried to queue more than 10 commands. This indicates an application problem such as failing to turn syQueue off at the appropriate time.

120.2 Error 177 - COMMAND CANNOT BE QUEUED. The application tried to queue a command that cannot be queued (see Appendix A, Table A-7). This indicates an application problem such as failing to turn syQueue off at the appropriate time.

120.3 Error 191 - GENERAL SYSTEM ERROR. A problem occurred within the system group that is not listed above or about which no further information is available.

APPENDIX H

130. VISUAL-MANAGEMENT PROBLEMS

130.1 Error 192 - SYNCHRONIZATION ERROR. The video signal could not be genlocked to the computer's graphics because the signal has an inappropriate scan rate. Overlay is not possible.

130.2 Error 193 - GRAPHICS MODE PROBLEM. The system could not do a requested action because the graphics mode does not support it. For example, the application tried to turn on transparency in a graphics mode that does not support overlays.

130.3 Error 194 - UNSUPPORTED GRAPHICS MODE. The system could not switch to a requested graphics mode or emulation state because the hardware does not support the mode. For example, issuing vmSetGraphics emulation=0, which requires a VGA adapter, on a CGA or EGA system would cause this error. Issuing vmSetGraphics mode=14, which requires an EGA graphics adapter, on a CGA system would also cause this error (see 5.4).

130.4 Error 207 - GENERAL VISUAL-MANAGEMENT ERROR. A problem occurred with the visual management functions that is not listed above or about which no further information is available.

140. VIDEODISC PROBLEMS

140.1 Error 208 - ACTION NOT SUPPORTED BY DISC. The command requested an action that is not supported by the videodisc. For example the application tried to do a frame search on a CLV videodisc, or a chapter search on a videodisc that without chapter stops.

140.2 Error 209 - DISC NOT SPUN UP. The application issued a command such as **vdPlay** that requires the videodisc to be spun up, and the videodisc has not been initialized and spun up to operating speed.

140.3 Error 210 - DISC NOT SPUN DOWN. The application issued a command such as **vdSet door=1** that requires the videodisc to be spun down or stopped, and it has not been spun down.

140.4 Error 211 - DOOR OPEN. The application issued a videodisc motion command other than **vdSet door=0** with the player door open. Typically, this is a user error that can be corrected without exiting the application.

140.5 Error 212 - NO DISC IN TRAY. The application issued a videodisc motion command other than **vdSet door=1** with the player door closed but without a videodisc in the tray. Typically, this is a user error that can be corrected without exiting the application.

140.6 Error 213 - BAD DISC SECTION. It was impossible to seek to the required frame because of a physical problem with the videodisc.

140.7 Error 214 - FELL OFF DISC. An attempt was made to play backward past the beginning or forward past the end of the videodisc. This normally indicates improper use of a videodisc motion command.

140.8 Error 215 - INVALID FRAME NUMBER. The command specified a frame that is not present on the videodisc.

140.9 Error 216 - INVALID CHAPTER NUMBER. The command specified a chapter that is not present on the videodisc.

140.10 Error 217 - INVALID TIME CODE. The command specified a time code that is not present on the videodisc.

140.11 Error 239 - GENERAL VIDEODISC PLAYER ERROR. A videodisc error occurred that is not listed above or about which no information is available.

150. XY-INPUT DEVICE PROBLEMS

150.1 Error 240 - DEVICE NOT CALIBRATED. A required, implementation- specific, calibration process has not been done. The user should verify that the XY-input device is installed correctly.

150.2 Error 241 - INVALID COORDINATE. A coordinate was specified that is outside the acceptable range. This normally indicates an application problem.

150.3 Error 242 - CURSOR PROBLEM. A problem with the graphics device caused a cursor display problem. This indicates that the application requires a facility that VDI Management does not furnish.

150.4 Error 255 - GENERAL XY-INPUT ERROR. An XY-input error occurred that is not listed above or about which no information is available.

APPENDIX H

This Page Intentionally Left Blank.

APPENDIX I

APPLICATION PROGRAMMING EXAMPLES

10. SCOPE

10.1 Scope. This appendix provide several brief programming examples that use the ASCII and binary interfaces. The examples are intended only to furnish a starting point for programmers. They are not intended to be overly sophisticated or complete.

10.1.1 Terms, abbreviations, and acronyms used in this appendix. Key terms, abbreviations, and acronyms used in this appendix are defined as specified in Section 3 of the basic handbook.

20. APPLICABLE DOCUMENTS

20.1 Applicable documents. This section is not applicable to this appendix.

30. USING THE ASCII INTERFACE

30.1 Using the ASCII interface. Even programming systems with minimal facilities for interfacing to other languages can use an installable device driver. The two short programs provided below use the Microsoft GWBASIC interpreter. For clarity and brevity, these programs do not determine which service groups are present or always check for errors after issuing commands. However, well-behaved applications should do both.

30.1.1 BASIC example 1.

```
10 OPEN "O",#1,"ivdev"
20 PRINT #1, "syinit"
30 PRINT #1, "vminit"
40 PRINT #1, "vdinit"
50 PRINT #1, "xyinit"
50 PRINT #1,"vdplay start=1000,stop=2000,wait"
60 CLOSE #1
70 OPEN "I",#1,"ivdev"
80 INPUT #1,R$
90 CLOSE #1
100 IF R$="OK" THEN PRINT "Playing" ELSE PRINT "PROBLEMS!";R$
110 OPEN "O",#1,"ivdev"
120 PRINT #1, "systop"
130 CLOSE #1
140 END
```

30.1.2 BASIC example 2.

APPENDIX I

```
10 OPEN "O",#1,"ivdev"
20 PRINT #1, "syinit"
30 PRINT #1, "vminit"
40 PRINT #1, "vdinit"
50 PRINT #1, "xyinit"
60 PRINT #1,"vmGetPalette color=5,r,g,b"
70 CLOSE #1
80 OPEN "I",#1,"ivdev"
90 INPUT #1,R$
100 CLOSE #1
110 REM The next section of code would parse R$, which is
120 REM in the form "<r value>,<g value>,<b value>"
...
200 REM Now shut it down
210 OPEN "O",#1,"ivdev"
220 PRINT #1, "systop"
230 CLOSE #1
240 END
```

APPENDIX I

40. USING SOFTWARE INTERRUPT CALLS

40.1 Using software interrupt calls. Programming systems with library facilities for software interrupts can use direct software interrupt calls for issuing commands. The following example uses Microsoft C, version 5.1. It is a code fragment, not a complete program, and, as such, is not compilable.

```
/*
 * This example gets the values for a logical color.
 * Obviously, a well-structured program would hide the
 * interrupt call in a separate function.
 * The example omits typical start-up and shut-down code
 */

#include <stdio.h>
#include <dos.h>

#define VMGETPALETTE 2049
#define COLORPARM    9
#define RED          38
#define GREEN        25
#define BLUE         4

struct ivparm
{
 long parm_id;
 long parm_val;
} parms [20], far *p;
union REGS cpuregs;
struct SREGS segregs;

int iv_int = 0x60    /* software interrupt, normally */
                     /* read from environment        */
/*
 * The following code fragment would be included in
 * function blocks
 */

  /* initialize far pointer */
  p = parms;

  /* Set up parameter block */
  parms[0].parm_id  = COLORPARM;
  parms[0].parm_val = 5;
```

```
parms[1].parm_id  = RED;
parms[2].parm_id  = GREEN;
parms[3].parm_id  = BLUE;

/*
 * Set up CPU registers and call software interrupt.
 * cpuregs and segregs are structures for manipulating
 * CPU registers.  FP_OFF and FP_SEG are macros that
 * find the absolute address of a variable.
 */

cpuregs.x.ax = VMGETPALETTE;
cpuregs.x.bx = 4;          /* number of parameters */
cpuregs.x.di = FP_OFF(p);
segregs.es   = FP_SEG(p);

/*
 * int86x() is an MSC library function that calls
 * a software interrupt.
 */

int86x(iv_int,&cpuregs,&cpuregs,&segregs);

/* Check for errors*/

if(cpuregs.x.ax != 0)
  {
  printf("Error code: %d\n", (int) cpuregs.x.ax);
  exit(1);
  }

/* Print the color values */
printf("Color 5 Red,Green,Blue %d,%d,%d\n",
  (int) parms[1].parm_val, (int) parms[2].parm_val,
  (int) parms[3].parm_val);
```

APPENDIX I

50. LIBRARY CALLS WITH PARAMETER NUMBERS

50.1 Library calls with parameter numbers. Vendors may furnish libraries for specific languages that use parameter numbers of the binary interface stored in a data structure appropriate to the language. Several languages support functions that accept variable numbers of parameters. The following example uses Microsoft C 5.1 to show how to use variable numbers of parameters to set up a parameter block. It omits the code that would then execute the function.

```
#include <stdio.h>
#include <stdarg.h>     /* for ANSI compatibility */

/*
 * This function might be part of a support library.
 * va_start() and va_arg() are macros for accessing
 * variable-length argument lists.
 * (See MSC 5.1 manuals for details.)
 */

iv_vdplay(long parm1, ...)
{
  va_list argp;
  struct ivparm
  {
    long parm_id;
    long parm_val;
  } parms [20];
  int i;

  va_start(argp, parm1);

  /* Set up parameter block from variable arg list */

  for(i = 0; parms[i].parm_id != NULL && i << 20; i++)
    {
    parms[i].parm_id = va_arg(argp, long);
    parms[i].parm_val = va_arg(argp, long);
    }

  /* Now insert rest of code to execute command */

  ...
}
```

APPENDIX I

60. ANALYZING BIT FIELDS

60.1 Analyzing bit fields. It is simple to analyze bit fields with any language that supports bit-wise "and". The following example uses the Microsoft GWBASIC interpreter.

```
100 REM Start by initializing the system
110 OPEN "O",#1,"ivdev"
120 PRINT #1,"syInit"
130 CLOSE #1
140 OPEN "I",#1,"ivdev"
150 INPUT #1,R$
160 CLOSE #1
170 IF R$ < > "OK" THEN PRINT "Cannot initialize system"
    :GOTO 300
180 REM Now request the support information
190 OPEN "O",#1,"ivdev"
200 PRINT #1,"syGetState"
210 CLOSE #1
220 OPEN "I",#1,"ivdev"
230 INPUT #1,R$
240 CLOSE #1
250 REM R$ now contains the decimal string which
260 represents the support bit field
270 R=VAL(R$)
280 REM Do the "AND" to check whether xy is supported
290 IF R AND 8 THEN PRINT "XY input is supported"
    ELSE PRINT "XY input is not supported"
300 OPEN "O",#1,"ivdev"
310 PRINT #1,"syStop"
320 CLOSE #1
330 END
```

APPENDIX I

70. DETERMINING THE NUMBER OF LOGICAL DEVICES

70.1 Determining the number of logical devices. The following example shows how to determine the number of logical devices installed for the videodisc service group. The same technique could be used for any service group that supports multiple devices. This example also uses the Microsoft GWBASIC interpreter.

```
100 REM Start by initializing the system
110 OPEN "O",#1,"ivdev"
120 PRINT #1,"sylnit"
130 CLOSE #1
140 OPEN "I",#1,"ivdev"
150 INPUT #1,R$
160 CLOSE #1
170 IF R$ < > "OK" THEN PRINT "Cannot initialize system"
   :GOTO 500
180 REM Get installed service group
190 OPEN "O", #1,"ivdev"
200 PRINT #1,"syGetState support"
210 CLOSE #1
220 OPEN "I",#1,"Ivdev"
230 INPUT #1,R$
240 CLOSE #1
250 REM R$ now contains the decimal string which represents the support
260 REM bit field. Determine if the vd service group is installed.
270 GROUPS = VAL(R$)
280 IF GROUPS AND 4 THEN PRINT "VD is supported"
   ELSE PRINT "VD is not supported" : GOTO 500
290 REM Initialize the vd service group
300 OPEN "O",#1,"ivdev"
310 PRINT #1, "vdInit"
320 CLOSE #1
330 OPEN "I",#1,"ivdev"
340 INPUT #1,R$
350 CLOSE #1
360 IF R$ < > "OK" THEN PRINT "Cannot initialize VD service Group"
   :GOTO 500
370 REM Now get the number of players
380 OPEN "O",#1,"ivdev"
390 PRINT #1, "vdGetState tdevices"
400 CLOSE #1
410 OPEN "I",#1,"ivdev"
420 INPUT #1,R$
430 CLOSE #1
```

```
440 PLAYERS = VAL(R$)
450 PRINT "The number of installed players is: "PLAYERS
500 OPEN "O",#1,"ivdev"
510 PRINT #1,"syStop"
520 CLOSE #1
530 END
```

APPENDIX J

ICW PORTABILITY PRACTICES HANDBOOK CROSS-REFERENCES

10. SCOPE

10.1 Scope. This appendix cross-references handbook information to equivalent and related information in MIL-STD-1379, Appendix D, the IMA Recommended Practices for Multimedia Portability (MS-DOS Based Systems), Release R1.1, and the IMA clarification document (CLAR 2) issued for the Recommended Practices. This appendix correlates handbook paragraphs, tables, and figures to equivalent or related requirements outlined in the standard and the IMA Recommended Practices. It cross-references ICW Portability Practices information and guidance in this handbook to the Software Interface and Command Requirements for Interactive Courseware and Authoring Systems prescribed by MIL-STD-1379, Appendix D and to the IMA Recommended Practices being implemented through this handbook.

10.2 How to use this appendix. This appendix provides cross-reference information through a series of cross-reference tables. There is a single cross-reference table for Sections 4 and 5, and a separate cross-reference table for each of the appendixes. Separate cross-reference tables are also provided for the handbook tables and figures.

a. The cross-reference tables only contain handbook paragraphs, tables, and figures that have an equivalent or related section, paragraph, table, or figure in the IMA Recommended Practices, or Appendix D of MIL-STD-1379, or both, or the IMA clarification document (CLAR 2).

b. Tables J-8, Cross-reference of handbook Appendix G, and J-10, Cross-reference of handbook Appendix I, do not apply to MIL-STD-1379, Appendix D. Therefore, these tables do not include a MIL-STD-1379 column.

c. The table heading "MIL-STD-1379 PARAGRAPH NUMBER" refers to the applicable paragraph in Appendix D, MIL-STD-1379. This is also true for Tables J-11 and J-12 cross-references to MIL-STD-1379 tables and figures, respectively.

d. Because the IMA Recommended Practices include several distinct paragraphs under each numbered section, several handbook paragraphs cross-reference to a single numbered section of the IMA document.

e. The IMA Recommended Practices do not number service group command information paragraphs, or the ASCII and binary interface information tables. Because of this, Tables J-2 through J-5 cross-reference IMA Recommended Practices to the applicable page number when a section or paragraph number is not available. Table J-11, Cross-reference of handbook tables, also references

the applicable IMA document page number that contains the equivalent ASCII and binary interface tables. The cross-reference to the IMA clarification document is to a paragraph number or table number.

10.2.1 Terms, abbreviations, and acronyms used in this appendix. Key terms, abbreviations, and acronyms used in this appendix are defined as specified in Section 3 of the basic handbook.

20. APPLICABLE DOCUMENTS.

20.1 Government documents.

20.1.1 Specifications, standards, and handbooks. The following specifications, standards, and handbooks form a part of this appendix to the extent specified herein.

STANDARD

MILITARY

MIL-STD-1379 Military Training Programs

(Unless otherwise specified, copies of military specifications, standards and handbooks are available from the Standardization Documents Order Desk, Building 4D, 700 Robbins Avenue, Philadelphia, PA 19111-5094.)

20.2 Non-Government publications. The following document forms a part of this document to the extent specified herein.

INTERACTIVE MULTIMEDIA ASSOCIATION (IMA)

Recommended Practices for Multimedia Portability, (MS-DOS Based Systems), Release R 1.1

The IMA Interactive Video SIG, "Clarifications for the Recommended Practices for Multimedia Portability (MS-DOS based systems) Release R 1.1, " Revision number: CLAR 2

(Application for copies should be addressed to the Interactive Multimedia Association (IMA), 3 Church Circle, Suite 800, Annapolis, MD 21401-1933.)

(Non-Government standards and other publications are normally available from the organizations that prepare or distribute the documents. These documents also may be available in or through libraries or other informational services.)

APPENDIX J

30. CROSS-REFERENCE TABLES

30.1 Cross-reference tables. The basic sections, appendixes, tables, and figures of the handbook are cross-referenced using 12 separate tables.

30.1.1 Sections 4 and 5 table. Table J-1 contains the basic cross-reference information for Sections 4 and 5 of the handbook.

30.1.2 Appendix A table. Table J-2 contains the cross-reference information for Appendix A.

30.1.3 Appendix B table. Table J-3 contains the cross-reference information for Appendix B.

30.1.4 Appendix C table. Table J-4 contains the cross-reference information for Appendix C.

30.1.5 Appendix D table. Table J-5 contains the cross-reference information for Appendix D.

30.1.6 Appendix E table. Table J-6 contains the cross-reference information for Appendix E.

30.1.7 Appendix F table. Table J-7 contains the cross-reference information for Appendix F.

30.1.8 Appendix G table. Table J-8 contains the cross-reference information for Appendix G.

30.1.9 Appendix H table. Table J-9 contains the cross-reference information for Appendix H.

30.1.10 Appendix I table. Table J-10 contains the cross-reference information for Appendix I.

30.1.11 Cross-reference of handbook tables. All handbook tables are cross-referenced in Table J-11. Table J-11 cross-reference information includes handbook tables contained in the various handbook appendixes.

30.1.12 Cross-reference of handbook figures. All handbook figures, including those in handbook appendixes, are cross-referenced in Table J-12.

APPENDIX J

TABLE J-1. Cross-reference of Sections 4 and 5.

HANDBOOK PARAGRAPH NUMBER	PARAGRAPH TITLE	IMA RELEASE R1.1 SECTION NUMBER	MIL-STD-1379 PARAGRAPH NUMBER
4.	GENERAL GUIDANCE	Not Applicable (N/A)	N/A
4.2.1	PORTCO/IMA Recommended Practices architecture	2.	40.2
4.3	Interactive courseware (ICW) portability practices	N/A	Appendix D
4.3.1	Introduction	1.	N/A
4.3.1.1	Scope of the ICW portability practices	1.1	
4.3.1.2	Goals of the ICW portability practices	1.2	
4.3.1.3	Benefits of the ICW portability practices		
4.3.1.4	Handbook conventions	1.4	30.2
4.3.2	System overview	2.	40.2
4.3.2.1	Hardware and operating system assumptions	2.1	40.1
4.3.2.2	Interface and command design criteria	2.2	40.5
4.3.2.3	The rationale for two interfaces	2.3	N/A
4.3.2.4	Service groups and command organization	2.4	40.3
4.3.2.5	Service groups		N/A
4.3.2.6	Core and extended commands and parameters	2.5	
4.3.2.6.1	Core commands and parameters		
4.3.2.6.2	Extended commands and parameters		
4.3.3	IBM PC and compatible graphics modes	Appendix B, B.1	80.3.1
4.3.3.1	Modes 0-3 restrictions	B.2	
4.3.4	Tracking IMA Recommended Practices	N/A	N/A
5.	DETAILED GUIDANCE	N/A	
5.1	Interface application	3.	40.5
5.1.1	The binary interface		40.5.1.2
5.1.2	The ASCII interface		40.5.1.1
5.1.3	Introduction to parameters and values	3.1	40.5.2
5.1.3.1	Parameter order		N/A

TABLE J-1. Cross-reference of Sections 4 and 5 - Continued.

HANDBOOK PARAGRAPH NUMBER	PARAGRAPH TITLE	IMA RELEASE R1.1 SECTION NUMBER	MIL-STD-1379 PARAGRAPH NUMBER
5.1.4	Using the binary interface	3.2	N/A
5.1.4.1	General procedure	3.2.1	
5.1.4.2	Confirming the binary interface	3.2.2	60.2
5.1.4.3	Parameter packets	3.2.3	60.4
5.1.4.3.1	Parameter token numbers		60.4.2
5.1.4.3.2	Parameter packet length		60.4.4
5.1.4.4	Return values to parameters	3.2.4	60.5
5.1.5	Using the ASCII interface	3.3	N/A
5.1.5.1	General procedure	3.3.1	
5.1.5.2	Confirming the ASCII interface	3.3.2	
5.1.5.3	Command strings	3.3.3	
5.1.5.4	Response strings	3.3.4	
5.1.6	Mixing ASCII and binary commands	3.4	40.5.3
5.2	Interface implementation	4.	N/A
5.2.1	Installation issues	4.1	
5.2.1.1	VDI Management installation	4.1.1	40.4
5.2.1.2	Logical device numbers	4.1.2	N/A
5.2.1.2.1	Multiple devices		
5.2.1.2.2	Assigning logical device numbers		40.4.1
5.2.1.2.3	Device mapping		40.4.2
5.2.2	Operating system issues	4.2	N/A
5.2.2.1	Operating system requirements	4.2.1	40.1.7
5.2.2.2	Specific version MS-DOS compliance		40.1.7
5.2.2.3	MS-DOS reentrancy limitations	4.2.2	N/A
5.2.2.4	Background processing		
5.2.3	ASCII interface issues	4.3	50

TABLE J-1. Cross-reference of Sections 4 and 5 - Continued.

HANDBOOK PARAGRAPH NUMBER	PARAGRAPH TITLE	IMA RELEASE R1.1 SECTION NUMBER	MIL-STD-1379 PARAGRAPH NUMBER
5.2.3.1	ASCII string formats	4.3.1	50.1
5.2.3.1.1	Command string tokens		50.1.1
5.2.3.1.2	Command string delimiters		50.1.2
5.2.3.1.3	Command string length		50.1.3
5.2.3.1.4	Multiple commands in one string		50.1.4
5.2.3.1.5	Response strings		50.2
5.2.3.2	ASCII string formal syntax	4.3.2	50.4
5.2.3.3	ASCII parameter value formats	4.3.3	50.3
5.2.3.3.1	Integers		50.3.1
5.2.3.3.2	Bit fields		50.3.2
5.2.3.3.3	ASCII text		50.3.3
5.2.3.4	Device driver buffer behavior	4.3.4	50.5
5.2.3.5	Device driver function and mode considerations	4.3.5	50.6
5.2.3.5.1	Device driver specific functions		50.6.1
5.2.3.5.2	Device driver function implementation		50.6.1
5.2.3.5.2.1	MS-DOS version 2.0 and higher		50.6.1.1
5.2.3.5.2.2	MS-DOS versions 3.0 and 3.1		50.6.1.2
5.2.3.5.2.3	MS-DOS version 3.2 and higher		50.6.1.3
5.2.3.5.3	Cooked and raw I/O modes		
5.2.3.5.4	Driver compliance		50.6.2
5.2.3.5.5	Interrupt 21 H file functions		
5.2.4	Binary interface issues	4.4	60.
5.2.4.1	Setting the software interrupt	4.4.1	60.2
5.2.4.2	Binary parameter value formats	4.4.2	60.6
5.2.4.2.1	Integers		60.6.1
5.2.4.2.2	Bit fields		60.6.2

TABLE J-1. Cross-reference of Sections 4 and 5 - Continued.

HANDBOOK PARAGRAPH NUMBER	PARAGRAPH TITLE	IMA RELEASE R1.1 SECTION NUMBER	MIL-STD-1379 PARAGRAPH NUMBER
5.2.4.2.3	Pointers	4.4.2	60.6.3
5.2.4.2.4	Strings		60.6.4
5.2.4.2.5	Color arrays		60.6.5
5.2.4.2.5.1	Array parameter		60.6.5.1 and 60.6.5.2
5.2.4.2.5.2	Length and color parameters		60.6.5.3 and 60.6.5.4
5.3	Command and parameter summaries	5	N/A
5.3.1	Command names and token numbers	5.1	60.7
5.3.1.1	Service group prefix values		
5.3.1.2	Command word values		
5.3.1.3	Combined service group prefix and command name values		
5.3.2	Parameter names and token numbers	5.2	60.8
5.3.3	System (sy) Commands	Section 6	Section 70
5.3.4	Visual Management (vm) commands	Section 7	Section 80
5.3.5	Videodisc (vd) commands	Section 8	Section 90
5.3.6	XY-Input (xy) commands	Section 9	Section 100
5.3.7	Digital audio (da) commands	4.3.3 and 4.4.2	N/A
5.3.8	Audio management (am) commands	4.3.3 and 4.4.2	N/A
5.4	Graphics default positions	Appendix A	80.3
5.5	Error handling	Appendix D	Section 110
5.5.1	General information	D.1	110.1
5.5.1.1	Error codes		110.1.1
5.5.1.2	Error messages		110.1.2
5.5.1.3	Error recovery		110.1.2
5.5.2	Error Listings	D.2	110.1.3
5.6	Application programming examples	Appendix C	N/A

APPENDIX J

TABLE J-2. Cross-reference of Appendix A, SYSTEM (sy) COMMANDS.

APPENDIX A PARAGRAPH NUMBER	PARAGRAPH TITLE	IMA RELEASE R1.1 SECTION/PAGE NUMBER	MIL-STD-1379 PARAGRAPH NUMBER
10.1	Scope	6.	70.1
30.	syCheckError COMMAND	Page 6-2	70.3
30.1	syCheckError command summary	6-3	
30.2	Description		N/A
30.3	Implementation		70.3.1
30.3.1	ASCII interface	6-4	70.3.1.1
30.3.2	Binary interface		70.3.1.2
30.4	Command parameters		70.3.2
30.4.1	Command parameter		70.3.2.1
30.4.2	Device parameter		70.3.2.2
30.4.3	Errno parameter		70.3.2.3
30.5	Implementation notes	6-5	N/A
30.6	Return values		70.3.3
30.6.1	ASCII returns		70.3.3.1
30.6.2	Binary returns		70.3.3.2
30.7	Related commands		N/A
30.8	Examples	6-6	
30.8.1	ASCII		
30.8.2	Binary		
40.	syErrorMsg COMMAND	6-8	70.4
40.1	syErrorMsg command summary		
40.2	Description		N/A
40.3	Command parameters		70.4.1
40.3.1	Errno parameter		70.4.1.1
40.3.2	Pmsg parameter		70.4.1.2
40.4	Implementation notes	6-9	N/A

TABLE J-2. Cross-reference of Appendix A, SYSTEM (sy) COMMANDS - Continued.

APPENDIX A PARAGRAPH NUMBER	PARAGRAPH TITLE	IMA RELEASE R1.1 SECTION/PAGE NUMBER	MIL-STD-1379 PARAGRAPH NUMBER
40.5	Return values	6-9	70.4.2
40.5.1	ASCII returns		70.4.2.1
40.5.2	Binary returns		70.4.2.2
40.6	Related commands		N/A
40.7	Examples		
40.7.1	ASCII		
40.7.2	Binary		
50.	syGetState COMMAND	6-11	70.5
50.1	syGetState command summary	6-12	
50.2	Command parameters		70.5.1
50.2.1	Ivver parameter		70.5.1.1
50.2.2	Mfgname and mfgver parameters		70.5.1.2
50.2.3	Support parameter	6-13	70.5.1.3
50.2.4	Parameters resulting in errors		N/A
50.3	Implementation notes		
50.4	Return values	6-14	70.5.2
50.4.1	ASCII returns		70.5.2.1
50.4.2	Binary returns		70.5.2.2
50.5	Related commands		N/A
50.6	Examples		
50.6.1	ASCII		
50.6.2	Binary		
60.	sylnit COMMAND	6-16	70.6
60.1	sylnit command summary		
60.2	Description		N/A
60.3	Command parameters		70.6.1

TABLE J-2. Cross-reference of Appendix A, SYSTEM (sy) COMMANDS - Continued.

APPENDIX A PARAGRAPH NUMBER	PARAGRAPH TITLE	IMA RELEASE R1.1 SECTION/PAGE NUMBER	MIL-STD-1379 PARAGRAPH NUMBER
60.4	Implementation notes	6-16	N/A
60.5	Return values	6-17	70.6.2
60.5.1	ASCII returns		
60.5.2	Binary returns		
60.6	Related commands		N/A
60.7	Examples		
60.7.1	ASCII		
60.7.2	Binary		
70.	syQueue COMMAND	6-18	70.7
70.1	syQueue command summary		
70.2	Description		N/A
70.3	Command parameters	6-19	70.7.1
70.3.1	Clear parameter		70.7.1.1
70.3.2	Execute parameter		70.7.1.2
70.3.3	State parameter		70.7.1.3
70.3.4	Combining parameters		70.7.1.4
70.4	Unqueueable commands	6-20	70.7.2
70.5	Queued commands causing errors		
70.6	Implementation notes		N/A
70.7	Return values	6-21	70.7.3
70.7.1	ASCII returns		
70.7.2	Binary returns		
70.8	Related commands		N/A
70.9	Examples		
70.9.1	ASCII		
70.9.2	Binary	6-22	

TABLE J-2. Cross-reference of Appendix A, SYSTEM (sy) COMMANDS - Continued.

APPENDIX A PARAGRAPH NUMBER	PARAGRAPH TITLE	IMA RELEASE R1.1 SECTION/PAGE NUMBER	MIL-STD-1379 PARAGRAPH NUMBER
80.	syStop COMMAND	6-23	70.8
80.1	syStop command summary		
80.2	Description		N/A
80.3	Command parameters		70.8.1
80.4	Implementation notes		N/A
80.5	Return values		70.8.2
80.5.1	ASCII returns		
80.5.2	Binary returns		
80.6	Related commands		N/A
80.7	Examples	6-24	
80.7.1	ASCII		
80.7.2	Binary		

TABLE J-3. Cross-reference of Appendix B, VISUAL-MANAGEMENT (vm) COMMANDS.

APPENDIX B PARAGRAPH NUMBERS	PARAGRAPH TITLES	IMA RELEASE R1.1 SECTION/PAGE NUMBERS	MIL-STD-1379 PARAGRAPH NUMBERS
10.1	Scope	7.	80.1
30.	GENERAL GUIDANCE	7.	80.3
30.1	Terms of reference	7.1	N/A
30.2	General information and assumptions	7.2	80.3
30.2.1	Overlayable graphics modes	7.2.1	80.3.1
30.2.2	Mode trapping	7.2.2	80.3.2
30.2.3	Genlock control	7.2.3	80.3.3
30.2.4	Graphics registration to the background video	7.2.4	80.3.4
30.2.5	VGA graphics versus CGA and EGA graphics	7.2.5	80.3.5
30.2.6	Logical versus physical colors	7.2.6	
30.3	Rounding methods for fades and dissolves	7.3	80.3.6
30.3.1	Fade and dissolve levels		
30.3.2	Level value rounding		
30.4	Palette issues	10.[1]	N/A
30.4.1	Initial commands and palette settings	10.1[1]	
30.4.1.1	Relevant commands	10.1.1[1]	
30.4.1.1.1	sylnit		
30.4.1.1.2	vminit		
30.4.1.1.3	vmSetGraphics		
30.4.1.2	Initialization	10.1.2[1]	
30.4.1.3	Reinitialization		
30.4.2	Effects of vmSetPalette on true CGAs and EGAs	10.2[1] 10.2.1[1]	

[1] This equivalent IMA Recommended Practice is defined in the IMA Interactive Video SIG, "Clarification for the Recommended Practices for Multimedia Portability (MS-DOS based systems) Release R 1.1", Revision number: CLAR 2. The number shown in the table identifies the applicable paragraph number in the CLAR 2 document.

TABLE J-3. Cross-reference of Appendix B, VISUAL-MANAGEMENT (vm) COMMANDS.

APPENDIX B PARAGRAPH NUMBERS	PARAGRAPH TITLES	IMA RELEASE R1.1 SECTION/PAGE NUMBERS	MIL-STD-1379 PARAGRAPH NUMBERS
30.4.3	Return values from vmGetPalette on true CGAs and EGAs	10.2[1] 10.2.2[1]	N/A
30.4.4	Mode 4 support on true CGAs	10.3[1]	
30.4.5	Large palette support in 16-color, 200-line EGA modes	10.4[1]	
30.4.6	Suggested default palettes	10.5[1]	
30.4.6.1	CGA mode 4	10.5.1[1]	
30.4.6.2	CGA mode 6	10.5.2[1]	
30.4.6.3	EGA 16-and 64-color modes	10.5.3[1]	
30.4.6.4	VGA 256-color modes	10.5.4[1]	
40.	vmFade COMMAND	Page 7-6	80.4
40.1	vmFade command summary	7-7	
40.2	Description		N/A
40.3	Command parameters		80.4.1
40.3.1	Dlevel parameter		80.4.1.1
40.3.2	Glevel parameter	7-8	80.4.1.2
40.3.3	Vlevel parameter		80.4.1.3
40.3.4	Time parameter		80.4.1.4
40.3.5	Wait parameter		80.4.1.5
40.4	Implementation notes		N/A
40.5	Return values	7-9	80.4.2
40.5.1	ASCII returns		80.4.2.1
40.5.2	Binary returns		80.4.2.2
40.6	Related commands	7-9	N/A

[1] This equivalent IMA Recommended Practice is defined in the IMA Interactive Video SIG, "Clarification for the Recommended Practices for Multimedia Portability (MS-DOS based systems) Release R 1.1", Revision number: CLAR 2. The number shown in the table identifies the applicable paragraph number in the CLAR 2 document.

TABLE J-3. Cross-reference of Appendix B, VISUAL-MANAGEMENT (vm) COMMANDS - Continued.

APPENDIX B PARAGRAPH NUMBERS	PARAGRAPH TITLES	IMA RELEASE R1.1 SECTION/PAGE NUMBERS	MIL-STD-1379 PARAGRAPH NUMBERS
40.7	Examples	7-9	N/A
40.7.1	ASCII		
40.7.2	Binary	7-10	
50.	vmGetPalette COMMAND	7-11	80.5
50.1	vmGetPalette command summary	7-12	
50.2	Description		N/A
50.3	Command parameters		80.5.1
50.3.1	Color + r, g, and b parameters		80.5.1.1
50.3.2	Color + length and array parameters	7-13	80.5.1.2
50.3.3	Parameters resulting in errors		
50.4	Implementation notes		N/A
50.5	Return values	7-14	80.5.2
50.5.1	ASCII returns		80.5.2.1
50.5.2	Binary returns		80.5.2.2
50.6	Related commands		N/A
50.7	Examples		
50.7.1	ASCII		
50.7.2	Binary	7-15	
60.	vmGetState COMMAND	7-16	80.6
60.1	vmGetState command summary	7-18	
60.2	Command parameters		80.6.1
60.2.1	Color parameter		80.6.1.1
60.2.2	Enable parameter		80.6.1.2
60.2.3	Defsource parameter		80.6.1.3
60.2.4	Dlevel, glevel, and vlevel parameters		80.6.1.4
60.2.5	Emulation parameter	7-18	80.6.1.5

TABLE J-3. Cross-reference of Appendix B, VISUAL-MANAGEMENT (vm) COMMANDS - Continued.

APPENDIX B PARAGRAPH NUMBERS	PARAGRAPH TITLES	IMA RELEASE R1.1 SECTION/PAGE NUMBERS	MIL-STD-1379 PARAGRAPH NUMBERS
60.2.6	Gmode parameter	7-19	80.6.1.6
60.2.7	Horzpix and vertpix parameters		80.6.1.7
60.2.8	Logcolors and physcolors parameters		80.6.1.8
60.2.9	Transcolors parameter		80.6.1.9
60.2.10	Tsources parameter		80.6.1.10
60.2.11	Vmode parameter		80.6.1.11
60.2.12	Width parameter		80.6.1.12
60.2.13	Xoffset and yoffset parameters	7-20	80.6.1.13
60.2.14	Parameters resulting in errors		N/A
60.3	Implementation notes		
60.4	Return values		80.6.2
60.4.1	ASCII returns		80.6.2.1
60.4.2	Binary returns		80.6.2.2
60.5	Related commands		N/A
60.6	Examples		
60.6.1	ASCII		
60.6.2	Binary	7-21	
70.	vminit COMMAND	7-22	80.7
70.1	vminit command summary		
70.2	Command parameters		80.7.1
70.3	Conditions set by vminit		80.7.2
70.4	Implementation notes	7-23	N/A
70.5	Return values		80.7.3
70.5.1	ASCII returns		
70.5.2	Binary returns		
70.6	Related commands	7-23	N/A

TABLE J-3. Cross-reference of Appendix B, VISUAL-MANAGEMENT (vm) COMMANDS - Continued.

APPENDIX B PARAGRAPH NUMBERS	PARAGRAPH TITLES	IMA RELEASE R1.1 SECTION/PAGE NUMBERS	MIL-STD-1379 PARAGRAPH NUMBERS
70.7	Examples	7-23	N/A
70.7.1	ASCII		
70.7.2	Binary		
80.	vmSetGraphics COMMAND	7-24	80.8
80.1	vmSetGraphics command summary		
80.2	Command parameters	7-25	80.8.1
80.2.1	Emulation parameter		80.8.1.1
80.2.2	Gmode parameter		80.8.1.2
80.2.3	Width parameter		80.8.1.3
80.2.4	Xoffset and yoffset parameters		80.8.1.4
80.3	Implementation notes	7-26	N/A
80.4	Return values		80.8.2
80.4.1	ASCII returns		
80.4.2	Binary returns		
80.5	Related commands		N/A
80.6	Examples		
80.6.1	ASCII		
80.6.2	Binary	7-27	
90.	vmSetPalette COMMAND	7-28	80.9
90.1	vmSetPalette command summary	7-29	
90.2	Description		N/A
90.3	Command parameters		80.9.1
90.3.1	Color + r, g, and b parameters		80.9.1.1
90.3.2	Color + length and array parameters	7-30	80.9.1.2
90.4	Implementation notes		N/A
90.5	Return values	7-31	80.9.2

TABLE J-3. Cross-reference of Appendix B, VISUAL-MANAGEMENT (vm) COMMANDS - Continued.

APPENDIX B PARAGRAPH NUMBERS	PARAGRAPH TITLES	IMA RELEASE R1.1 SECTION/PAGE NUMBERS	MIL-STD-1379 PARAGRAPH NUMBERS
90.5.1	ASCII returns	7-31	80.9.2
90.5.2	Binary returns		
90.6	Related commands		N/A
90.7	Examples		
90.7.1	ASCII		
90.7.2	Binary		
100.	vmSetTrans COMMAND	7-33	80.10
100.1	vmSetTrans command summary		
100.2	Command parameters		80.10.1
100.2.1	Clear parameter		80.10.1.1
100.2.2	Color and state parameters	7-34	80.10.1.2
100.2.3	Enable parameter		80.10.1.3
100.3	Implementation notes		N/A
100.4	Return values	7-35	80.10.2
100.4.1	ASCII returns		
100.4.2	Binary returns		
100.5	Related commands		N/A
100.6	Examples		
100.6.1	ASCII		
100.6.2	Binary		
110.	vmSetVideo COMMAND	7-37	80.11
110.1	vmSetVideo command summary		
110.2	Command parameters		80.11.1
110.2.1	Defsource parameter		80.11.1.1
110.2.2	Vmode parameter		80.11.1.2
110.3	Implementation notes	7-38	N/A

TABLE J-3. Cross-reference of Appendix B, VISUAL-MANAGEMENT (vm) COMMANDS - Continued.

APPENDIX B PARAGRAPH NUMBERS	PARAGRAPH TITLES	IMA RELEASE R1.1 SECTION/PAGE NUMBERS	MIL-STD-1379 PARAGRAPH NUMBERS
110.4	Return values	7-38	80.11.2
110.4.1	ASCII returns		
110.4.2	Binary returns		
110.5	Related commands		N/A
110.6	Examples		
110.6.1	ASCII		
110.6.2	Binary		

TABLE J-4. Cross-reference of Appendix C, VIDEODISC (vd) COMMANDS.

APPENDIX C PARAGRAPH NUMBERS	PARAGRAPH TITLES	IMA RELEASE R1.1 SECTION/PAGE NUMBERS	MIL-STD-1379 PARAGRAPH NUMBERS
10.1	Scope	8.	90.1
30.1	General information and assumptions	8.1	90.3
30.1.1	CAV and CLV videodisc support	8.1.1	90.3.1
30.1.2	Play and scan speeds	8.1.2	
30.1.3	Searches and instant jumps	8.1.3	90.3.2
30.1.4	Fields, frames, and chapters	8.1.4	90.3.3
30.2	Rounding methods for player speeds	8.2	90.3.4
30.3	Multisided and multidisc applications	8.3	90.5
30.3.1	Reference frame display		
30.3.2	Picture stops		
30.3.3	Chapter number search		
40.	vdGetState COMMAND	Page 8-7	90.4
40.1	vdGetState command summary	8-9	
40.2	Command parameters		90.4.1
40.2.1	Audio1 and audio2 parameters		90.4.1.1
40.2.2	Cdisplay parameter		90.4.1.2
40.2.3	Chapter parameter		90.4.1.3
40.2.4	Defdevice parameter		90.4.1.4
40.2.5	Device parameter		90.4.1.5
40.2.6	Disctype parameter		90.4.1.6
40.2.7	Door parameter	8-10	90.4.1.7
40.2.8	Frame parameter		90.4.1.8
40.2.9	Idxdisplay parameter		90.4.1.9
40.2.10	Motion parameter		90.4.1.10
40.2.11	Remote parameter		90.4.1.11
40.2.12	Speed parameter		90.4.1.12

TABLE J-4. Cross-reference of Appendix C, VIDEODISC (vd) COMMANDS - Continued.

APPENDIX C PARAGRAPH NUMBERS	PARAGRAPH TITLES	IMA RELEASE R1.1 SECTION/PAGE NUMBERS	MIL-STD-1379 PARAGRAPH NUMBERS
40.2.13	Spin parameter	8-10	90.4.1.13
40.2.14	Tdevices parameter		90.4.1.14
40.2.15	Video parameter	8-11	90.4.1.15
40.2.16	Parameters resulting in errors		N/A
40.3	Implementation notes		
40.4	Return values		90.4.2
40.4.1	ASCII returns		90.4.2.1
40.4.2	Binary returns		90.4.2.2
40.5	Related commands		N/A
40.6	Examples		
40.6.1	ASCII		
40.6.2	Binary	8-12	
50.	vdInit COMMAND	8-13	90.5
50.1	vdInit command summary		
50.2	Command parameters	8-14	90.5.1
50.2.1	Device parameter		90.5.1.1
50.2.2	Conditions set by vdInit	8-15	90.5.2
50.3	Implementation notes		N/A
50.4	Return values	8-16	90.5.3
50.4.1	ASCII returns		
50.4.2	Binary returns		
50.5	Related commands		N/A
50.6	Examples		
50.6.1	ASCII		
50.6.2	Binary		
60.	vdPassThru COMMAND	8-17	90.7

TABLE J-4. Cross-reference of Appendix C, VIDEODISC (vd) COMMANDS - Continued.

APPENDIX C PARAGRAPH NUMBERS	PARAGRAPH TITLES	IMA RELEASE R1.1 SECTION/PAGE NUMBERS	MIL-STD-1379 PARAGRAPH NUMBERS
60.1.	vdPassThru command summary	8-18	90.7
60.2	Command parameters		90.7.1
60.2.1	Device parameter		90.7.1.1
60.2.2	Pmsg parameter		90.7.1.2
60.2.3	Time parameter	8-19	90.7.1.3
60.3	Return values		90.7.2
60.3.1	ASCII returns		90.7.2.1
60.3.2	Binary returns	8-20	90.7.2.2
60.4	Examples		N/A
60.4.1	ASCII		
60.4.2	Binary		
70.0	vdPlay COMMAND	8-21	90.8
70.1	vdPlay command summary	8-22	
70.2	Command parameters		90.8.1
70.2.1	No parameters		
70.2.2	Chapter parameter		90.8.1.1
70.2.2.1	Compatible parameters		
70.2.3	Device parameter		90.8.1.2
70.2.3.1	Compatible parameters		
70.2.4	Direction parameter	8-23	90.8.1.3
70.2.4.1	Compatible parameters		
70.2.5	From parameter		90.8.1.4
70.2.5.1	Compatible parameters		
70.2.6	Speed parameter		90.8.1.5
70.2.6.1	Compatible parameters	8-24	
70.2.7	To parameter		90.8.1.6

TABLE J-4. Cross-reference of Appendix C, VIDEODISC (vd) COMMANDS - Continued.

APPENDIX C PARAGRAPH NUMBERS	PARAGRAPH TITLES	IMA RELEASE R1.1 SECTION/PAGE NUMBERS	MIL-STD-1379 PARAGRAPH NUMBERS
70.2.7.1	Compatible parameters	8-24	90.8.1.6
70.2.8	Wait parameter		90.8.1.7
70.2.8.1	Compatible parameters	8-25	
70.3	Implementation notes		N/A
70.4	Return values	8-26	90.8.2
70.4.1	ASCII returns		90.8.2.1
70.4.2	Binary returns		90.8.2.2
70.5	Related commands		N/A
70.6	Examples		
70.6.1	ASCII		
70.6.2	Binary	8-27	
80.	vdScan COMMAND	8-28	90.9
80.1	vdScan command summary		
80.2	Command parameters	8-29	90.9.1
80.2.1	vdScan with no parameters		
80.2.2	Device parameter		90.9.1.1
80.2.3	Direction parameter		90.9.1.2
80.2.4	Wait parameter		90.9.1.3
80.3	Implementation notes		N/A
80.4	Return values		90.9.2
80.4.1	ASCII returns		
80.4.2	Binary returns		
80.5	Related commands		N/A
80.6	Examples	8-30	
80.6.1	ASCII		
80.6.2	Binary		

TABLE J-4. Cross-reference of Appendix C, VIDEODISC (vd) COMMANDS - Continued.

APPENDIX C PARAGRAPH NUMBERS	PARAGRAPH TITLES	IMA RELEASE R1.1 SECTION/PAGE NUMBERS	MIL-STD-1379 PARAGRAPH NUMBERS
90.	vdSearch COMMAND	8-31	90.10
90.1	vdSearch command summary		
90.2	Command parameters	8-32	90.10.1
90.2.1	Chapter parameter		90.10.1.1
90.2.2	Device parameter		90.10.1.2
90.2.3	Frame parameter		90.10.1.3
90.2.4	Wait parameter		90.10.1.4
90.3	Return values	8-33	90.10.2
90.3.1	ASCII returns		
90.3.2	Binary returns		
90.4	Related commands		N/A
90.5	Examples		
90.5.1	ASCII		
90.5.2	Binary		
100.	vdSet COMMAND	8-34	90.11
100.1	vdSet command summary	8-35	
100.2	Command parameters		90.11.1
100.2.1	Audio1 and audio2 parameters		90.11.1.1
100.2.2	Cdisplay parameter		90.11.1.2
100.2.3	Defdevice parameter	8-36	90.11.1.3
100.2.4	Device parameter		90.11.1.4
100.2.5	Door parameter,		90.11.1.5
100.2.6	Idxdisplay parameter		90.11.1.6
100.2.7	Remote parameter	8-37	90.11.1.7
100.2.8	Spin parameter		90.11.1.8
100.2.9	Video parameter		90.11.1.9

TABLE J-4. Cross-reference of Appendix C, VIDEODISC (vd) COMMANDS - Continued.

APPENDIX C PARAGRAPH NUMBERS	PARAGRAPH TITLES	IMA RELEASE R1.1 SECTION/PAGE NUMBERS	MIL-STD-1379 PARAGRAPH NUMBERS
100.2.10	Wait parameter	8-37	90.11.1.10
100.3	Implementation notes		N/A
100.4	Return values		90.11.2
100.4.1	ASCII returns		
100.4.2	Binary returns		
100.5	Related commands	8-38	N/A
100.6	Examples		
100.6.1	ASCII		
100.6.2	Binary		
110.	vdStep COMMAND	8-40	90.12
110.1	vdStep command summary		
110.2	Command parameters		90.12.1
110.2.1	No parameters		
110.2.2	Device parameter		90.12.1.1
110.2.3	Direction parameter	8-41	90.12.1.2
110.3	Implementation notes		N/A
110.4	Return values		90.12.2
110.4.1	ASCII returns		
110.4.2	Binary returns		
110.5	Related commands		N/A
110.6	Examples		
110.6.1	ASCII		
110.6.2	Binary		
120.	vdStill COMMAND	8-42	90.12
120.1	vdStill command summary		
120.2	Command parameters		90.12.1

TABLE J-4. Cross-reference of Appendix C, VIDEODISC (vd) COMMANDS - Continued.

APPENDIX C PARAGRAPH NUMBERS	PARAGRAPH TITLES	IMA RELEASE R1.1 SECTION/PAGE NUMBERS	MIL-STD-1379 PARAGRAPH NUMBERS
120.2.1	Device parameter	8-42	90.12.1.1
120.3	Return values	8-43	90.12.2
120.3.1	ASCII returns		
120.3.2	Binary returns		
120.4	Related commands		N/A
120.5	Examples		
120.5.1	ASCII		
120.5.2	Binary		

TABLE J-5. Cross-reference of Appendix D, XY-INPUT (xy) COMMANDS.

APPENDIX D PARAGRAPH NUMBERS	PARAGRAPH TITLES	IMA RELEASE R1.1 SECTION/PAGE NUMBERS	MIL-STD-1379 PARAGRAPH NUMBERS
10.1	Scope	9.	100.1
30.1	General information and assumptions	9.1	N/A
30.1.1	Device mapping	9.1.1	100.3.1
30.1.2	Handling the graphics plane and cursor	9.1.2	100.3.2
30.1.3	Coordinate space mapping	9.1.3	100.3.3
30.1.3.1	Clipping values		100.3.3.2
30.1.3.2	Coordinate space calibration		100.3.3.3
30.1.4	Buttons	9.1.4	100.3.4
30.2	Stream-mode and point-mode devices	9.2	100.3.5
40.	xyGetInput COMMAND	Page 9-5	100.4
40.1	xyGetInput command summary		
40.2	Command parameters	9-6	100.4.1
40.2.1	Buttons parameter		100.4.1.1
40.2.2	Device parameter		100.4.1.2
40.2.3	Xpos and ypos parameters		100.4.1.3
40.3	Implementation notes		N/A
40.4	Return values	9-7	100.4.2
40.4.1	ASCII returns		100.4.2.1
40.4.2	Binary returns		100.4.2.2
40.5	Related commands		N/A
40.6	Examples		
40.6.1	ASCII		
40.6.2	Binary	9-8	
50.	xyGetState COMMAND	9-9	100.5
50.1	xyGetState command summary	9-10	
50.2	Command parameters	9-11	100.5.1

TABLE J-5. Cross-reference of Appendix D, XY-INPUT (xy) COMMANDS - Continued.

APPENDIX D PARAGRAPH NUMBERS	PARAGRAPH TITLES	IMA RELEASE R1.1 SECTION/PAGE NUMBERS	MIL-STD-1379 PARAGRAPH NUMBERS
50.2.1	Cursor parameter	9-11	100.5.1.1
50.2.2	Defdevice parameter		100.5.1.2
50.2.3	Device parameter		100.5.1.3
50.2.4	Tbuttons parameter		100.5.1.4
50.2.5	Tdevices parameter		100.5.1.5
50.2.6	Xmin, ymin, xmax, and ymax parameters		100.5.1.6
50.2.7	Xminclip, yminclip, xmaxclip, and yminclip parameters	9-12	100.5.1.7
50.2.8	Parameters resulting in errors		100.5.1
50.3	Implementation notes		N/A
50.4	Return values		100.5.2
50.4.1	ASCII returns		100.5.2.1
50.4.2	Binary returns		100.5.2.2
50.5	Related commands		N/A
50.6	Examples		
50.6.1	ASCII		
50.6.2	Binary	9-13	
60.	xyinit COMMAND	9-14	100.6
60.1	xyinit command summary		
60.2	Command parameters		100.6.1
60.2.1	Device parameter		100.6.1.1
60.2.2	Conditions set by xyinit	9-15	100.6.1.2
60.2.3	Effects of xyinit on the cursor parameter		
60.3	Implementation notes	9-15	N/A
60.4	Return values	9-16	100.6.2
60.4.1	ASCII returns		
60.4.2	Binary returns		

TABLE J-5. Cross-reference of Appendix D, XY-INPUT (xy) COMMANDS - Continued.

APPENDIX D PARAGRAPH NUMBERS	PARAGRAPH TITLES	IMA RELEASE R1.1 SECTION/PAGE NUMBERS	MIL-STD-1379 PARAGRAPH NUMBERS
60.5	Related commands	9-16	N/A
60.6	Examples		
60.6.1	ASCII		
60.6.2	Binary		
70.	xySet COMMAND	9-17	100.7
70.1	xySet command Summary	9-18	
70.2	Command parameters	9-19	100.7.1
70.2.1	Cursor parameter		100.7.1.1
70.2.2	Defdevice parameter		100.7.1.2
70.2.3	Device parameter		100.7.1.3
70.2.4	Xmin, ymin, xmax, and ymax parameters		100.7.1.4
70.2.5	Xminclip, yminclip, xmaxclip, and ymaxclip parameters	9-20	100.7.1.5
70.2.6	Xpos and ypos parameters		100.7.1.6
70.3	Return values		100.7.2
70.3.1	ASCII returns		
70.3.2	Binary returns		
70.4	Related commands		N/A
70.5	Examples	9-21	
70.5.1	ASCII		
70.5.2	Binary examples		

TABLE J-6. Cross-reference of Appendix E, DIGITAL AUDIO (da) COMMANDS.

APPENDIX E PARAGRAPH NUMBERS	PARAGRAPH TITLE	IMA RELEASE R1.1 PARAGRAPH NUMBER	MIL-STD-1379 PARAGRAPH NUMBER
10.	SCOPE	4.3.3 and 4.4.2	Not Applicable[1]
10.1	Scope		
10.2	Application guidance		

[1] The digital audio service group commands and parameters have not been defined. The service group and Appendix E are identified to support future ICW portability requirements.

TABLE J-7. Cross-reference of Appendix F, AUDIO MANAGEMENT (am) COMMANDS.

APPENDIX F PARAGRAPH NUMBERS	PARAGRAPH TITLES	IMA RELEASE R1.1 PARAGRAPH NUMBERS	MIL-STD-1379 PARAGRAPH NUMBERS
10.	SCOPE	4.3.3 and 4.4.2	Not Applicable[1]
10.1	Scope		
10.2	Application guidance		

[1] The audio management service group commands and parameters have not been defined. The service group and Appendix F are identified to support future ICW portability requirements.

TABLE J-8. Cross-reference of Appendix G, DEFAULT POSITIONS OF ICW GRAPHICS.

APPENDIX G PARAGRAPH NUMBER	PARAGRAPH TITLE	IMA RELEASE R1.1 PARAGRAPH NUMBER
10.1	Scope	A.
30.	GENERAL GUIDANCE	Appendix A
30.1	Introduction	A.
30.1.1	General application	
30.1.2	Graphics registration	
30.1.3	Graphics display	A.1
30.1.4	Graphics values	
30.2	Special considerations for VGA graphics	A.2
30.2.1	Differences in signals and timing	A.2.1
30.2.2	Differences in the size of active graphics	A.2.2
40.	HORIZONTAL POSITIONS	A.3
40.1	Horizontal positions	A.3
40.1.1	General assumptions	A.3.1
40.2	NTSC Video	A.3.2
40.2.1	Position of true CGA and EGA graphics	
40.2.2	Position of VGA graphics emulating CGA and EGA modes	
40.3	PAL Video	A.3.3
40.3.1	Position of true CGA and EGA graphics	
40.3.2	Position of VGA graphics emulating CGA and EGA modes	
50.	VERTICAL POSITIONS	A.4
50.1	Vertical positions	A.4
50.1.1	General assumptions	A.4.1
50.2	NTSC Video	A.4.2
50.2.1	Position of graphics in lines relative to vertical sync	
50.2.1.1	Starting and ending positions for 200-line graphics	

TABLE J-8. Cross-reference of Appendix G, DEFAULT POSITIONS OF ICW GRAPHICS - Continued.

APPENDIX G PARAGRAPH NUMBER	PARAGRAPH TITLE	IMA RELEASE R1.1 PARAGRAPH NUMBER
50.2.1.2	Starting and ending positions for 240-line (640 X 480) graphics	A.4.2
50.2.2	Position of graphics as a proportion of total video	A.4.2
50.2.2.1	Starting and ending positions for 200-line graphics	
50.2.2.2	Starting and ending positions for 240-line (640 X 480) graphics	
50.2.3	Position of graphics as a proportion of active video	
50.2.3.1	Starting and ending positions for 200-line graphics	
50.2.3.2	Starting and ending positions for 240-line (640 X 480) graphics	
50.3	PAL Video	A.4.3
50.3.1	Position of graphics in lines relative to vertical sync	
50.3.1.1	Starting and ending positions for 200-line graphics	
50.3.1.2	Starting and ending positions for 240-line graphics	
50.3.2	Position of graphics as a proportion of total video	
50.3.2.1	Starting and ending positions for 200-line graphics	
50.3.2.2	Starting and ending positions for 240-line graphics	
50.3.3	Position of graphics as a proportion of active video	
50.3.3.1	Starting and ending positions for 200-line graphics	
50.3.3.2	Starting and ending positions for 240-line graphics	

TABLE J-9. Cross-reference of Appendix H, ICW PORTABILITY PRACTICES ERROR HANDLING.

APPENDIX H PARAGRAPH NUMBER	PARAGRAPH SUBJECT	IMA RELEASE R1.1 PARAGRAPH NUMBER	MIL-STD-1379 PARAGRAPH NUMBER
30.	ERROR HANDLING	Appendix D	Section 110
30.1	Introduction	D.	110.1
30.2	Error listings	D.2	110.1.3
30.3	Return message format		
40.	COMMAND PROBLEMS	D.2.1	110.3
40.1	Error 1 - SERVICE GROUP NOT INSTALLED	D.2.1	110.3.1
40.2	Error 2 - UNKNOWN COMMAND		110.3.2
40.3	Error 3 - SYSTEM NOT INITIALIZED		110.3.3
40.4	Error 15 - GENERAL COMMAND ERROR		110.3.4
50	ASCII INTERFACE PROBLEMS	D.2.2	110.4
50.1	Error 16 - BAD COMMAND SYNTAX		110.4.1
50.2	Error 17 - COMMAND TOO LONG		110.4.2
50.3	Error 18 - RESPONSE TOO LONG		110.4.3
50.4	Error 19 - DEVICE DRIVER READ BEFORE WRITE		110.4.4
50.5	Error 31 - GENERAL ASCII INTERFACE ERROR		110.4.5
60.	BINARY INTERFACE PROBLEMS	D.2.3	110.5
60.1	Error 32 - INVALID PARAMETER COUNT		110.5.1
60.2	Error 33 - INVALID PARAMETER PACKET ADDRESS		110.5.2
60.3	Error 34 - INVALID POINTER IN PARAMETER PACKET		110.5.3
60.4	Error 47 - GENERAL BINARY INTERFACE ERROR		110.5.4
70	PARAMETER PROBLEMS	D.2.4	110.6
70.1	Error 48 - RESERVED		110.6.1
70.2	Error 49 - INSUFFICIENT PARAMETERS		110.6.2

TABLE J-9. Cross-reference of Appendix H, ICW PORTABILITY PRACTICES ERROR HANDLING.

APPENDIX H PARAGRAPH NUMBER	PARAGRAPH SUBJECT	IMA RELEASE R1.1 PARAGRAPH NUMBER	MIL-STD-1379 PARAGRAPH NUMBER
70.3	Error 50 - PARAMETERS CANNOT BE USED TOGETHER	D.2.4	110.6.3
70.4	Error 51 - PARAMETER VALUE INVALID OR OUT OF RANGE		110.6.4
70.5	Error 52 - PARAMETER INVALID FOR THIS COMMAND		110.6.5
70.6	Error 53 - MISSING PARAMETER VALUE		110.6.6
70.7	Error 54 - PARAMETER USED MORE THAN ONCE		110.6.7
70.8	Error 79 - GENERAL PARAMETER ERROR		110.6.8
80.	HARDWARE PROBLEMS	D.2.5	110.7
80.1	Error 80 - INITIALIZATION ERROR		110.7.1
80.2	Error 81 - DEVICE NOT INITIALIZED		110.7.2
80.3	Error 82 - COMMUNICATIONS TIMEOUT		110.7.3
80.4	Error 83 - COMMUNICATIONS ERROR		110.7.4
80.5	Error 84 - DEVICE REPORTS ERROR		110.7.5
80.6	Error 85 - DEVICE CANCELED REQUEST		110.7.6
80.7	Error 86 - DEVICE NOT READY		110.7.7
80.8	Error 87 - ACTION NOT SUPPORTED BY DEVICE		110.7.8
80.9	Error 88 - UNABLE TO RETURN REQUESTED INFORMATION		110.7.9
80.10	Error 111 - GENERAL HARDWARE ERROR		110.7.10
90.	SYSTEM RESOURCES	D.2.6	110.8
90.1	Error 112 - INSUFFICIENT MEMORY		110.8.1
90.2	Error 113 - NEEDED HARDWARE INTERRUPT IN USE		110.8.2
90.3	Error 114 - NEEDED SOFTWARE INTERRUPT IN USE		110.8.3

TABLE J-9. Cross-reference of Appendix H, ICW PORTABILITY PRACTICES ERROR HANDLING - Continued.

APPENDIX H PARAGRAPH NUMBER	PARAGRAPH SUBJECT	IMA RELEASE R1.1 PARAGRAPH NUMBER	MIL-STD-1379 PARAGRAPH NUMBER
90.4	Error 115 - NEEDED DMA CHANNEL NOT AVAILABLE	D.2.6	110.8.4
90.5	Error 116 - NEEDED TIMER NOT AVAILABLE		110.8.5
90.6	Error 127 - GENERAL RESOURCES ERROR		110.8.6
100.	FILING SYSTEM PROBLEMS	D.2.7	110.9
100.1	Error 128 - INVALID FILENAME		110.9.1
100.2	Error 129 - INVALID PATH		110.9.2
100.3	Error 130 - INVALID DRIVE		110.9.3
100.4	Error 131 - INVALID FILE NUMBER		110.9.4
100.5	Error 132 - CANNOT OPEN OR CREATE FILE		1410.9.5
100.6	Error 133 - CANNOT CLOSE FILE		110.9.6
100.7	Error 134 - FILE ALREADY OPEN		110.9.7
100.8	Error 135 - FILE ALREADY EXISTS		110.9.8
100.9	Error 136 - FILE DOES NOT EXIST		110.9.9
100.10	Error 137 - FILE ACCESS DENIED		110.9.10
100.11	Error 138 - FILE SEEK ERROR		110.9.11
100.12	Error 139 - TOO MANY OPEN FILES		110.9.12
100.13	Error 140 - DISK FULL		110.9.13
100.14	Error 141 - DISK READ ERROR		110.9.14
100.15	Error 142 - DISK WRITE ERROR		110.9.15
100.16	Error 159 - GENERAL FILING-SYSTEM ERROR		110.9.16
110.	MISCELLANEOUS PROBLEMS	D.2.8	110.10
110.1	Error 160 - INVALID DEVICE NUMBER		110.10.1
110.2	Error 161 - BUFFER OVERFLOW		110.10.2
110.3	Error 162 - INTERNAL CALCULATION ERROR		110.10.3
110.4	Error 163 - COPY PROTECTION ERROR		110.10.4

TABLE J-9. Cross-reference of Appendix H, ICW PORTABILITY PRACTICES ERROR HANDLING - Continued.

APPENDIX H PARAGRAPH NUMBER	PARAGRAPH SUBJECT	IMA RELEASE R1.1 PARAGRAPH NUMBER	MIL-STD-1379 PARAGRAPH NUMBER
110.5	Error 164 - INTERFACE BUSY	D.2.8	110.10.5
110.6	Error 165 - INVALID INTERRUPT NUMBER		110.10.6
110.7	Error 173 - GENERAL INTERNAL ERROR		110.10.7
110.8	Error 174 - GENERAL OPERATING SYSTEM ERROR		110.10.8
110.9	Error 175 - GENERAL ERROR		110.10.9
120.	SYSTEM GROUP PROBLEMS	D.2.9	110.11
120.1	Error 176 - QUEUE FULL		110.11.1
120.2	Error 177 - COMMAND CANNOT BE QUEUED		110.11.2
120.3	Error 191 - GENERAL SYSTEM ERROR		110.11.3
130.	VISUAL-MANAGEMENT PROBLEMS	D.2.10	110.12
130.1	Error 192 - SYNCHRONIZATION ERROR		110.12.1
130.2	Error 193 - GRAPHICS MODE PROBLEM		110.12.2
130.3	Error 194 - UNSUPPORTED GRAPHICS MODE		110.12.3
130.4	Error 207 - GENERAL VISUAL-MANAGEMENT ERROR		110.12.4
140.	VIDEODISC PROBLEMS	D.2.11	110.13
140.1	Error 208 - ACTION NOT SUPPORTED BY DISC		110.13.1
140.2	Error 209 - DISC NOT SPUN UP		110.13.2
140.3	Error 210 - DISC NOT SPUN DOWN		110.13.3
140.4	Error 211 - DOOR OPEN		110.13.4
140.5	Error 212 - NO DISC IN TRAY		110.13.5
140.6	Error 213 - BAD DISC SECTION		110.13.6
140.7	Error 214 - FELL OFF DISC		110.13.7
140.8	Error 215 - INVALID FRAME NUMBER		110.13.8
140.9	Error 216 - INVALID CHAPTER NUMBER		110.13.9
140.10	Error 217 - INVALID TIME CODE		110.13.10

TABLE J-9. Cross-reference of Appendix H, ICW PORTABILITY PRACTICES ERROR HANDLING - Continued.

APPENDIX H PARAGRAPH NUMBER	PARAGRAPH SUBJECT	IMA RELEASE R1.1 PARAGRAPH NUMBER	MIL-STD-1379 PARAGRAPH NUMBER
140.11	Error 239 - GENERAL VIDEODISC PLAYER ERROR	D.2.11	110.13.11
150.	XY-INPUT DEVICE PROBLEMS	D.2.12	110.14
150.1	Error 240 - DEVICE NOT CALIBRATED		110.14.1
150.2	Error 241 - INVALID COORDINATE		110.14.2
150.3	Error 242 - CURSOR PROBLEM		110.14.3
150.4	Error 255 - GENERAL XY-INPUT ERROR		110.14.4

TABLE J-10. Cross-reference of Appendix I, Application Programming Examples.

APPENDIX I PARAGRAPH NUMBER	PARAGRAPH SUBJECT	IMA RELEASE R1.1 PARAGRAPH NUMBER
10.1	Scope	C.
30.	USING THE ASCII INTERFACE	C.1
30.1	Using the ASCII interface	
30.1.1	BASIC example 1	
30.1.2	BASIC example 2	
40.	USING SOFTWARE INTERRUPT CALLS	C.2
40.1	Using software interrupt calls	
50.	LIBRARY CALLS WITH PARAMETER NUMBERS	C.3
50.1	Library calls with parameter numbers	
60.	ANALYZING BIT FIELDS	C.4
60.1	Analyzing bit fields	
70.	DETERMINING THE NUMBER OF LOGICAL DEVICES	C.5
70.1	Determining the number of logical devices	

APPENDIX J

TABLE J-11. Cross-reference of tables.

HANDBOOK TABLE NUMBER	TABLE TITLE	IMA RELEASE R1.1 TABLE/PAGE NUMBER	MIL-STD-1379 TABLE NUMBER
1.	IBM-compatible graphics modes	Table B-1	D-11
2.	Parameter block layout	Table 3-1	D-2
3.	Formal syntax for ASCII command and response strings	Table 4-1	D-1
4.	ASCII bit field values	Table 4-2	N/A
5.	Binary bit field values	Table 4-3	N/A
6.	Service group prefix values for the binary interface	Table 5-1	N/A
7.	Command word values for the binary interface	Table 5-2	N/A
8.	ASCII command name summary	Table 5-3	D-3
9.	A summary of parameter labels including binary token numbers	Table 5-4	D-4
A-1.	System (sy) Commands Summary	Table 6-1	D-3
A-2.	syCheckError parameters	Page 6-2	D-5
A-3.	syErrorMsg parameters	6-8	D-6
A-4.	syGetState parameters	6-11	D-7
A-5.	syGetState support return values	6-13	D-8
A-6.	syQueue parameters	6-18	D-9
A-7.	Unqueueable commands	6-20	D-10
B-1.	Visual management (vm) Commands Summary	Table 7-1	D-3
B-2.	Suggested default values for CGA mode 4	Table 1[1]	N/A
B-3.	Suggested default values for CGA mode 6	Table 2[1]	N/A
B-4.	Suggested default values for EGA 16- and 64-color modes	Table 3[1]	N/A
B-5.	Suggested default values for VGA 256-color modes	Table 4[1]	N/A
B-6.	vmFade parameters	Page 7-6	D-12
B-7.	vmGetPalette parameters	7-11	D-13

[1] This equivalent IMA Recommended Practice is defined in the IMA Interactive Video SIG, "Clarification for the Recommended Practices for Multimedia Portability (MS-DOS based systems) Release R 1.1", Revision number: CLAR 2. The number shown identifies the applicable table number in the CLAR 2 document.

TABLE J-11. Cross-reference of tables - Continued.

HANDBOOK TABLE NUMBER	TABLE TITLE	IMA RELEASE R1.1 TABLE/PAGE NUMBER	MIL-STD-1379 TABLE NUMBER
B-8.	vmGetState ASCII parameters	7-16	D-14
B-9.	vmGetState binary parameters	7-17	D-15
B-10	Parameter values set by vmInit	7-22	D-16
B-11.	vmSetGraphics parameters	7-24	D-17
B-12.	vmSetPalette parameters	7-28	D-18
B-13.	vmSetTrans parameters	Page 7-33	D-19
B-14.	vmSetVideo parameters	7-37	D-20
C-1.	Videodisc (vd) Commands Summary	Table 8-1	D-3
C-2.	Effects of rounding on speed parameters for Sony LDP-2000	Table 8-2	N/A
C-3.	Effects of rounding on speed parameters for Pioneer 4200	Table 8-3	N/A
C-4.	Example speed parameter values for boundary player speeds	Table 8-4	D-21
C-5.	vdGetState ASCII parameters	Page 8-7	D-22
C-6.	vdGetState binary parameters	8-8	D-23
C-7.	vdInit parameters	8-13	D-24
C-8.	Parameter values set by vdInit	8-15	D-25
C-9.	vdPassThru parameters	8-17	D-26
C-10.	vdPlay parameters	8-21	D-27
C-11.	Effects of the wait parameter on vdPlay	8-25	D-28
C-12.	vdScan parameters	8-28	D-29
C-13.	vdSearch parameters	8-31	D-30
C-14.	vdSet ASCII parameters	8-34	D-31
C-15.	vdSet binary parameters	8-35	
C-16.	vdStep parameters	8-40	D-32
C-17.	vdStill parameters	8-42	D-33
D-1.	XY-Input (xy) Commands Summary	Table 9-1	D-3

TABLE J-11. Cross-reference of tables - Continued.

HANDBOOK TABLE NUMBER	TABLE TITLE	IMA RELEASE R1.1 TABLE/PAGE NUMBER	MIL-STD-1379 TABLE NUMBER
D-2.	xyGetInput parameters	Page 9-5	D-34
D-3.	xyGetState ASCII parameters	9-9	D-35
D-4.	xyGetState binary parameters	9-10	D-36
D-5.	xyInit parameters	9-14	D-37
D-6.	Parameter values set by xyInit	9-15	D-38
D-7.	xySet ASCII parameters	Page 9-17	D-39
D-8.	xySet Binary parameters	9-18	D-40

TABLE J-12. Cross-reference of figures.

HANDBOOK FIGURE NUMBER	FIGURE TITLE	IMA RELEASE R1.1 FIGURE NUMBER	MIL-STD-1379 FIGURE NUMBER
1.	General architecture of a compliant system	2-1	D-1
B-1.	Simplified functional model of a video overlay subsystem	7-1	N/A
G-1.	Simplified diagram of an overlay display using CGA or EGA graphics	A-1	
G-2.	One horizontal line of NTSC video with 640- or 320-pixel overlay graphics	A-2	
G-3.	One horizontal line of PAL video with 640- and 320-pixel overlay graphics	A-3	
G-4.	NTSC vertical timing with 200-line overlay graphics	A-4	
G-5.	PAL vertical timing with 200-line overlay graphics	A-5	

This Page Intentionally Left Blank.

CONCLUDING MATERIAL

Custodians:
Army - AV
Navy - SH
Air Force - 11

Review activities:
Army - TM
Navy - AS, EC, MC, TD
Air Force - 13, 94

Preparing activity:
Navy - SH

Agent:
Navy - OS
(Project ILSS-0052-02)

STANDARDIZATION DOCUMENT IMPROVEMENT PROPOSAL

INSTRUCTIONS

1. The preparing activity must complete blocks 1, 2, 3, and 8. In block 1, both the document number and revision letter should be given.
2. *The submitter of this form must complete* blocks 4, 5, 6, and 7.
3. The preparing activity must provide a reply within 30 days from receipt of the form.

NOTE: This form may not be used to request copies of documents, nor to request waivers, or clarification of requirements on current contracts. Comments submitted on this form do not constitute or imply authorization to waive any portion of the referenced document(s) or to amend contractual requirements.

I RECOMMEND A CHANGE:	1. DOCUMENT NUMBER MIL-HDBK-284-2	2. DOCUMENT DATE (YYMMDD) 920722

3. DOCUMENT TITLE
Interactive Courseware (ICW) for Military Training, Portability Practices For (Part 2 of 3 Parts)

4. NATURE OF CHANGE *(Identify paragraph number and include proposed rewrite, if possible. Attach extra sheets as needed.)*

5. REASON FOR RECOMMENDATION

6. SUBMITTER

a. NAME (Last, First, Middle Initial)	b. ORGANIZATION
c. ADDRESS (Include Zip Code)	d. TELEPHONE (Include Area Code) (1) Commercial (2) AUTOVON (If applicable) — 7. DATE SUBMITTED (YYMMDD)

8. PREPARING ACTIVITY

a. NAME	b. TELEPHONE (Include Area Code) (1) Commercial	(2) AUTOVON
Commander, Naval Sea Systems Command (SEA 55Z3)	(703)602-0160	332-0160
c. ADDRESS (Include Zip Code) Department of the Navy Washington, DC 20362-5101	IF YOU DO NOT RECEIVE A REPLY WITHIN 45 DAYS, CONTACT: Defense Quality and Standardization Office 5203 Leesburg Pike, Suite 1403, Falls Church, VA 22041-3466 Telephone (703) 756-2340 AUTOVON 289-2340	

DD Form 1426, OCT 89 *Previous editions are obsolete.* 198/290

NOTICE OF CANCELLATION

MIL-HDBK-284/2
NOTICE 1
3 September 1999

MILITARY HANDBOOK

INTERACTIVE COURSEWARE (ICW) FOR MILITARY TRAINING, PORTABILITY PRACTICES FOR (PART 2 OF 3 PARTS)

MIL-HDBK-284/2, dated 22 July 1992, is hereby canceled. Guidance on the development of interactive multimedia instruction is contained in MIL-HDBK-1379/3, "Development of Interactive Multimedia Instruction (IMI) (Part 3 of 4 Parts)."

(Copies of MIL-HDBK-1379/3 are available from the Standardization Document Order Desk, 700 Robbins Avenue, Building 4D, Philadelphia, PA 19111-5094.)

Custodians:
Army-AV
Navy-SH
Air Force-11

Review activities:
Army-TM2
Navy-AS, EC, MC, TD
Air Force-13, 94

Preparing activity:
Navy-SH
(Project ALSS-0071)

AMSC N/A

AREA ALSS

DISTRIBUTION STATEMENT A. Approved for public release; distribution is unlimited.