IDA DOCUMENT D-855

RECOMMENDED PRACTICES FOR INTERACTIVE VIDEO PORTABILITY

Philip Dodds
Randall House Associates

Scott Lewis
DSL Associates

David McFarling
SBCS Inc.

Henry Mistrot
Video Associates Labs

Geoff Snowman
Randall House Associates

Jack Spiegelberg
IVID Communications

October 1990

Prepared for
Office of the Assistant Secretary of Defense
(Force Management and Personnel)
and
Office of the Assistant Secretary of Defense for Public Affairs
(American Forces Information Service)

Approved for public release; distribution unlimited.

INSTITUTE FOR DEFENSE ANALYSES
1801 N. Beauregard Street. Alexandria, Virginia 22311-1772
DEFINITIONS
IDA publishes the following documents to report the results of its work.

Reports
Reports are the most authoritative and most carefully considered products IDA publishes. They normally embody results of major projects which (a) have a direct bearing on decisions affecting major programs, (b) address issues of significant concern to the Executive Branch, the Congress and/or the public, or (c) address issues that have significant economic implications. IDA Reports are reviewed by outside panels of experts to ensure their high quality and relevance to the problems studied, and they are released by the President of IDA.

Group Reports
Group Reports record the findings and results of IDA established working groups and panels composed of senior individuals addressing major issues which otherwise would be the subject of an IDA Report. IDA Group Reports are reviewed by the senior individuals responsible for the project and others as selected by IDA to ensure their high quality and relevance to the problems studied, and are released by the President of IDA.

Papers
Papers, also authoritative and carefully considered products of IDA, address studies that are narrower in scope than those covered in Reports. IDA Papers are reviewed to ensure that they meet the high standards expected of refereed papers in professional journals or formal Agency reports.

Documents
IDA Documents are used for the convenience of the sponsors or the analysts (a) to record substantive work done in quick reaction studies, (b) to record the proceedings of conferences and meetings, (c) to make available preliminary and tentative results of analyses, (d) to record data developed in the course of an investigation, or (e) to forward information that is essentially unanalyzed and unevaluated. The review of IDA Documents is suited to their content and intended use.

The work reported in this document was conducted under contract MDA 903 89 C 0003 for the Department of Defense. The publication of this IDA document does not indicate endorsement by the Department of Defense, nor should the contents be construed as reflecting the official position of that Agency.
The recommended practices in this document provide platform independence for Level III interactive video systems. Platform independence permits applications to run without modification on any hardware platform based on a general class of host computers. These recommendations establish a uniform application interface and command set for achieving platform independence. They are intended for system-level, not device-level independence. The recommendations address MS-DOS and PC DOS systems based on Intel 80x86 processor architecture. Other operating systems and architectures will be addressed in the future. Adoption of the current recommendations will benefit the interactive video community by furnishing a common ground for manufacturers, system integrators, courseware developers, and end-users in a field of rapidly changing technologies.
RECOMMENDED PRACTICES FOR INTERACTIVE VIDEO PORTABILITY

Philip Dodds  
*Randall House Associates*

Scott Lewis  
*DSL Associates*

David McFarling  
*SBCS Inc.*

Henry Mistrot  
*Video Associates Labs*

Geoff Snowman  
*Randall House Associates*

Jack Spiegelberg  
*VID Communications*

October 1990

Approved for public release; distribution unlimited.

IDA DOCUMENT D-855

INSTITUTE FOR DEFENSE ANALYSES  
Contract MDA 903 89 C 0003  
Tasks T-L2-565 and T-Z2-629.3
FOREWORD

These standard practices were developed by the Compatibility Committee of the Interactive Video Industry Association with the support and guidance of the Institute for Defense Analyses. They are being distributed for comment to major vendors, developers, and users of interactive video systems, software, and courseware in industry and government. Once this process is complete, these practices will be incorporated in Military Standard 1379D "Military Training Programs," for use by all departments and agencies of the Department of Defense. The version documented here has been reviewed by J. D. Fletcher for the Institute for Defense Analyses, by the Interactive Video Industry Association Board of Directors, and by senior technical staff of the following companies: Baker Videoactive, Online Computer Systems, Sony Corporation of America, and WICAT Systems.

This work was sponsored by the Office of the Assistant Secretary of Defense for Force Management and Personnel under the technical monitorship of Gary Boycan and by the American Forces Information Service, Office of the Assistant Secretary of Defense for Public Affairs, under the technical monitorship of LTC G. A. Redding, USA.
ABSTRACT

The recommended practices in this document provide platform independence for Level III interactive video systems. Platform independence permits applications to run without modification on any hardware platform based on a general class of host computers. These recommendations establish a uniform application interface and command set for achieving platform independence. They are intended for system-level, not device-level independence. The recommendations address MS-DOS and PC DOS systems based on Intel 80x86 processor architecture. Other operating systems and architectures will be addressed in the future. Adoption of the current recommendations will benefit the interactive video community by furnishing a common ground for manufacturers, system integrators, courseware developers, and end-users in a field of rapidly changing technologies.
## Contents

### 1. Introduction 1–1

- 1.1 Scope of the recommended practices 1–1
- 1.2 Goals and benefits 1–2
- 1.3 Document organization and intended audience 1–3
- 1.4 Document conventions 1–4
- 1.5 Comment submission 1–5

### 2. System overview 2–1

- 2.1 Hardware and operating system assumptions 2–2
- 2.2 Interface design criteria 2–2
- 2.3 The rationale for two interfaces 2–3
- 2.4 Service groups and command organization 2–4
- 2.5 Core and extended commands and parameters 2–4

### 3. Using the interfaces 3–1

- 3.1 An introduction to parameters and values 3–1
- 3.2 Using the binary interface 3–3
  - 3.2.1 General procedure 3–3
  - 3.2.2 Confirming that the binary interface exists 3–4
  - 3.2.3 Parameter packets 3–4
  - 3.2.4 Return values to parameters 3–6
- 3.3 Using the ASCII interface 3–7
  - 3.3.1 General procedure 3–7
  - 3.3.2 Confirming that the ASCII interface exists 3–7
  - 3.3.3 Command strings 3–8
  - 3.3.4 Response strings 3–8
- 3.4 Mixing ASCII and binary commands 3–9

### 4. Implementation details 4–1

- 4.1 Installation issues 4–1

April 15, 1990 Release R 1.0
Contents

4.1.1 VDI Management installation ........................................... 4-1
4.1.2 Logical device numbers .................................................. 4-2
4.2 Operating system issues .................................................... 4-3
  4.2.1 Operating system requirements ..................................... 4-3
  4.2.2 MS–DOS reentrancy limitations ................................... 4-3
4.3 ASCII interface issues ..................................................... 4-4
  4.3.1 ASCII string formats .................................................. 4-4
  4.3.2 ASCII string formal syntax ......................................... 4-5
  4.3.3 ASCII parameter value formats ................................... 4-7
  4.3.4 Device driver buffer behavior ..................................... 4-8
  4.3.5 Device driver IOCTL and mode options ......................... 4-9
4.4 Binary interface issues ................................................... 4-10
  4.4.1 Setting the software interrupt .................................... 4-9
  4.4.2 Binary parameter value formats .................................. 4-10

5. Command set summary tables ............................................. 5-1
  5.1 Command names and token numbers .................................. 5-1
  5.2 Parameter names and token numbers .................................. 5-4

6. System commands (sy) ....................................................... 6-1
  syCheckError ........................................................................... 6-2
  syErrorMsg ............................................................................. 6-6
  syGetState .............................................................................. 6-8
  syInit ....................................................................................... 6-12
  syQueue .................................................................................... 6-14
  syStop ...................................................................................... 6-19

7. Visual-management commands (vm) ..................................... 7-1
  7.1 Terms of reference ........................................................... 7-1
  7.2 General information and assumptions ............................... 7-2
    7.2.1 Overlayable graphics modes ....................................... 7-2
    7.2.2 Mode trapping .......................................................... 7-3
    7.2.3 Genlock control .......................................................... 7-3
    7.2.4 Graphics registration to the background video .............. 7-3
    7.2.5 VGA graphics versus CGA and EGA graphics ............... 7-3
    7.2.6 Logical versus physical colors .................................. 7-4
  7.3 Rounding methods for fades and dissolves ....................... 7-4
  vmFade ..................................................................................... 7-6
  vmGetPalette ......................................................................... 7-11
  vmGetState ............................................................................ 7-15
8. Videodisc commands (vd) 8–1

8.1 General information and assumptions 8–1
  8.1.1 CAV and CLV videodisc support 8–1
  8.1.2 Play and scan speeds 8–2
  8.1.3 Searches and instant jumps 8–2
  8.1.4 Fields, frames, and chapters 8–2
8.2 Rounding methods for player speeds 8–3
  vdGetState 8–5
  vdInit 8–11
  vdPassThru 8–15
  vdPlay 8–18
  vdScan 8–24
  vdSearch 8–27
  vdSet 8–30
  vdStep 8–35
  vdStill 8–37

9. XY-input commands (xy) 9–1

9.1 General information and assumptions 9–1
  9.1.1 Device mapping 9–1
  9.1.2 Handling the graphics plane and cursor 9–2
  9.1.3 Coordinate space mapping 9–2
  9.1.4 Buttons 9–3
9.2 Stream-mode and point-mode devices 9–3
  xyGetInput 9–4
  xyGetState 9–7
  xyInit 9–12
  xySet 9–15

A. Default positions of graphics relative to video A–1

  A.1 Terms of reference A–2
  A.2 Special considerations for VGA graphics A–3
    A.2.1 Differences in signals and timing A–3
<table>
<thead>
<tr>
<th>Contents</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A.2.2 Differences in the size of active graphics</td>
<td>A-3</td>
</tr>
<tr>
<td>A.3 Horizontal positions</td>
<td>A-4</td>
</tr>
<tr>
<td>A.3.1 General assumptions</td>
<td>A-4</td>
</tr>
<tr>
<td>A.3.2 NTSC</td>
<td>A-5</td>
</tr>
<tr>
<td>A.3.3 PAL</td>
<td>A-8</td>
</tr>
<tr>
<td>A.4 Vertical positions</td>
<td>A-11</td>
</tr>
<tr>
<td>A.4.1 General assumptions</td>
<td>A-11</td>
</tr>
<tr>
<td>A.4.2 NTSC</td>
<td>A-11</td>
</tr>
<tr>
<td>A.4.3 PAL</td>
<td>A-15</td>
</tr>
<tr>
<td>B. IBM PC and compatible graphics modes</td>
<td>B-1</td>
</tr>
<tr>
<td>C. Application programming examples</td>
<td>C-1</td>
</tr>
<tr>
<td>C.1 Using the ASCII interface</td>
<td>C-1</td>
</tr>
<tr>
<td>C.2 Using software interrupt calls</td>
<td>C-2</td>
</tr>
<tr>
<td>C.3 Library calls with parameter numbers</td>
<td>C-4</td>
</tr>
<tr>
<td>C.4 Analyzing bit fields</td>
<td>C-5</td>
</tr>
<tr>
<td>D. Error handling</td>
<td>D-1</td>
</tr>
<tr>
<td>D.1 General information</td>
<td>D-1</td>
</tr>
<tr>
<td>D.2 Error listings</td>
<td>D-2</td>
</tr>
<tr>
<td>D.2.1 Command problems</td>
<td>D-2</td>
</tr>
<tr>
<td>D.2.2 ASCII interface problems</td>
<td>D-3</td>
</tr>
<tr>
<td>D.2.3 Binary interface problems</td>
<td>D-3</td>
</tr>
<tr>
<td>D.2.4 Parameter problems</td>
<td>D-4</td>
</tr>
<tr>
<td>D.2.5 Hardware problems</td>
<td>D-5</td>
</tr>
<tr>
<td>D.2.6 System resources</td>
<td>D-6</td>
</tr>
<tr>
<td>D.2.7 Filing system problems</td>
<td>D-7</td>
</tr>
<tr>
<td>D.2.8 Miscellaneous problems</td>
<td>D-9</td>
</tr>
<tr>
<td>D.2.9 System group problems</td>
<td>D-10</td>
</tr>
<tr>
<td>D.2.10 Visual-management problems</td>
<td>D-10</td>
</tr>
<tr>
<td>D.2.11 Videodisc problems</td>
<td>D-11</td>
</tr>
<tr>
<td>D.2.12 XY-input device problems</td>
<td>D-12</td>
</tr>
<tr>
<td>Index</td>
<td>I-1</td>
</tr>
</tbody>
</table>
Figures

Figure 2-1. The general software architecture of a compliant system ........ 2-1

Figure 7-1. A simplified functional model of a video overlay subsystem ........ 7-2

Figure A-1. A simplified diagram of an overlayed display using CGA or EGA graphics ................................................. A-2

Figure A-2. One horizontal line of NTSC video with 640- or 320-pixel overlayed graphics ................................................ A-5

Figure A-3. One horizontal line of PAL video with 640- and 320-pixel overlayed graphics ................................................. A-8

Figure A-4. NTSC vertical timing with 200-line overlayed graphics .......... A-12

Figure A-5. PAL vertical timing with 200-line overlayed graphics ............ A-16
### Tables

Table 3-1. Example parameter block layout. ........................................ 3–5
Table 4-1. Formal BNF syntax for ASCII command and response strings .... 4–6
Table 4-2. ASCII bit field values ....................................................... 4–7
Table 4-3. Binary bit field values ..................................................... 4–11
Table 5-1. Service group prefix values for the binary interface. .............. 5–1
Table 5-2. Command word values for the binary interface ....................... 5–2
Table 5-3. A summary of command names including binary token numbers and types 5–3
Table 5-4. A summary of parameter labels including binary token numbers . 5–5
Table 6-1. System command names, token numbers, and types ................. 6–1
Table 7-1. Visual-management command names, token numbers, and types .. 7–1
Table 7-1. A simplified functional model of a video overlay subsystem ........ 7–2
Table 8-1. Videodisc command names, token numbers, and types ............. 8–1
Table 8-2. The effects of rounding on speed parameters for the Sony 2000 .... 8–3
Table 8-3. The effects of rounding on speed parameters for the Pioneer 4200 .. 8–3
Table 8-4. Example speed parameter values for boundary player speeds ...... 8–4
Table 9-1. XY-input command names, token numbers, and types ............... 9–1
Table B–1. IBM-compatible graphics modes .................................. B–1
1 Introduction

This document presents recommendations for commands and interface mechanisms used in level-III interactive video (IV) systems. The recommendations are based on functional definitions that were developed with input from IV manufacturers, developers and users. The foundation for these definitions includes an extensive set of models that were developed for internal use to ensure an approach that addresses current needs and capabilities while providing for future growth in IV technology.

This is a working document. As such, it serves two major purposes. First, manufacturers and developers can start working to implement the recommendations now instead of waiting for the completion of the formal ratification process. Second, industry members and end-users can contribute to the final recommendations by submitting comments.

1.1 Scope of the recommended practices

The recommended practices in this document provide platform independence but not device interoperability (plug-and-play). 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.

Furnishing such consistency is the immediate goal of the software definitions in this working document. This goal is not trivial. We anticipate changes in this document as specific needs are uncovered during the implementation of the recommendations. However, we anticipate that these changes will be uncovered early in the implementation process and will not be extensive enough to significantly impact implementations based on this document.

1. Device interoperability requires classes of related devices to furnish functionally identical services at the component level. This document does not address device interoperability.
Recommended Practices for Interactive Video Portability

The current recommendations address MS-DOS and PC DOS systems based on Intel 80x86 processor architecture. Other operating systems and architectures will be addressed in the future.

1.2 Goals and benefits

IV 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. We recognize the critical need for compatibility in IV technology to promote IV usage; lower costs for integrators, developers, and users; and encourage integration in a competitive market.

The specific goal of the current recommendations is to establish a uniform application interface and command set that supports unmodified IV application programs on different delivery systems. We do not endorse specific IV hardware systems or mandate the use of products from specific suppliers. Instead, our ultimate mission is to end such requirements.

Adoption of the current recommendations will benefit the IV industry by furnishing a common ground for manufacturers, systems integrators, courseware developers, and end-users in a field of rapidly changing technologies. The recommendations will furnish many benefits including:

- Broader acceptance of IV technology and subsequent growth of the marketplace.
- Assurance of long-lasting investments in IV applications and systems.
- Reduced need to change and support IV applications for specific hardware configurations.
- Increased productivity, lower maintenance costs, and improved application consistency across platforms.
- Higher quality and less costly IV applications resulting from a larger marketplace and the application of resources to software development instead of customizing software and platforms.
- Greater compatibility between international IV communities.
1.3 Document organization and intended audience

The information in this document is intended for IV manufacturers, systems integrators, authoring system developers, and software developers who wish to comply with the recommended practices. This document includes nine major sections and four appendices.

- Section 1, this section, includes background information, the scope of the recommendations, the benefits of adopting them, document conventions, and comment procedures.

- Section 2 includes an overview of the software architecture, the hardware and operating system assumptions, a summary of the IV service groups addressed by the command set, and an explanation of required and optional commands and parameters.

- Section 3 introduces the binary and ASCII application interfaces, explains the parameters and associated values used by the interfaces, and explains how to use the interfaces.

- Section 4 covers implementation issues including software installation, operating system issues, and ASCII and binary interface issues. It includes detailed information on command and parameter value formats, buffer behavior, device driver modes, and binary interface interrupt settings.

- Section 5 includes summary tables for the command set and its parameters including lists of available commands and parameters including their token numbers.

- Sections 6 through 9 contain detailed command descriptions for the system, visual-management, videodisc, and XY-input device service groups, respectively.

- Appendix A gives detailed recommendations on graphics registration relative to background video.

- Appendix B summarizes accepted graphics modes.

- Appendix C includes brief programming examples in BASIC and C.

- Appendix D covers error handling and lists error numbers with messages and explanations.

Note: Those who are interested primarily in the command set should read Section 1.4 and all of Section 3 for background information before proceeding to the command sections.
1.4 Document conventions

Familiarity with the conventions used in this document will aid understanding. These conventions are discussed below.

Figures and tables are referenced in the text by number. Each number consists of the major section number followed by a sequential figure or table number starting with one for each major section. For example, Table 2-3 refers to the third table in Section 2. Typically, figures and tables immediately follow the first paragraph in which they are referenced.

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.

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 0001011B. Decimal numbers are written without a suffix.

Command names and parameters in the text are in bold face. In the example vdPlay from=1000, vdPlay is a command and from is a parameter. Command names use mixed lower and upper case for clarity, while parameters use lower case only. However, case is not significant.

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.

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.

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 consisting 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.
1.5 Comment submission

To succeed, we need broad industry support and invite detailed commentary from IV developers, manufacturers, and users. Please describe questions and concerns fully and include alternate suggestions and options. In general, we cannot consider blanket criticisms that do not specify the natures of concerns or suggest alternatives. For example, we cannot consider “I don’t like this command” with no other commentary for future revisions.

Steps to follow:

1. Designate qualified technical representatives within your organization who can ably review the various major sections of the document.

2. Ask your technical representatives to:
   a. Respond to the recommendations on an item-by-item basis for your organization.
   b. Furnish a list of questions, comments, and suggestions, referencing the commands by name.
   c. Furnish a list of questions, comments, and suggestions for other sections of the documents referencing document number, section number, page number, and paragraph.

3. Comments must be submitted in writing and should include your mailing address, telephone number, and, if available, fax number.

4. To ensure that your comments are considered and that your organization receives further information and documents, address all correspondence to the address or fax number given in the accompanying cover letter. (If the cover letter is not available, contact the publisher given on the copyright page.)

5. Along with your comments please state specific areas of interest.

It may be useful to identify several technical representatives with different interests and backgrounds, and to have each individual or group review sections in their areas of expertise. This may make in-depth responses easier than having several individuals review the entire document.
Recommended Practices for Interactive Video Portability
2 System overview

Figure 2–1 shows the general software architecture of a compliant system. The recommended 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. VDI Management, in turn, executes the commands by calling appropriate low-level services and passes responses back to the application via the application interface. VDI Management is responsible for making different delivery platforms functionally compatible at the application interface level, thus enabling the implementation of platform independent commands.

Figure 2–1. The general software architecture of a compliant system

1. In the context of this document, the term "application" includes authoring software that may have been used to develop the application.
2.1 Hardware and operating system assumptions

The recommended practices make several assumptions about the computer, its operating system, and the IV hardware. These include:

- The computer is based on an Intel 80x86 processor with MS-DOS version 2.0 or higher, PC DOS version 2.0 or higher, or a functionally equivalent operating system.
- The computer uses an IBM PC/AT-compatible ROM BIOS
- The system has a graphics/video overlay capability using CGA, EGA, or VGA graphics, or uses two monitors, one with video and the other with graphics.
- One or more of several XY-input devices may be present (touch screen, mouse, light pen, bit pad, or other).
- One or more videodisc players or functionally equivalent video sources may be present.

Although this document describes interfaces for MS-DOS, we realize the import of other platforms. IV systems based on the Apple II and the Macintosh are widely used, and applications for OS/2 and Microsoft Windows are starting to appear. Procurement policies of the U.S. Government will encourage POSIX applications, and applications exist for UNIX, VMS, and others.

Therefore, we designed the recommended practices 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 are, of necessity, specific to MS-DOS.

2.2 Interface design criteria

The criteria used in designing the interfaces in the recommended practices include:

1. The recommended 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, computer-based training (CBT) authoring systems, artificial intelligence (AI) tools, prototyping systems, and database managers.

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

3. It should be possible to upgrade the recommended practices as required by technological developments without affecting existing applications.
4. The recommended practices should include both simple, easy-to-use commands for doing simple tasks and sophisticated functions to support the most demanding IV applications.

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

6. Except for features that affect the application interfaces, the recommended practices should not require a specific VDI Management software architecture.

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

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

We have used these criteria to define a software platform that should furnish a high degree of portability for a variety of IV applications while keeping implementation costs and run-time resource requirements to a minimum.

2.3 The rationale for two interfaces

The goal of supporting a variety of programming languages led us to propose that for MS-DOS one interface should include an installable device driver capable of standard, ASCII communications. 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.

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

In the current recommended practices the 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.

However, 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 should simplify their implementation and use.

In future versions, we would not rule out 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.
2.4 Service groups and command organization

Commands that map to related services are separated into service groups. Systems suppliers and applications 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 generally 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.

Currently, the recommended 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.

The general system group (sy) is the only group that must be included in all compliant implementations.

System suppliers 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. System suppliers 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 covers these issues in detail.)

2.5 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.

Core commands in a given service group furnish the general functionality required by IV applications. If a given service group is implemented, all core commands for the group must be included for the implementation to be compliant. Similarly, if a command is implemented, all core parameters for the command must be included.

Extended commands and parameters are nonportable. They 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 compliancy. However, an extended command may include core parameters so that the command's inclusion is optional, but, if it is included, the inclusion of the core parameters is mandatory.

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

Note: Some extended commands and parameters may be under consideration for future inclusion in the core command set. However, we do not guarantee that they will be included. An application that relies on these commands and parameters is noncompliant. However, an application that uses extended commands or parameters when present and still executes properly in their absence is compliant.
3 Using the interfaces

The recommended practices include two interfaces for MS-DOS, the binary interface and the ASCII interface. By providing two interfaces, the recommended 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 basic characteristics of each interface are:

- **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 to 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.

- **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.

The rest of this section explains how to use both interfaces. Section 4 gives detailed information on implementing the interfaces.

3.1 An 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.
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.

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. 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. 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 PAL, respectively.

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.

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 minimum parameter packet size depends on the number of parameters.

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

```
vdPlay
```

To search to frame 1000, then play to frame 1500 use:

```
vilPlay from=1000, to=1500
```

In the first case above, the binary interface does not pass a parameter packet with vdPlay. In the second case, 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.

1. Or vdPlay from 1000 to 1500; vdPlay from, 1000, to, 1500; or VDPLAY TO 1500 FROM 1000. All are equivalent, case and parameter order are not significant, and equals signs and commas are optional.
With both the binary and ASCII interfaces, the order of parameters is insignificant. For example:

```
vdPlay from=1000, to=2000
```

and:

```
vdPlay to=2000, from=1000
```

execute identically.

Note: 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.

### 3.2 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.

#### 3.2.1 General procedure

Applications will typically take the following steps to call VDI Management through the binary interface:

1. Build in application memory a packet containing parameter token numbers and values to pass to VDI Management with the command.
2. Load the token number for the command into the AX register.
3. Load the number of parameters contained in the packet into the BX register.
4. 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.)
5. Issue the appropriate software interrupt.
6. On return from the software interrupt, check the value of AX. If AX is non-zero, it contains an error code (see Section D), and the application should take appropriate action to recover.

The software interrupt used depends on the contents of the environment space. (See Section 4.4.1 for details.)
3.2.2 Confirming that the binary interface exists

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 syInit. To this end, the binary interface includes a 16-byte signature of the form:

IVERnnn...

where “nnn...” is restricted to ASCII decimal digits (ASCII 30H–39H), the decimal point (ASCII 2EH), and terminal NUL (ASCII 00H) characters to pad to 16 bytes if necessary. This is the official version number for the recommended practices implemented by VDI Management as it would be returned by the command syGetState ivver (see Section 6). The version number for the current recommendations is “1.0”, which yields a signature of:

IVER1.0

followed by eight NULL characters.

This signature resides at the address pointed to by the IVINT interrupt vector minus 10H. (See Section 4.4.1 for more information on IVINT.)

Note: To confirm that the binary interface is installed, an application must determine the address pointed to by IVINT, then retrieve the information stored in the 16-byte paragraph immediately preceding that address. (We assume that languages that use the binary interface have this capability.)

3.2.3 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. (Section 4.4.2 gives detailed information on parameter value formats.)

Parameter token numbers are constants, as defined in Section 5.2. 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 and unsigned 32-bit integers, pointers consisting of a 16-bit segment and a 16-bit offset, and signed 32-bit fractional numbers.

For example, assume an application has initialized VDI Management and displayed video on the monitor, and 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:

```plaintext
AX 3081 ; token number of vdPlay command
BX 2 ; packet contains two parameters
```
Section 3. Using the Interfaces

ES:DI is a pointer to a packet containing:

<table>
<thead>
<tr>
<th>Block</th>
<th>Token/Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>24; token number of from parameter</td>
</tr>
<tr>
<td>2</td>
<td>1000; value of from parameter</td>
</tr>
<tr>
<td>3</td>
<td>48; token number of to parameter</td>
</tr>
<tr>
<td>4</td>
<td>2000; value of to parameter</td>
</tr>
</tbody>
</table>

(Note: All values in the above example use decimal notation.)

The length of the parameter packet varies with the number of parameters. A packet with five parameters is laid out as shown in Table 3–1. Longer packets are possible by simply adding the extra parameters at the end.

<table>
<thead>
<tr>
<th>Address</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>ES:DI[0]</td>
<td>Parameter 1 token number</td>
</tr>
<tr>
<td>ES:DI[4]</td>
<td>Parameter 1 value</td>
</tr>
<tr>
<td>ES:DI[8]</td>
<td>Parameter 2 token number</td>
</tr>
<tr>
<td>ES:DI[C]</td>
<td>Parameter 2 value</td>
</tr>
<tr>
<td>ES:DI[10]</td>
<td>Parameter 3 token number</td>
</tr>
<tr>
<td>ES:DI[14]</td>
<td>Parameter 3 value</td>
</tr>
<tr>
<td>ES:DI[18]</td>
<td>Parameter 4 token number</td>
</tr>
<tr>
<td>ES:DI[1C]</td>
<td>Parameter 4 value</td>
</tr>
<tr>
<td>ES:DI[20]</td>
<td>Parameter 5 token number</td>
</tr>
<tr>
<td>ES:DI[24]</td>
<td>Parameter 5 value</td>
</tr>
</tbody>
</table>

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 $8 \times 22$ bytes or a 176-byte block.

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 is insignificant. On return, all registers are unchanged except AX. AX contains zero if the command was successful, or an error code indicating a problem. When VDI Management rounds values, it changes the values in the parameter packet in application memory to the rounded values. (See the service group command sections for more information on rounding.)

2. Note that addresses of parameter tokens and value 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.
3.2.4 Return values to parameters

The contents of a parameter packet 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 Section 3.2.3.

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 calculation or by interrogating the hardware, it puts the value into the appropriate parameter value block.

For vdGetState frame, the application might pass:

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

On return, the values might be:

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

(Note: All values in the above example use decimal notation.)

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

Note 1: 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.

Note 2: VDI Management may round parameters such as play speeds. If so, VDI Management changes the values in the parameter packet to the actual values used after rounding and subsequent queries return actual values instead of requested values.
3.3 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.

3.3.1 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. Applications will typically take the following steps to call VDI Management through the ASCII interface:

1. Format a command string specifying the required function.
2. Open the device driver for writing.
3. Write the command string to the device driver.
4. Close the device driver.
5. Open the device driver for reading.
6. Read the response string from the device driver.
7. Close the device driver.
8. 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.

The exact method of communicating with the driver depends on the programming system.

3.3.2 Confirming that the ASCII interface exists

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 IVDEV if it does not exist. Therefore, an application should open the file, write an syInit command to it (see Section 6), and read the response string.

- If the response string consists of the ASCII characters "OK" followed by CR/LF, syInit was successful. The application should continue normally.
- If the response string consists of "ERROR n..." where "n..." is an error number followed by CR/LF, then 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.
- 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 or serious problems exist with VDI Manage-
Recommended Practices for Interactive Video Portability

ment. Some languages may automatically create a file named IVDEV if the driver is not installed. The application should politely exit with an error message and, if necessary, delete the bogus file (preferred) or at least tell the user that a bogus IVDEV file may have been created.

Note: 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.

3.3.3 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. (Section 4.3 gives detailed information on formal command string syntax and parameter value data types.)

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

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.

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.

3.3.4 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:

```
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).

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:** 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.

### 3.4 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, 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.

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.
4 Implementation details

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.

4.1 Installation issues

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

4.1.1 VDI Management installation

Specific methods of installing VDI Management are outside the scope of the recommended practices. VDI developers can use any appropriate method. Two obvious possibilities are terminate-and-stay-resident (TSR) software and software that takes an application name as a command-line parameter and spawns the application.

For applications that use the binary interface only, the device driver need not be loaded. Similarly, for applications that use the ASCII interface only, the binary interface need not be loaded.

Although VDI implementers must supply both interfaces for an implementation to be compliant, both do not have to be installed in the IV system.
4.1.2 Logical device numbers

Some commands accept a logical device number as a parameter. These include videodisc commands, which may be used on systems with multiple videodisc players, and XY-input commands, which may be used on systems with multiple XY-input devices.

Often, applications will need to know the relationship between device numbers and physical devices. For example, if a system includes both NTSC and PAL videodisc players, an NTSC application should ensure that it controls the NTSC player.

When logical device numbers are allocated, VDI Management lets users assign numbers to specific devices using an appropriate setup facility. Device numbers must be contiguous and start at zero, but VDI Management does not make any other assumptions about logical-to-physical device mapping.

Providing a set-up facility for users to assign logical device numbers and requiring contiguous device numbers starting with zero are compliance requirements.

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 cannot possibly use this information while executing. Therefore, application authors must clearly state any requirements for a particular device number mapping, so that users can set up VDI Management appropriately.

For example, assume that a system has two XY-input devices, a mouse and a touchscreen. Also assume that the mouse is installed as device zero and the touchscreen as device one. An application that requires the touchscreen to be device zero cannot change the logical numbering at run time. Therefore, the application author must clearly inform the user that the touchscreen must be installed as device zero when the user installs VDI Management.

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.
4.2 Operating system issues

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

4.2.1 Operating system requirements

To support the MS-DOS version of the recommended practices, the operating system must be 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 recommended practices regardless of the attached IV hardware.

Developers may provide VDI Management software that requires a specific version of MS-DOS. We would describe this 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 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 IV hardware and courseware should ensure that the versions of MS-DOS, VDI Management, and courseware that they purchase are compatible.

4.2.2 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.

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. Also, 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.

VDI Management assumes it can call MS-DOS during any call to the binary interface. Therefore, application software must 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
must provide mechanisms to ensure that VDI Management is called at appropriate times only.

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.

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.

Note: The binary interface cannot be called directly from within the device driver because the interface, in turn, calls VDI Management and VDI Management must be able to call MS-DOS.

4.3 ASCII interface issues

ASCII interface issues include ASCII text string formats, parameter value formats, device driver buffer behavior, device driver communications modes, and using the IOCTL function. The following subsections cover these issues.

4.3.1 ASCII string formats

Command strings are simply tokens separated by delimiters. Return strings consist of comma-separated values. The following subsections discuss command and return strings formats.

Command string tokens

A command string is a series of tokens, separated by delimiters, and ending with a CR. Tokens are strings of one or more printable characters. They include command names, parameter identifiers, and parameter values. Command names and parameter identifiers consist of characters in the ranges “A” to “Z” and “a” to “z”. Case is not significant. For example, the command name “vdPlay” could be supplied as “vdplay”, “VDPLAY”, or even “vDpLaY”. Parameter values consist of characters in the ranges “a” to “z”, “A” to “Z”, and “0” to “9” plus “.”, “+”, and “-”.

4-4 Release R 1.0  April 15, 1990
Command string delimiters

Delimiters are the characters equals (ASCII 3DH), space (ASCII 20H), tab (ASCII 09H), LF (ASCII 0AH) and comma (ASCII 2CH). Two adjacent delimiters are treated as if they were a single delimiter. Extra delimiters at the start or end of the string are ignored.

Command string length

Command strings can be at most 255 characters including the CR. Redundant delimiters do not count towards the 255-character limit.

Multiple commands separated by CRs may be included in a single string and sent to the ASCII interface with single write operation. The 255-character limit applies to each command in the write operation, not to the write operation as a whole.

Response strings

Response strings always end with CR/LF. 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.

4.3.2 ASCII string formal syntax

The formal syntax description for ASCII command and response strings in Table 4–1 uses a notation derived from the Backus Naur Form (BNF). The syntax uses the following rules.

1. Angle brackets (<> ) enclose items that are defined by the formal descriptions.
2. Vertical bars (|) separate sets of alternatives—in deriving a valid command string, one of the alternatives should be chosen.
3. Square brackets ([ ]) enclose optional items or sets of items. Their presence or absence does not affect the string's validity.
4. Spaces separate sets of required items that should occur in the given order.
5. The “:=” sign means “consists of.” The item on the left of “:=” consists of the definition on the right.
6. The strings “equals”, “space”, “tab”, “line feed”, and “comma” are delimiters and stand for the indicated characters.
7. The string “carriage return” is a terminator and stands for the indicated character.
8. Items in quotes (“”) are string literals and stand for themselves.
Table 4-1. Formal BNF syntax for ASCII command and response strings

<table>
<thead>
<tr>
<th>Syntax Element</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;command string&gt;</td>
<td>:= [&lt;delimiter string&gt;] &lt;command name&gt; [ &lt;parameter string&gt; ] [ &lt;delimiter string&gt; ] carriage return</td>
</tr>
<tr>
<td>&lt;response string&gt;</td>
<td>:= &quot;OK&quot; carriage return line feed</td>
</tr>
<tr>
<td>&lt;parameter string&gt;</td>
<td>:= &lt;parameter&gt;</td>
</tr>
<tr>
<td>&lt;parameter&gt;</td>
<td>:= &lt;delimiter string&gt; &lt;parameter id&gt; [ &lt;delimiter string&gt; ] &lt;parameter value&gt;</td>
</tr>
<tr>
<td>&lt;parameter id&gt;</td>
<td>:= &lt;letter string&gt; [ &lt;digit string&gt; ]</td>
</tr>
<tr>
<td>&lt;command name&gt;</td>
<td>:= &lt;letter string&gt;</td>
</tr>
<tr>
<td>&lt;result string&gt;</td>
<td>:= &lt;parameter value&gt; [ &lt;parameter value&gt; comma &lt;result string&gt; ]</td>
</tr>
<tr>
<td>&lt;parameter value&gt;</td>
<td>:= &lt;letter string&gt;</td>
</tr>
<tr>
<td>&lt;delimiter string&gt;</td>
<td>:= &lt;delimiter&gt;</td>
</tr>
<tr>
<td>&lt;delimiter&gt;</td>
<td>:= &lt;delimiter&gt;</td>
</tr>
<tr>
<td>&lt;letter string&gt;</td>
<td>:= &lt;letter&gt;</td>
</tr>
<tr>
<td>&lt;digit string&gt;</td>
<td>:= &lt;digit string&gt;</td>
</tr>
<tr>
<td>&lt;file name&gt;</td>
<td>:= &lt;legal file name string for operating system&gt;</td>
</tr>
<tr>
<td>&lt;number&gt;</td>
<td>:= [&lt;sign&gt;] &lt;digit string&gt;</td>
</tr>
<tr>
<td>&lt;digit&gt;</td>
<td>:= &lt;digit string&gt;</td>
</tr>
<tr>
<td>&lt;delimiter&gt;</td>
<td>:= &quot;A&quot;</td>
</tr>
<tr>
<td>&lt;sign&gt;</td>
<td>:= &quot;+&quot;</td>
</tr>
</tbody>
</table>

1Redundant delimiters in a <delimiter string> are ignored and do not count toward any length limits. For a <command string>, redundant delimiters include all leading delimiters, all trailing delimiters after the carriage return, and any instance of more than one <delimiter> in a <delimiter string> between the <command name> and the carriage return.

2This is not a complete definition. The items on the left can take on a limited range of values (see Section 5).

The formal description above omits the 255-character limit and the equivalency of lower and upper case.

4-6 Release R 1.0 April 15, 1990
4.3.3 ASCII parameter value formats

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

Numbers

Numeric parameters include device numbers, mode numbers, time periods, and error numbers. These are passed in decimal format, which is defined as:

- an optional sign ("+" or "-") followed by a string of decimal digits (for numbers without fractional parts);

or:

- an optional sign ("+" or "-") followed by an optional string of decimal digits followed by "." followed by a mandatory string of decimal digits (for numbers with fractional parts);

Note that by these definitions "5", "0.5", and ".5" are legal but ".5" is not (a decimal point with no fractional part).

Bit fields

A bit field is represented as a 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. Table 4-2 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. (Note that the system (sy) group is always present if VDI Management is working.)

<table>
<thead>
<tr>
<th>Service group</th>
<th>Decimal value returned by syGetState</th>
</tr>
</thead>
<tbody>
<tr>
<td>System (sy)</td>
<td>1</td>
</tr>
<tr>
<td>Visual management (vm)</td>
<td>2</td>
</tr>
<tr>
<td>Videodisc (vd)</td>
<td>4</td>
</tr>
<tr>
<td>XY input (xy)</td>
<td>8</td>
</tr>
</tbody>
</table>

Assume a system supports vm and vd, but not xy commands. The syGetState return value is 1 + 2 + 4 or decimal 7. If all command groups are supported, the return value is 15.
Some return strings for query parameters include values other than numbers. These strings consist of at most upper-case alpha characters, decimal digits, commas, the decimal point, and a sign (+ or -).

4.3.4 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. 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.\(^1\)

When the driver receives a character, it adds it to the command buffer and checks to see if it is a CR. If it is not, the driver returns. If it is, the driver and VDI Management process the contents of the command buffer, execute the command, and generate a response string. The response string replaces any existing contents of the response buffer.

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).

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.

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 driver before writing at least one command or it may read an invalid response left by the previous application.

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).

1. Note that 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.
4.3.5 Device driver IOCTL and mode options

Developers must make several choices about which functions to provide with MS-DOS device drivers. These include support for IOCTL functions, output until busy, and generic IOCTL commands.

Although the recommended practices do not require any of these capabilities, the device drive may furnish them and still be compliant. For example, a driver might use IOCTL commands to switch between an compliant mode and a dedicated, nonportable mode. However, applications that use these facilities are noncompliant because the facilities are not part of the recommended practices and may not be supported.

Another choice is whether device drivers should operate in ASCII ("cooked") or binary ("raw") mode. The driver should work properly using either mode because applications may choose either and be compliant.

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 CP/M-compatible file control blocks. Only the read/write functions that use handles work with device drivers.

This does not mean that routines within a programming system must explicitly pass handles to the application. However, the system should not use MS-DOS interrupt 21H functions 0F, 10-17, 1A, 21-24, 27, and 29H, which work with file control blocks. This is unlikely to be an issue because using file handle functions has been the method of preference since the introduction of MS-DOS 2.0.

4.4 Binary interface issues

Binary interface issues include establishing which interrupt will be used and parameter value formats. The following subsections cover these issues.

4.4.1 Setting the software interrupt

The binary interface software interrupt is a user interrupt in the range 60H to 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-

---

2. In cooked mode, certain characters such as control characters and EOF are subject to MS-DOS interpretation; in raw mode, they are not. A device driver that is opened with the MS-DOS open handle function, Int 21H function 3DH, is by default in cooked mode. However, an MS-DOS IOCTL function, Int 21H function 44H, can force a handle to raw mode. 

---

April 15, 1990 Release R 1.0 4-9
character string representing a value from 60H through 66H. 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 politely declines to execute.

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. If the variable value is invalid, the application politely declines to execute.

### 4.4.2 Binary parameter value formats

Binary parameter values include integers, real numbers, bit fields, strings, pointers, and color arrays. These formats are discussed below.

#### Integers

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

#### Real numbers

Real numbers are passed by multiplying the real number by 65,536, rounding towards negative infinity, and using the integer part expressed in hexadecimal with 2’s complement. Four bytes can represent fractional numbers in the range -32768 (80000000H) to +32767.99998 (7FFFFFFFH) with an accuracy of ±1/65536 (±0.000015).

In this notation, the most significant 16 bits represent the integer part of the number in 2’s complement. The least significant 16 bits represent a positive binary fraction to be added to that integer. For example, to represent \(-\frac{1}{2}\), set the most significant 16 bits to FFFFH or -1 decimal and the least significant 16 bits to 8000H or \(+\frac{1}{2}\) decimal.

Normal 2’s complement addition and subtraction still yields correct results. For example:

- FFFF8000 \((-\frac{1}{2})\) + FFFF4000 \((-\frac{3}{2})\) = FFFEC000 \((-1\frac{1}{4})\)
- FFFF4000 \((-\frac{3}{4})\) + 00002000 \((+2)\) = 00014000 \((1\frac{1}{4})\)
- 00001999 \((0.1)\) + 00003333 \((0.2)\) = 00004CC0 \((0.299988)\)
Section 4. Implementation details

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, on or off, true or false, or present or absent.

Table 4-3 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.)

<table>
<thead>
<tr>
<th>Service group</th>
<th>Least significant byte value is group if present</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Binary</td>
</tr>
<tr>
<td>System (sy)</td>
<td>00000001</td>
</tr>
<tr>
<td>Visual management (vm)</td>
<td>00000010</td>
</tr>
<tr>
<td>Videodisc (vd)</td>
<td>00000100</td>
</tr>
<tr>
<td>XY input (xy)</td>
<td>00001000</td>
</tr>
</tbody>
</table>

For example, if the VDI implementation supports vm and vd commands, but not xy, the least significant byte returned by syGetState is 00000111B or 07H. (See Appendix C for a code fragment showing how to analyze bit fields.)

Strings

Strings returned via the binary interface are passed by reference. The return value is a 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. The most significant 16 bits of the pointer contain the string’s segment address; the least significant 16 bits contain the offset within the segment.

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

Color arrays

The vmGetPalette and vmSetPalette commands use arrays containing palette information. These arrays are passed by reference. The parameter packet contains three parameters, a logical color parameter, a length parameter, and an array address parameter.
The array parameter value is a long 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:

- Byte 0, the least significant byte, represents B(lue);
- Byte 1 represents G(reen);
- Byte 2 represents R(ed); and
- Byte 3, the most significant byte is reserved and is set to zero.

The reserved byte must be set to zero—VDI Management returns error 51 (Parameter value invalid or out of range) if it is not.

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

The pointer format is the same as that for a string—the most significant 16 bits contain the segment address and the least significant 16 bits contain the offset within that segment.
5 Command set summary tables

This section includes summary tables of the commands and parameters for both the ASCII and binary interfaces with token numbers for the binary interface. See Section 3 for detailed information on how to use the interfaces and Sections 4.3 and 4.4 for information on ASCII string syntax and ASCII and binary parameter value formats. Appendixes C and D provide programming examples and cover error handling, respectively.

5.1 Command names and token numbers

The binary interface uses token numbers instead of command names. These numbers map directly to ASCII command 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.

Table 5-1 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 recommendations evolve and allowing the determination of which service group an token number is in with a single shift right and compare operation.

<table>
<thead>
<tr>
<th>Service group prefix</th>
<th>Value (decimal)</th>
<th>Value (hexadecimal)</th>
</tr>
</thead>
<tbody>
<tr>
<td>sy</td>
<td>1024</td>
<td>0400</td>
</tr>
<tr>
<td>vm</td>
<td>2048</td>
<td>0800</td>
</tr>
<tr>
<td>vd</td>
<td>3072</td>
<td>0C00</td>
</tr>
<tr>
<td>xy</td>
<td>4096</td>
<td>1000</td>
</tr>
</tbody>
</table>

Table 5-1. Service group prefix values for the binary interface.
Recommended Practices for Interactive Video Portability

Table 5–2 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 recommended 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.

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

Table 5–3 lists ASCII commands and their equivalent 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, in C a series of #define statements might include:

```
#define SY 1024
#define VM 2048
#define INIT 7
```
Then, the token number for syInit could be derived with:

\[ SY + INIT \]

The table also indicates whether the commands are core or extended. Core commands must be implemented for a given service group to be compliant. Extended commands are optional and should be considered nonportable unless an application is written to use them if present and handle their absence.

<table>
<thead>
<tr>
<th>ASCII name</th>
<th>Token number (decimal)</th>
<th>Token number (hexadecimal)</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>syCheckError</td>
<td>1025</td>
<td>0401</td>
<td>Core</td>
</tr>
<tr>
<td>syErrorMs</td>
<td>1026</td>
<td>0402</td>
<td>Extended</td>
</tr>
<tr>
<td>syGetState</td>
<td>1030</td>
<td>0406</td>
<td>Core</td>
</tr>
<tr>
<td>syInit</td>
<td>1031</td>
<td>0407</td>
<td>Core</td>
</tr>
<tr>
<td>syQueue</td>
<td>1034</td>
<td>040A</td>
<td>Core</td>
</tr>
<tr>
<td>syStop</td>
<td>1044</td>
<td>0414</td>
<td>Core</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ASCII name</th>
<th>Token number (decimal)</th>
<th>Token number (hexadecimal)</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>vmFade</td>
<td>2051</td>
<td>0803</td>
<td>Core</td>
</tr>
<tr>
<td>vmGetPalette</td>
<td>2053</td>
<td>0805</td>
<td>Core</td>
</tr>
<tr>
<td>vmGetState</td>
<td>2054</td>
<td>0806</td>
<td>Core</td>
</tr>
<tr>
<td>vmlInit</td>
<td>2055</td>
<td>0807</td>
<td>Core</td>
</tr>
<tr>
<td>vmSetGraphics</td>
<td>2062</td>
<td>080E</td>
<td>Core</td>
</tr>
<tr>
<td>vmSetPalette</td>
<td>2063</td>
<td>080F</td>
<td>Core</td>
</tr>
<tr>
<td>vmSetTrans</td>
<td>2064</td>
<td>0810</td>
<td>Core</td>
</tr>
<tr>
<td>vmSetVideo</td>
<td>2065</td>
<td>0811</td>
<td>Core</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ASCII name</th>
<th>Token number (decimal)</th>
<th>Token number (hexadecimal)</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>vdGetState</td>
<td>3078</td>
<td>0C06</td>
<td>Core</td>
</tr>
<tr>
<td>vdInit</td>
<td>3079</td>
<td>0C07</td>
<td>Core</td>
</tr>
<tr>
<td>vdPassThru</td>
<td>3080</td>
<td>0C08</td>
<td>Core</td>
</tr>
<tr>
<td>vdPlay</td>
<td>3081</td>
<td>0C09</td>
<td>Core</td>
</tr>
<tr>
<td>vdScan</td>
<td>3083</td>
<td>0C0B</td>
<td>Core</td>
</tr>
<tr>
<td>vdSearch</td>
<td>3084</td>
<td>0C0C</td>
<td>Core</td>
</tr>
<tr>
<td>vdSet</td>
<td>3085</td>
<td>0C0D</td>
<td>Core</td>
</tr>
<tr>
<td>vdStep</td>
<td>3090</td>
<td>0C12</td>
<td>Core</td>
</tr>
<tr>
<td>vdStill</td>
<td>3091</td>
<td>0C13</td>
<td>Core</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ASCII name</th>
<th>Token number (decimal)</th>
<th>Token number (hexadecimal)</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>xyGetInput</td>
<td>4100</td>
<td>1004</td>
<td>Core</td>
</tr>
<tr>
<td>xyGetState</td>
<td>4102</td>
<td>1006</td>
<td>Core</td>
</tr>
<tr>
<td>xylInit</td>
<td>4103</td>
<td>1007</td>
<td>Core</td>
</tr>
<tr>
<td>xySet</td>
<td>4109</td>
<td>100D</td>
<td>Core</td>
</tr>
</tbody>
</table>
5.2 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.

Table 5-4 lists parameter names and their binary token numbers. Parameters are numbered 1–67 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 recommended practices evolve.

The table also indicates whether the parameters are core or extended. Core parameters for a given service group must be implemented for compliance. Extended parameters are optional and should be considered nonportable unless an application is written to use them if present and handle their absence.
Table 5-4. A summary of parameter labels including binary token numbers

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

$^1$Currently used only as an extended parameter in the xy service group.

$^2$Currently used only as an extended parameter in the vd service group.

$^3$Currently used only as an extended parameter in the vm service group.
6 System commands (sy)

This section describes commands that relate to overall VDI software operation. These commands initialize the basic IV 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 6-1 lists the commands covered in this section, their token numbers, and their types.

Table 6-1. ASCII command names, token numbers, and types

<table>
<thead>
<tr>
<th>ASCII command name¹</th>
<th>Binary Interface token number (decimal)</th>
<th>Type²</th>
</tr>
</thead>
<tbody>
<tr>
<td>syCheckError</td>
<td>1025</td>
<td>Core</td>
</tr>
<tr>
<td>syErrorMsg</td>
<td>1026</td>
<td>Extended</td>
</tr>
<tr>
<td>syGetState</td>
<td>1030</td>
<td>Core</td>
</tr>
<tr>
<td>syInit</td>
<td>1031</td>
<td>Core</td>
</tr>
<tr>
<td>syQueue</td>
<td>1034</td>
<td>Core</td>
</tr>
<tr>
<td>syStop</td>
<td>1044</td>
<td>Core</td>
</tr>
</tbody>
</table>

¹Upper or lower case for command names is not significant.
²Compliant implementations must support "Core" commands. Supporting "Extended" commands is optional, and these commands should be considered nonportable unless properly handled when absent.
syCheckError

Parameters

ASCII

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Core or extended</th>
<th>Associated calling value</th>
<th>Type as ASCII</th>
<th>Associated return value</th>
<th>Type as ASCII</th>
<th>Default if parameter not used</th>
</tr>
</thead>
<tbody>
<tr>
<td>command</td>
<td>Core</td>
<td>None</td>
<td>N/A</td>
<td>Command name that caused error</td>
<td>Text</td>
<td>No action</td>
</tr>
<tr>
<td>errno</td>
<td>Core</td>
<td>None</td>
<td>N/A</td>
<td>Last detected error number</td>
<td>Integer</td>
<td>No action</td>
</tr>
</tbody>
</table>

At least one parameter is required or an error is returned.

Binary

Command code: 1025 decimal.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Core or extended</th>
<th>Token number (decimal)</th>
<th>Type</th>
<th>Associated calling value</th>
<th>Associated return value</th>
<th>Default if parameter not used</th>
</tr>
</thead>
<tbody>
<tr>
<td>command</td>
<td>Core</td>
<td>10</td>
<td>Integer</td>
<td>Any value</td>
<td>Command token that caused error</td>
<td>No action</td>
</tr>
<tr>
<td>errno</td>
<td>Core</td>
<td>21</td>
<td>Integer</td>
<td>Any value</td>
<td>Last detected error number</td>
<td>No action</td>
</tr>
</tbody>
</table>

At least one parameter is required or an error is returned.

Description

Summary

syCheckError returns the number of the last error detected by VDI Management, if present, and the command that caused the error. syCheckError then clears this error information.

General discussion

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 after being accepted. syCheckError is provided to detect these types of errors, although it will return the last error regardless of the error's cause.
Assume a player accepts a valid \texttt{vdPlay} command without a \texttt{wait} modifier (see Section 8). VDI Management will return success immediately ("OK" for the ASCII interface, \texttt{AX = 0} for the binary interface). If the player then fails during the specified motion sequence, an error state exists. \texttt{syCheckError} determines if such a situation has arisen and, if so, returns the error. Similarly, if a fade is successfully initiated and subsequently fails, an unreported error results. Finally, a queued command may result in an error although \texttt{syQueue execute} was successful. \texttt{syCheckError} returns the error resulting from the queued command. However, the return does not specify which queued command generated the error.

While actual implementations may vary in the way they store and translate the information needed by \texttt{syCheckError}, in concept VDI Management maintains three buffers for returning error information, the response buffer, the check error buffer, and the check command buffer. The contents of these buffers (or the translations thereof) depends on which interface is used.

For the ASCII interface:

- the response buffer contains the return string for the most recent command, either "ERROR n..." where "n..." is an error number, "OK", or requested information;
- the check error buffer contains the error number as "n..." of the most recent error caused by any command or "0" if no error has occurred or the buffer has been cleared; and
- the check command buffer contains the name of the command that caused the error or "OW" if no error has occurred or the buffer has been cleared.

For the binary interface:

- the response buffer contains the value to be returned in the AX register for the most recent command, either an error number or zero if no error occurred;
- the check error buffer contains the error number of the most recent error caused by any command or zero if no error has occurred or the buffer has been cleared; and
- 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.

The \texttt{command} parameter requests the command that caused the last error, if present. Issuing \texttt{syCheckError} with \texttt{command} clears the error number in the check number buffer as well as the command name or token in the check command buffer.
syCheckError

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 as well as the error number in the check number buffer.

Notes
1. 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.

2. Only syCheckError and syInit can clear the check error 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.

3. 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 error and check command 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.

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

Returns

ASCII
On success: For errno parameter, last error number as "n..." or "0" if no error. For command parameter, name of command causing last error as an upper-case alpha string or "OK" if no error.

On failure: "ERROR n...",

Binary
On success: AX = 0. Value associated with the errno parameter is the last error number as a 32-bit integer or 0 if no error. Value associated with the command parameter is the token number of the command that caused the error or 0 if no error.

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

See also: syErrorMsg, syInit, syQueue.
syCheckError

Examples

ASCII

Pass non-
syCheckError
command that
causes an error

Pass non-
syCheckError
command that
causes an error

Now query for last
error and related
command

Now query for last
error and related
command

Requery after
clearing

Requery after
clearing

Binary

Query for last
error and related
command token

Query for last
error and related
command token

After return

After return

<table>
<thead>
<tr>
<th>AX</th>
<th>1025</th>
<th>; syCheckError decimal ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>BX</td>
<td>1</td>
<td>; number of parameters (required for return)</td>
</tr>
<tr>
<td>ES:DI[0]</td>
<td>21</td>
<td>; errno decimal ID</td>
</tr>
<tr>
<td>ES:DI[4]</td>
<td>any value</td>
<td>; place holder for value on return</td>
</tr>
<tr>
<td>ES:DI[8]</td>
<td>10</td>
<td>; command decimal ID</td>
</tr>
<tr>
<td>ES:DI[C]</td>
<td>any value</td>
<td>; place holder for value on return</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>AX</th>
<th>0</th>
<th>; returns 0 if successful (nonzero if not)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ES:DI[4]</td>
<td>error no.</td>
<td>; number of most recent error, if present</td>
</tr>
<tr>
<td>ES:DI[C]</td>
<td>token no.</td>
<td>; token number of command causing error</td>
</tr>
</tbody>
</table>

Missing parameter value

clears all last error information

“53,SYERRORMSG”

“0,OK”

“ERROR 53”;

“53,SYERRORMSG”; clears all last error information

“0,OK”
Recommended Practices for Interactive Video Portability

syErrorMsg

Last revision: R 1.0
Type: extended

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Core or extended</th>
<th>Associated calling value</th>
<th>Type as ASCII</th>
<th>Associated return value</th>
<th>Type as ASCII</th>
<th>Default if parameter not used</th>
</tr>
</thead>
<tbody>
<tr>
<td>errno</td>
<td>Core</td>
<td>Error number</td>
<td>Integer</td>
<td>Error description string</td>
<td>Text</td>
<td>Causes error</td>
</tr>
</tbody>
</table>

Errno must be specified or an error is returned.

Binary

Command code: 1026 decimal.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Core or extended</th>
<th>Token number (decimal)</th>
<th>Type</th>
<th>Associated calling value</th>
<th>Associated return value</th>
<th>Default if parameter not used</th>
</tr>
</thead>
<tbody>
<tr>
<td>errno</td>
<td>Core</td>
<td>21</td>
<td>Integer</td>
<td>Error number</td>
<td>None</td>
<td>Causes error</td>
</tr>
<tr>
<td>pmsg</td>
<td>Core</td>
<td>37</td>
<td>Pointer</td>
<td>Any value</td>
<td>Pointer to error description string</td>
<td>Causes error</td>
</tr>
</tbody>
</table>

Both parameters must be specified or an error is returned.

Description

Summary

syErrorMsg returns an ASCII string of up to 255 characters that describes the specified error number.

General discussion

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 a description. VDI Management developers may opt to keep error descriptions in memory or a separate file.

Errno parameter

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

Pmsg parameter

The pmsg parameter for the binary interface has an associated return value that points to the location of the descriptive string.
Section 6. System commands (sy)

syErrorMsg

Notes

1. Appendix D lists strings for specific error numbers.
2. Trying to queue syErrorMsg causes error 177 (Command cannot be queued) at the time of the attempt.

Returns

ASCII

On success: Error description string.

On failure: "ERROR n...".

Binary

On success: AX = 0. Value associated with the pmsg parameter is a 32-bit pointer to a null-terminated error description string.

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

See also: syCheckError, syQueue.

Examples

ASCII

Pass an invalid parameter
syErrorMsg error=55
(return) "ERROR 48"

Get string describing error 48
syErrorMsg errno=48
(return) "UNKNOWN PARAMETER"

Pass insufficient parameters
syErrorMsg
(return) "ERROR 49"

Get string describing error 49
syErrorMsg errno=49
(return) "INSUFFICIENT PARAMETERS"

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 (nonzero if not)
ES:DI[C] pointer ; pointer to message string

April 15, 1990 Release R 1.0 6-7
syGetState

Last revision: R 1.0
Type: core

Parameters

<table>
<thead>
<tr>
<th>ASCII</th>
<th>Parameter</th>
<th>Core or extended</th>
<th>Associated calling value</th>
<th>Type as ASCII</th>
<th>Associated return value</th>
<th>Type as ASCII</th>
<th>Default if parameter not used</th>
</tr>
</thead>
<tbody>
<tr>
<td>lvver</td>
<td>Core</td>
<td>None</td>
<td>N/A</td>
<td>Recommended practices version number</td>
<td>Real</td>
<td>No action</td>
<td></td>
</tr>
<tr>
<td>mfgname</td>
<td>Core</td>
<td>None</td>
<td>N/A</td>
<td>MFG name¹</td>
<td>Text</td>
<td>No action</td>
<td></td>
</tr>
<tr>
<td>mfgver</td>
<td>Core</td>
<td>None</td>
<td>N/A</td>
<td>MFG version¹</td>
<td>Text</td>
<td>No action</td>
<td></td>
</tr>
<tr>
<td>support</td>
<td>Core</td>
<td>None</td>
<td>N/A</td>
<td>Value of bit field for installed service groups</td>
<td>Integer</td>
<td>No action</td>
<td></td>
</tr>
</tbody>
</table>

At least one parameter is required or an error is returned.

¹Eight characters max. Restricted to printable characters and cannot include white space (ASCII 20H, 09H), backspace (ASCII 08H), a comma (ASCII 2CH), CR (ASCII 0DH), or LF (ASCII 0AH).

Binary

Command code: 1030 decimal.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Core or extended</th>
<th>Token number (decimal)</th>
<th>Type</th>
<th>Associated calling value</th>
<th>Associated return value</th>
<th>Default if parameter not used</th>
</tr>
</thead>
<tbody>
<tr>
<td>lvver</td>
<td>Core</td>
<td>30</td>
<td>Real</td>
<td>Any value</td>
<td>Recommended practices version number</td>
<td>No action</td>
</tr>
<tr>
<td>mfgname</td>
<td>Core</td>
<td>33</td>
<td>Pointer</td>
<td>Any value</td>
<td>Pointer to MFG name string¹</td>
<td>No action</td>
</tr>
<tr>
<td>mfgver</td>
<td>Core</td>
<td>34</td>
<td>Pointer</td>
<td>Any value</td>
<td>Pointer to MFG version string¹</td>
<td>No action</td>
</tr>
<tr>
<td>support</td>
<td>Core</td>
<td>43</td>
<td>Bit field</td>
<td>Any value</td>
<td>Bit field of installed service groups</td>
<td>No action</td>
</tr>
</tbody>
</table>

At least one parameter is required or an error is returned.

¹Eight characters max. Restricted to printable characters and cannot include white space (ASCII 20H, 09H), backspace (ASCII 08H), a comma (ASCII 2CH), CR (ASCII 0DH), or LF (ASCII 0AH).
Section 6. System commands (sy)

syGetState

Description

Summary syGetState returns supported service groups for which VDI Management was configured at installation, the version of the recommended practices supported, and manufacturer name and version information.

Ivver parameter

The ivver parameter returns the version number of the recommended practices with which the VDI Management software complies. IV applications can use this number to determine compatibility with VDI Management implementations. An application that requires a given version number will also be compatible with any higher version number.

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 recommendations 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.

The mfgname and mfgver return strings are eight-character strings of printable characters. The strings cannot include white space (ASCII 20H, 09H), backspace (ASCII 08H), a comma (ASCII 2CH), CR (ASCII 0DH), or LF (ASCII 0AH).

Support parameter

The support parameter returns a bit field or an ASCII representation thereof that specifies service groups that are supported and 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. 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.

The following table shows return values for each service group after an 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 is not. An ASCII return value of "7" means the same.
syGetState

<table>
<thead>
<tr>
<th>Service Group</th>
<th>Binary Interface return value (low byte)</th>
<th>ASCII Interface return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>System (sy)</td>
<td>00000001</td>
<td>1</td>
</tr>
<tr>
<td>Visual management (vm)</td>
<td>00000010</td>
<td>2</td>
</tr>
<tr>
<td>Videodisc (vd)</td>
<td>00000100</td>
<td>4</td>
</tr>
<tr>
<td>XY input (xy)</td>
<td>00001000</td>
<td>8</td>
</tr>
</tbody>
</table>

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.

Notes

1. Values for mfgname and mfgver are not under the control of the recommended practices. (In the future, mfgname may be required to be unique and registered.)

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

Returns

ASCII

On success: Comma-separated string with response for each specified parameter as described above.

On failure: “ERROR n...”.

Binary

On success: AX = 0. Value associated with support parameter is a 32-bit bit field as described above. Value associated with ivver is a 32-bit real. Values associated with mfgname and mfgver parameters are pointers to null-terminated strings as described above.

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

See also:
syQueue, vdGetState, vdInit, vmGetState, vmInit, xyGetState, xyInit.
Section 6. System commands (sy)

syGetState

Examples

ASCII

Get services supported by installed system
syGetState support
(returns) "15" ; system, visual management, videodisc, and XY commands supported

Get version number of recommendations
syGetState ivver
(returns) "1.0" ; conforms with recommended practices
; standard, version 1.0

Get manufacturer and version
syGetState mfname,mfver
(returns) "IVMAKER,1.3" ; VDI Management written by IVMAKER,
; version 1.3

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 (nonzero if not)
ES:DI[4] bit field ; support parameter bit field
syInit

Parameters

ASCII

No parameters.

Binary

Command code: 1031 decimal.

No parameters.

Description

Summary

syInit initializes VDI Management and the sy service group and confirms communications between VDI Management and the application.

General discussion

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-IV operating modes, and do other basic start-up chores. syInit also does the following specific initialization tasks:

- Sets the default logical device or logical source for all service groups to zero.
- Clears the error buffers used by syCheckError.
- Issues syQueue clear, state=0 to clear the command queue and turn it off.

Notes

1. Application programs should make no assumptions about the state of the IV system or the presence of service groups after syInit. To determine which service groups are present and enable the groups, an application should:

a. Issue syInit.

b. Issue syGetState support to determine which services are present.

c. Issue the initialization commands for the service groups to be used by the application.
Section 6. System commands (sy)

syInit

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

2. 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 c and d above to ensure that the system is in a known state.

3. Trying to queue syInit causes error 177 (Command cannot be queued) at the time of the attempt.

Returns

**ASCII**

On success: “OK”.

On failure: “ERROR n...”.

**Binary**

On success: AX = 0.

On failure: AX = error number.

See also: syCheckError, syGetState, syQueue, vdInit, vmInit, xyInit.

Examples

**ASCII**

Initialize VDI Management

syInit (returns) “OK”

**Binary**

Initialize VDI Management

<table>
<thead>
<tr>
<th>AX</th>
<th>1031</th>
</tr>
</thead>
<tbody>
<tr>
<td>BX</td>
<td>0</td>
</tr>
</tbody>
</table>

; syInit decimal ID
; no parameters

After return

| AX | 0    |

; returns 0 if successful (nonzero if not)

April 15, 1990

Release R 1.0

6-13
**Recommended Practices for Interactive Video Portability**

**syQueue**

**Last revision:** R 1.0  
**Type:** core

### Parameters

#### ASCII

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Core or extended</th>
<th>Associated calling value</th>
<th>Type as ASCII</th>
<th>Associated return value</th>
<th>Type as ASCII</th>
<th>Default if parameter not used</th>
</tr>
</thead>
<tbody>
<tr>
<td>clear</td>
<td>Core</td>
<td>None</td>
<td>N/A</td>
<td>None</td>
<td>N/A</td>
<td>No action</td>
</tr>
<tr>
<td>execute</td>
<td>Core</td>
<td>None</td>
<td>N/A</td>
<td>None</td>
<td>N/A</td>
<td>No action</td>
</tr>
<tr>
<td>state</td>
<td>Core</td>
<td>1 (on)</td>
<td>0 (off)</td>
<td>Integer</td>
<td>None</td>
<td>N/A</td>
</tr>
</tbody>
</table>

At least one parameter is required or an error is returned.

#### Binary

**Command code:** 1034 decimal.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Core or extended</th>
<th>Token number (decimal)</th>
<th>Type</th>
<th>Associated calling value</th>
<th>Associated return value</th>
<th>Default if parameter not used</th>
</tr>
</thead>
<tbody>
<tr>
<td>clear</td>
<td>Core</td>
<td>8</td>
<td>N/A</td>
<td>Any value</td>
<td>None</td>
<td>No action</td>
</tr>
<tr>
<td>execute</td>
<td>Core</td>
<td>32</td>
<td>N/A</td>
<td>Any value</td>
<td>None</td>
<td>No action</td>
</tr>
<tr>
<td>state</td>
<td>Core</td>
<td>42</td>
<td>Integer</td>
<td>1 (on)</td>
<td>0 (off)</td>
<td>None</td>
</tr>
</tbody>
</table>

At least one parameter is required or an error is returned.

### Description

**Summary**

*syQueue* manages a fixed-length internal queue with exactly 10 slots for storing at most 10 commands. The queue can be turned on and off, cleared, and executed.

**General discussion**

*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.
Section 6. System commands (sy)

syQueue

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.

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 on or off state. A queue that has been turned off remains executable.

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

State parameter

The state parameter turns the queue on and off. State=1 instructs VDI Management to store at most 10 commands for later execution. If more than 10 commands are issued while the queue is on, the extra commands return error 176 (Queue full), and the commands in the queue are left intact.

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.

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

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.
Recommended Practices for Interactive Video Portability

**syQueue**

### Unqueueable commands

The following commands cannot be queued 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.

<table>
<thead>
<tr>
<th>Unqueueable commands</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>syCheckError</code></td>
</tr>
<tr>
<td><code>syStop</code></td>
</tr>
<tr>
<td><code>vmlnit</code></td>
</tr>
<tr>
<td><code>syErrorMsg</code></td>
</tr>
<tr>
<td><code>vdGetState</code></td>
</tr>
<tr>
<td><code>xyGetInput</code></td>
</tr>
<tr>
<td><code>syGetState</code></td>
</tr>
<tr>
<td><code>vdInit</code></td>
</tr>
<tr>
<td><code>xyGetState</code></td>
</tr>
<tr>
<td><code>sylinit</code></td>
</tr>
<tr>
<td><code>vmGetPalette</code></td>
</tr>
<tr>
<td><code>xyInit</code></td>
</tr>
<tr>
<td><code>syQueue</code></td>
</tr>
<tr>
<td><code>vmGetState</code></td>
</tr>
</tbody>
</table>

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. `syQueue` ignores the illegal command. It 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.

### Queued commands resulting in errors

If 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.

### Notes

1. `syQueue` always executes immediately and cannot be queued.

2. If `vmSetPalette` is queued, the pointer information in the command's parameter block is stored with the queue (see Section 7). 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 modified array will be used. An application should not deallocate the memory for the palette array before clearing the queue. Doing so could load invalid palette information.
Section 6. System commands (sy)

3. Trying to queue more than 10 commands causes error 176 (Queue full). This error can be returned for any queueable command. 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.

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

Returns

**ASCII**

On success: “OK”.

On failure: “ERROR n...”.

**Binary**

On success: \( AX = 0 \).

On failure: \( AX = \) error number.

See also: vmSetPalette.

Examples

**ASCII**

| Turn on queue and store commands | syQueue state=1               |
|                                  | (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
syQueue

**Execute queue, then unqueue commands**

- execute queue
  - (returns) “OK”
  - (command 8)
  - (returns) “OK”
  - (command 9)
  - (returns) “OK”

**Reexecute queue and clear**

- execute queue, clear
  - (returns) “OK”

**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 (nonzero 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 for “off”

**After return**

- AX  0; returns 0 if successful (nonzero if not)
Section 6. System commands (sy)

syStop

Parameters

ASCII
No parameters.

Binary
Command code: 1044 decimal.
No parameters.

Description

Summary
syStop frees all possible resources used by the interfaces and VDI Management to make the resources available for non-IV use.

General discussion
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-IV applications. The command's actions are highly implementation and configuration dependent. syStop does not change the graphics mode, therefore, applications must handle the mode after exit separately. However, if VDI Management turned mode trapping on, which it typically would, syStop turns it off.

After an syStop, all VDI commands are undefined except sylnit. Upon resumption of IV activities, an sylnit must be issued before any other IV command.

Notes
1. An attempt to queue syStop causes error 177 (Command cannot be queued) at the time of the attempt.

Returns

ASCII
On success: “OK”.
On failure: “ERROR n...”.

Binary
On success: AX = 0.
On failure: AX = error number.

See also:
syInit, syQueue.
Recommended Practices for Interactive Video Portability

syStop

Examples

ASCII

Place VDI Management in syStop (returns) "OK"
Inactive state

Binary

Place VDI AX 1044 ; syStop decimal ID
Management in BX 0 ; no parameters
Inactive state
After return AX 0 ; returns 0 if successful (nonzero if not)
7 Visual-management commands (vm)

This section 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 7–1 lists the commands covered in this section, their token numbers, and their types.

Table 7–1.
Visual-management command names, token numbers, and types

<table>
<thead>
<tr>
<th>ASCII command name</th>
<th>Binary interface token number (decimal)</th>
<th>Type²</th>
</tr>
</thead>
<tbody>
<tr>
<td>vmFade</td>
<td>2051</td>
<td>Core</td>
</tr>
<tr>
<td>vmGetPalette</td>
<td>2053</td>
<td>Core</td>
</tr>
<tr>
<td>vmGetState</td>
<td>2054</td>
<td>Core</td>
</tr>
<tr>
<td>vmlnit</td>
<td>2055</td>
<td>Core</td>
</tr>
<tr>
<td>vmSetGraphics</td>
<td>2062</td>
<td>Core</td>
</tr>
<tr>
<td>vmSetPalette</td>
<td>2063</td>
<td>Core</td>
</tr>
<tr>
<td>vmSetTrans</td>
<td>2064</td>
<td>Core</td>
</tr>
<tr>
<td>vmSetVideo</td>
<td>2065</td>
<td>Core</td>
</tr>
</tbody>
</table>

¹Upper or lower case for command names is not significant.
²Compliant implementations must support "Core" commands.

7.1 Terms of reference

Figure 7–1 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.
Recommended Practices for Interactive Video Portability

Figure 7-1. A simplified functional model of a video overlay subsystem

Notes:

1. 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.

2. 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).

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

4. The dissolve unit:
   a. In its simplest form is a switch between video only and graphics over video (a graphics ON/OFF capability).
   b. At the next level of complexity supports plane 0→plane 1 cross-fades.
   c. In future systems may become a pixel-rate translucency setting controlled by a palette extension (not shown).

7.2 General information and assumptions

The general information and assumptions given in this subsection were used in the definition of the visual-management commands. The material below is based on Intel 80x86 processor architecture, MS-DOS compatible operating systems, and standard IBM-compatible graphics modes.

7.2.1 Overlayable graphics modes

Appendix B lists IBM-compatible graphics modes as they would be returned by BIOS interrupt 10H, service OFH, including whether the modes are text or graphics, resolutions, the adapters that support them, and whether they are overlayable.
Compliance requires that graphics modes zero through three are guaranteed to be overlayable when the selected video mode is NTSC or PAL.

This means that when the video mode is not set to "native," these modes are restricted to 200 lines regardless of the actual number of lines that would normally be displayed by a given monitor adapter. (See the vmSetGraphics and vmSetVideo commands.)

### 7.2.2 Mode trapping

The visual-management commands do not control trapping interrupt 10H. VDI implementers 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 using the VDI visual-management commands. This is especially important for applications that may use separate graphics function libraries and similar tools.

### 7.2.3 Genlock control

The ability to turn genlock on and off is not included in the command set. The recommended practices assume that all video inputs and graphics are synchronous at all times from the application's viewpoint. Controlling genlocking is a video device driver issue that is left to the VDI implementer.

### 7.2.4 Graphics registration to the background video

Appendix A gives detailed information on assumed positions of graphics relative to background video. The information in the appendix should furnish reliable registration within about two pixels both horizontally and vertically.

However, applications that have critical registration requirements should include a position reference frame to allow dynamic positioning of the graphics plane at run-time. The command set provides commands for varying the graphics origin both horizontally and vertically. An extended feature supports setting the total width of the active graphics.

### 7.2.5 VGA graphics versus CGA and EGA graphics

Some graphics modes, in particular 620 x 200 and 320 x 200 resolution modes, are displayed differently by VGA adapters versus CGA and EGA adapters. 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. Therefore, for CGA and EGA applications to be portable to VGA-based systems, the VGA system...
must have the ability to display these modes at full width and as they would be displayed by a CGA or EGA adapter. Appendix A gives detailed information on VGA emulation of CGA and EGA graphics displays.

Compliance requires VGA emulation of CGA and EGA graphics displays.

### 7.2.6 Logical versus physical colors

The visual-management commands distinguish between logical and physical colors. A typical computer cannot 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 visual management 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—this is also commonly called the palette size.

### 7.3 Rounding methods for fades and dissolves

The visual management `vmFade` command must be supported. However, compliant VDI Management software can be developed for IV hardware without fade circuitry. For systems with fade circuitry, variations in available fade levels must be treated consistently.

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

VDI Management allocates each available hardware setting a level in the range 0–255 and rounds passed values to the nearest possible level. For example, if the hardware furnishes four fade levels with intensities of full off, ¼ on, ¾ on, and full on, these are allocated the numeric values 0, 85, 170, and 255, respectively. Passed values in the range 0–42 are rounded to full off, 43–127 to ¼ intensity, 128–212 to ¾ intensity, and 213–255 to full on.

If a fader is uneven, rounding ranges are adjusted accordingly. For example, if a fader can do full off, ½ on, ¾ on, and full on only, these are allocated the values 0, 128, 192, and 255.

For fades that use nonzero time periods, the fade levels are calculated as:

$$\text{current level} = \text{start level} + \frac{\text{time since start}}{\text{fade duration}} \times (\text{end level} - \text{start level})$$
The fade levels are then rounded in the usual way. Assume a fade from full off to full on over 2.55 seconds on a system that can fade to off, \( \frac{1}{3} \) on, \( \frac{2}{3} \) on, and full on. The fader stays off for the first 0.425 seconds, at \( \frac{1}{3} \) from 0.425 to 1.275 seconds, at \( \frac{2}{3} \) from 1.275 to 2.125 seconds, and at full on from 2.125 to 2.55 seconds. Note that noninteger fade levels round in the usual way.

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 levels 0–127 and full on for 128–255.

**Note:** Application authors should not assume rounded times are exact. On typical systems, the resolution of the tick interrupt or system clock will restrict the accuracy of timings.
Recommended Practices for Interactive Video Portability

vmFade

Last revision: R 1.0

Type: core

Parameters

**ASCII**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Core or extended</th>
<th>Associated calling value</th>
<th>Type as ASCII</th>
<th>Associated return value</th>
<th>Type as CII</th>
<th>Default if parameter not used</th>
</tr>
</thead>
<tbody>
<tr>
<td>dlevel</td>
<td>Core</td>
<td>Dissolve level, 0–255</td>
<td>Integer</td>
<td>None</td>
<td>N/A</td>
<td>No action</td>
</tr>
<tr>
<td>glevel</td>
<td>Core</td>
<td>Graphics level, 0–255</td>
<td>Integer</td>
<td>None</td>
<td>N/A</td>
<td>No action</td>
</tr>
<tr>
<td>vlevel</td>
<td>Core</td>
<td>Video level, 0–255</td>
<td>Integer</td>
<td>None</td>
<td>N/A</td>
<td>No action</td>
</tr>
<tr>
<td>time</td>
<td>Core</td>
<td>Seconds for fade or dissolve</td>
<td>Real</td>
<td>None</td>
<td>N/A</td>
<td>0</td>
</tr>
<tr>
<td>wait</td>
<td>Core</td>
<td>None</td>
<td>N/A</td>
<td>None</td>
<td>N/A</td>
<td>No wait</td>
</tr>
</tbody>
</table>

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

**Binary**

Command code: 2051 decimal.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Core or extended</th>
<th>Token number (decimal)</th>
<th>Type</th>
<th>Associated calling value</th>
<th>Associated return value</th>
<th>Default if parameter not used</th>
</tr>
</thead>
<tbody>
<tr>
<td>dlevel</td>
<td>Core</td>
<td>17</td>
<td>Integer</td>
<td>Dissolve level, 0–255</td>
<td>Actual level after rounding if required</td>
<td>No action</td>
</tr>
<tr>
<td>glevel</td>
<td>Core</td>
<td>26</td>
<td>Integer</td>
<td>Graphics level, 0–255</td>
<td>Actual level after rounding if required</td>
<td>No action</td>
</tr>
<tr>
<td>vlevel</td>
<td>Core</td>
<td>52</td>
<td>Integer</td>
<td>Video level, 0–255</td>
<td>Actual level after rounding if required</td>
<td>No action</td>
</tr>
<tr>
<td>time</td>
<td>Core</td>
<td>47</td>
<td>Real</td>
<td>Seconds for fade or dissolve</td>
<td>None</td>
<td>0</td>
</tr>
<tr>
<td>wait</td>
<td>Core</td>
<td>54</td>
<td>N/A</td>
<td>None</td>
<td>None</td>
<td>No wait</td>
</tr>
</tbody>
</table>

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

1See Section 4.4.2 for the hexadecimal representation of this value.
Section 7. Visual-management commands (vm)

vmFade

Description

Summary

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 after the specified level is reached if wait is specified. (Figure 7-1 in Section 7.1 shows the relationship of the level parameters to a video overlay subsystem.)

General discussion

Actual levels 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. vmGetState returns the actual level that was set after any required rounding when vmFade has finished execution. Similarly, the binary version of vmFade returns the actual level that will be set after any rounding in the parameter packet. (See Section 7.3 for more information on rounding levels.)

Dlevel parameter

The dlevel parameter creates transitions or dissolves between video only and hard keying (transparent colors are full video, opaque colors are 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 dlevel is 0-127 and to hard keying when dlevel is 128-255.

With dlevel=0 the video plane only is visible. With dlevel=255 nontransparent colors display graphics at full intensity and transparent colors display video only, assuming vmSetTrans state=1 has been issued. Assuming dlevel=255, to create a transition from graphics only to video only, turn transparency off with vmSetTrans state=0, then issue vmFade dlevel=0.

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.

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.
**vmFade**

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

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

**Time parameter**  The *time* parameter specifies the number of seconds over which the fade or dissolve occurs. If necessary, VDI Management rounds time to the nearest value the system supports.

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

**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 task.

The *wait* parameter functions the same even if the system does not support fades or dissolves.

**Notes**

1. *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.

2. 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, timer=60, wait
   ```

the system will remain at level 0 for 30 seconds, then switch to level 255, then return after another 30 seconds. (See Section 7.3 for more information on rounding times.)
Section 7. Visual-management commands (vm)

vmFade

3. 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.

Returns

ASCII

On success: “OK”.

On failure: “ERROR n...”.

Binary

On success: AX = 0. Value associated with dlevel, glevel, or vlevel parameter is a 32-bit integer that gives the actual level that will be set after rounding if required.

On failure: AX = error number.

See also:

syQueue, vmGetState, vmSetTrans.

Examples

ASCII

Dissolve to all video over 1.5 seconds, do not return until complete

vmFade dlevel=0, time=1.5, wait
(returns) “0”

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 0.3 seconds

vmFade glevel=255, time=.3
(returns) “255”

Fade video to zero over 1 second, do not return until complete

vmFade vlevel=0, time=1, wait
(returns) “0”

Set video to zero immediately

vmFade vlevel=0
(returns) “OK”
vmFade

Set video to half intensity immediately

vmFade vlevel=128

(returns) "OK"

Binary

Dissolve to 0 over 2.5 seconds and return when done:

AX 2051 ; vmFade decimal ID
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[10] 2.5 ; time value in seconds. See Section 4.4.2 for the hexadecimal representation of this value.
ES:DI[10] 54 ; wait decimal ID
ES:DI[14] any value ; no associated value

After return:

AX 0 ; returns 0 if successful (nonzero if not)
ES:DI[4] 0 ; actual level that will be set after rounding if required
### vmGetPalette

**Last revision:** R 1.0

**Type:** core

#### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Core or extended</th>
<th>Associated calling value</th>
<th>Type as ASCII</th>
<th>Associated return value</th>
<th>Type as ASCII</th>
<th>Default if parameter not used</th>
</tr>
</thead>
<tbody>
<tr>
<td>color</td>
<td>Core</td>
<td>Logical color number</td>
<td>Integer</td>
<td>None</td>
<td>N/A</td>
<td>Causes error</td>
</tr>
<tr>
<td>r</td>
<td>Core</td>
<td>None</td>
<td>N/A</td>
<td>Red value, 0–255</td>
<td>Integer</td>
<td>No action</td>
</tr>
<tr>
<td>g</td>
<td>Core</td>
<td>None</td>
<td>N/A</td>
<td>Green value, 0–255</td>
<td>Integer</td>
<td>No action</td>
</tr>
<tr>
<td>b</td>
<td>Core</td>
<td>None</td>
<td>N/A</td>
<td>Blue value, 0–255</td>
<td>Integer</td>
<td>No action</td>
</tr>
</tbody>
</table>

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

### Binary

**Command code:** 2053 decimal

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Core or extended</th>
<th>Token number (decimal)</th>
<th>Type</th>
<th>Associated calling value</th>
<th>Associated return value</th>
<th>Default if parameter not used</th>
</tr>
</thead>
<tbody>
<tr>
<td>color</td>
<td>Core</td>
<td>34</td>
<td>Integer</td>
<td>Logical color number</td>
<td>None</td>
<td>Causes error</td>
</tr>
<tr>
<td>r</td>
<td>Core</td>
<td>38</td>
<td>Integer</td>
<td>Any value</td>
<td>Red value, 0–255</td>
<td>No action</td>
</tr>
<tr>
<td>g</td>
<td>Core</td>
<td>25</td>
<td>Integer</td>
<td>Any value</td>
<td>Green value, 0–255</td>
<td>No action</td>
</tr>
<tr>
<td>b</td>
<td>Core</td>
<td>4</td>
<td>Integer</td>
<td>Any value</td>
<td>Blue value, 0–255</td>
<td>No action</td>
</tr>
<tr>
<td>length</td>
<td>Core</td>
<td>31</td>
<td>Integer</td>
<td>Number of color array entries(^1)</td>
<td>Number of color array entries(^1)</td>
<td>No action</td>
</tr>
<tr>
<td>array</td>
<td>Core</td>
<td>1</td>
<td>Pointer</td>
<td>Pointer to color array</td>
<td>Pointer to color array</td>
<td>No action</td>
</tr>
</tbody>
</table>

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.

\(^1\)color array entry = 4 bytes comprised of 3 components + reserved byte.
Recommended Practices for Interactive Video Portability

vmGetPalette

Description

Summary
vmGetPalette returns the amounts of red, green, and blue components in a
specified logical color via the ASCII interface or one or more sets of compo-
nent values for contiguous logical colors via the binary interface.

General
discussion
vmGetPalette returns the proportions of the red, green, and blue compo-
nents in a logical color as real numbers in the range 0–255 where 255 is fully
saturated for each component. Component values are returned for single col-
ors via the ASCII interface and for single or multiple colors via the binary
interface depending on the calling parameters.

vmGetState logcolors returns the number of available logical colors (colors
that can be simultaneously displayed). vmGetState physcolors returns the
number of available physical colors (palette size). vmSetPalette assigns
physical colors to logical colors, and vmGetPalette returns the component
values for the assigned colors. For example, a system might support 16 logi-
cal 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.

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 re-
turned by vmGetState logcolors minus one.

Exactly one color parameter must be listed. Specifying color twice causes er-
ror 54 (Parameter used more than once), and omitting color or failing to in-
clude at least one of r, g, and b causes error 49 (Insufficient parameters).

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.

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 pal-
ette values in application memory. The value associated with array is a 32-
bit pointer to a memory block containing one or more 4-byte structures.
Array must point to memory allocated by the application. The contents of
each structure in the palette array are:

- Byte 0, the least significant byte, represents blue.
- Byte 1 represents green.
- Byte 2 represents red.
- Byte 3 is reserved and always set to zero.
vmGetPalette

The length parameter specifies the number of 4-byte structures in the array. The values in the first structure of the array are for the logical color number specified by the color parameter. The second structure relates to color+1, the third to color+2, and so on up to the number of structures specified by length. (See Section 4.4.2 for more information about palette arrays.)

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.

Notes

1. Values returned by vmGetPalette may not exactly match values set with vmSetPalette because of rounding 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.

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

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

Returns

ASCII

On success: Comma-separated list of requested r, g, and b component values in the range “0” to “255” for color.

On Failure: “ERROR n...”.

Binary

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

See also: syQueue, vmGetState, vmSetPalette.
vmGetPalette

Examples

ASCII

Return values for all parameters for logical color 0

vmGetPalette color=0,r,g,b
  (returns) "255,127,0" ; logical color 0 has red=255, green=127, blue=0

Return red component for logical color 1

vmGetPalette color=1,r
  (returns) "170" ; logical color 1 has a red component of 170

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 (nonzero if not) |
| ES:DI[C] | g value | ; green value for color 3 |

Get color values from color 3 to 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 (nonzero if not) |
| ES:DI[14] pointer | ; pointer to same array with updated component values |
**vmGetState**

**Parameters**

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

At least one parameter is required or an error is returned.
# Recommended Practices for Interactive Video Portability

## vmGetState

**Binary**

**Command code:** 2054 decimal.

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

At least one parameter is required or an error is returned.

1 See Section 4.4.2 for the hexadecimal representation of this value.
Section 7. Visual-management commands (vm)

vmGetState

Description

Summary

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.

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).

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=on. 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.

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 video source is defined as source zero unless vmSetVideo defsource is used to change it.

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

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

Emulation parameter

The emulation parameter returns the state of VGA emulation of CGA and EGA graphics versus VGA native mode. In some graphics modes, CGA and EGA graphics displayed with a CGA or EGA adapter have different horizontal registration relative to video compared to graphics with the same mode number displayed with a VGA adapter. Implementations that support CGA or EGA graphics only always return one (on). (See Appendix A for more information on VGA emulation of CGA and EGA graphics.)
vmGetState

Gmode parameter  The gmode parameter returns the current video mode exactly as it would be returned by a request to BIOS interrupt 10H, service 0FH (see Appendix B).

Horzpix and vertpix parameters  The horzpix and vertpix parameters return the current pixel resolution. 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, MCGA, VGA).

Logcolors and physcolors parameters  The logcolors parameter returns the number of logical colors (simultaneously displayable colors) that are available. The physcolors parameter returns the range of colors (palette size) that can be assigned to logical colors. Both return values are determined by the capabilities of the graphics hardware and mode.

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

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

Vmode parameter  The vmode parameter returns the video mode as set by vmSetVideo. The vmode is either 0 (native), 1 (NTSC), or 2 (PAL). Native mode is a nonoverlay mode, but does not change overlay parameters. NTSC and PAL indicate the system is configured for the indicated video standard.

Width parameter  The width parameter returns the total graphics width in microseconds. This parameter lets applications accurately establish the right edge of the active graphics area relative to background video. (See Appendix A for more information on graphics registration.)

Width is an extended parameter. Using an unimplemented extended parameter causes error 48 (Unknown parameter).
Section 7. Visual-management commands (vm)

vmGetState

Xoffset and yoffset parameters

The xoffset and yoffset give the offset of the graphics plane relative to the video plane in pixels as set by vmSetGraphics. The origin of the graphics plane is the upper left corner of the graphics display area.

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.

Notes

1. Trying to queue vmGetState causes error 177 (Command cannot be queued) at the time of the attempt.

Returns

ASCII

On success: Comma-separated list of values for requested parameters as described above.

On failure: “ERROR n...”.

Binary

On success: AX = 0. Values associated with requested parameters are 32-bit values of the types given in the binary parameter table above.

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

See also:


Examples

ASCII

Get the current graphics level and mode

vmGetState glevel,gmode

(returns) “0,14” ; graphics level currently set to 0

; graphics mode is 14 (640 X 200)

Determine whether logical color three is transparent

vmGetState color=3

(returns) “1” ; color number 3 is set for transparency
Recommended Practices for Interactive Video Portability

vmGetState

**Binary**

<table>
<thead>
<tr>
<th>Get current</th>
<th>AX 2054</th>
<th>; vmGetState decimal ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>graphics level,</td>
<td>BX 3</td>
<td>; number of parameters</td>
</tr>
<tr>
<td>graphics mode,</td>
<td>ES:DI[0] 26</td>
<td>; glevel decimal ID</td>
</tr>
<tr>
<td>and number of</td>
<td>ES:DI[4] any value</td>
<td>; place holder for value on return</td>
</tr>
<tr>
<td>available physical</td>
<td>ES:DI[8] 27</td>
<td>; gmode decimal ID</td>
</tr>
<tr>
<td>colors</td>
<td>ES:DI[C] any value</td>
<td>; place holder for value on return</td>
</tr>
<tr>
<td></td>
<td>ES:DI[10] 36</td>
<td>; physcolors decimal ID</td>
</tr>
<tr>
<td></td>
<td>ES:DI[14] any value</td>
<td>; place holder for value on return</td>
</tr>
</tbody>
</table>

**After return**

<table>
<thead>
<tr>
<th>AX 0</th>
<th>; returns 0 if successful (nonzero if not)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ES:DI[4] level value</td>
<td>; contains current graphics level</td>
</tr>
<tr>
<td>ES:DI[C] gmode value</td>
<td>; contains current graphics mode number</td>
</tr>
<tr>
<td>ES:DI[14] physcolors value</td>
<td>; contains available physical colors</td>
</tr>
</tbody>
</table>
Section 7. Visual-management commands (vm)

vmlInit

Parameters

ASCII
No parameters.

Binary
Command code: 2055 decimal.
No parameters.

Description

Summary
vmlInit initializes the visual management hardware and software, placing both in a known state.

Conditions set by vmlInit
vmlInit sets the parameters in the following table to the specified values.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Command reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>dlevel</td>
<td>255 (hard keying)</td>
<td>vmFade</td>
</tr>
<tr>
<td>emulation</td>
<td>1 (on)</td>
<td>vmSetGraphics</td>
</tr>
<tr>
<td>enable</td>
<td>0 (off)</td>
<td>vmSetTrans</td>
</tr>
<tr>
<td>glevel</td>
<td>255 (full intensity)</td>
<td>vmFade</td>
</tr>
<tr>
<td>gmode</td>
<td>Current value</td>
<td>vmSetGraphics</td>
</tr>
<tr>
<td>horzpix</td>
<td>Current value</td>
<td>None</td>
</tr>
<tr>
<td>logcolors</td>
<td>System limit for gmode</td>
<td>None</td>
</tr>
<tr>
<td>physcolors</td>
<td>System limit for gmode</td>
<td>None</td>
</tr>
<tr>
<td>state</td>
<td>0 (transparency off)</td>
<td>vmSetTrans</td>
</tr>
<tr>
<td>transcolors</td>
<td>System limit for gmode</td>
<td>None</td>
</tr>
<tr>
<td>width¹</td>
<td>System default for gmode</td>
<td>vmSetGraphics</td>
</tr>
<tr>
<td>vertpix</td>
<td>Current value</td>
<td>None</td>
</tr>
<tr>
<td>vlevel</td>
<td>0 (off)</td>
<td>vmFade</td>
</tr>
<tr>
<td>vmode</td>
<td>0 (native)</td>
<td>vmSetVideo</td>
</tr>
<tr>
<td>xoffset</td>
<td>0</td>
<td>vmSetGraphics</td>
</tr>
<tr>
<td>yoffset</td>
<td>0</td>
<td>vmSetGraphics</td>
</tr>
</tbody>
</table>

¹If supported.
**vmInit**

**Notes**

1. In typical VDI Management implementations, vmInit turns mode trapping on to intercept video BIOS interrupt 10H calls so that applications cannot make 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.

2. 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.

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

**Returns**

**ASCII**

On success: “OK”.

On failure: “ERROR n...”.

**Bin:**

On success: AX = 0.

On failure: AX = error number.

**See also:** syInit, syStop, vdInit, xyInit.

**Examples**

**ASCII**

Initialize visual management services vmInit (returns) “OK”

**Binary examples**

<table>
<thead>
<tr>
<th>Initialize visual management services</th>
<th>AX</th>
<th>2055</th>
<th>; vmInit decimal ID</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>BX</td>
<td>0</td>
<td>; number of parameters</td>
</tr>
<tr>
<td>After return</td>
<td>AX</td>
<td>0</td>
<td>; returns 0 if successful (nonzero if not)</td>
</tr>
</tbody>
</table>
vmSetGraphics

**Emulation parameter**

The `emulation` parameter controls VGA emulation of CGA and EGA horizontal graphics positioning in common modes. 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. (See Appendix A for more information on graphics registration and VGA emulation of CGA and EGA graphics.)

Issuing `vmSetGraphics emulation=0` on a true CGA or EGA-based system returns error 194 (Unsupported graphics mode).

**Gmode parameter**

The `gmode` parameter sets the graphics display mode in accordance with IBM graphics mode numbers as returned by BIOS interrupt 10H, service 0FH (see Appendix B). 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).

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

**Width parameter**

The `width` parameter sets the total graphics width in microseconds. This parameter lets applications accurately establish the right-hand edge of the active graphics area relative to background video. An application would typically display a video disc position-reference frame for interactively setting `width`. (See Appendix A for more information on graphics registration.)

`Width` is an extended parameter. Using an unimplemented extended parameter causes error 48 (Unknown parameter).

**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. (See Appendix A for default offsets by graphics mode.)

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.

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.
Section 7. Visual-management commands (vm)

vmSetGraphics

Notes

1. `vmGetState` returns the current 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.

2. 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.

Returns

ASCII

On success: "OK".

On failure: "ERROR n...".

Binary

On success: AX = 0.

On failure: AX = error number.

See also:

`vmGetState`, `vmInit`.

Examples

ASCII

Set X and Y offsets to -1 and 5

```plaintext
vmSetGraphics xoffset=-1, yoffset=5
```

(returns) "OK"

Set graphics mode 16 (EGA 640x350)

```plaintext
vmSetGraphics gmode=16
```

(returns) "OK"

Binary

Shift graphics right one and down two

```plaintext
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[16] 2 ; pixel offset value
```

After return

```plaintext
AX 0 ; returns 0 if successful (nonzero if not)
```
Recommended Practices for Interactive Video Portability

vmSetPalette

Last revision: R 1.0

Type: core

Parameters

ASCII

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Core or extended</th>
<th>Associated calling value</th>
<th>Type as ASCII</th>
<th>Associated return value</th>
<th>Type as ASCII</th>
<th>Default if parameter not used</th>
</tr>
</thead>
<tbody>
<tr>
<td>color</td>
<td>Core</td>
<td>Logical color number</td>
<td>Integer</td>
<td>None</td>
<td>N/A</td>
<td>Causes error</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>r</td>
<td>Core</td>
<td>Red value, 0–255</td>
<td>Integer</td>
<td>None</td>
<td>N/A</td>
<td>No action</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>g</td>
<td>Core</td>
<td>Green value, 0–255</td>
<td>Integer</td>
<td>None</td>
<td>N/A</td>
<td>No action</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>Core</td>
<td>Blue value, 0–255</td>
<td>Integer</td>
<td>None</td>
<td>N/A</td>
<td>No action</td>
</tr>
</tbody>
</table>

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

Binary

Command code: 2063 decimal.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Core or extended</th>
<th>Token number (decimal)</th>
<th>Type</th>
<th>Associated calling value</th>
<th>Associated return value</th>
<th>Default if parameter not used</th>
</tr>
</thead>
<tbody>
<tr>
<td>color</td>
<td>Core</td>
<td>9</td>
<td>Integer</td>
<td>Logical color number</td>
<td>None</td>
<td>Causes error</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>r</td>
<td>Core</td>
<td>38</td>
<td>Integer</td>
<td>Red value, 0–255</td>
<td>None</td>
<td>No action</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>g</td>
<td>Core</td>
<td>25</td>
<td>Integer</td>
<td>Green value, 0–255</td>
<td>None</td>
<td>No action</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>Core</td>
<td>4</td>
<td>Integer</td>
<td>Blue value, 0–255</td>
<td>None</td>
<td>No action</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>length</td>
<td>Core</td>
<td>31</td>
<td>Integer</td>
<td>Number of color array entries</td>
<td>None</td>
<td>No action</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>array</td>
<td>Core</td>
<td>1</td>
<td>Pointer</td>
<td>Pointer to color array</td>
<td>None</td>
<td>No action</td>
</tr>
</tbody>
</table>

 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.

1 color array entry = 4 bytes comprised of 3 components + reserved byte.
Section 7. Visual-management commands (vm)

vmSetPalette

Description

Summary

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.

General discussion

vmSetPalette sets the proportions of the red, green, and blue components in a logical color as real numbers 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.

vmGetState logcolors returns the number of available logical colors (colors that can be simultaneously displayed) for a system and vmGetState physcolors returns the number of available physical colors (palette size). 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.

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.

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.

Exactly one color parameter must be listed. 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). Any or all of r, g, and b can be specified in the same call.
vmSetPalette

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. The value associated with array is a 32-bit pointer to a memory block containing one or more 4-byte structures. Array must point to memory allocated by the application. The contents of each structure in the palette array are:

- Byte 0, the least significant byte, represents blue.
- Byte 1 represents green.
- Byte 2 represents red.
- Byte 3 is reserved and must be set to zero.

The length parameter specifies the number of 4-byte structures in the array. The values in the first structure of the array are for the logical color specified by the color parameter. The second structure relates to color+1, the third to color+2, and so on up to the number of structures specified by length. (See Section 4.4.2 for more information about color arrays.)

Using the length and array with any of r, g, or b causes error 50 (Parameters cannot be used together).

Notes

1. Use syQueue to set multiple logical colors in the same vertical interval via the ASCII interface.
2. 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.
3. 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 \times 4.
4. If a vmSetPalette is used with palette array and queued with syQueue, do not deallocate the array before executing the queue.

Returns

ASCII

On success: "OK".

On failure: "ERROR n...".

Binary

On success: AX = 0.

On failure: AX = error number.
Section 7. Visual-management commands (vm)

**vmSetPalette**

**See also:** syQueue, vmGetPalette, vmGetState.

**Examples**

**ASCII**

Set red to 63 for color 0, do not change other components

```
vmSetPalette color=0,r=63
```

(returns) "OK"

Set color 1 to pure 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"

**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 (nonzero 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 0 if successful (nonzero if not)
```
vmSetTrans

Last revision: R 1.0
Type: core

Parameters

ASCII

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Core or extended</th>
<th>Associated calling value</th>
<th>Type as ASCII</th>
<th>Associated return value</th>
<th>Type as ASCII</th>
<th>Default if parameter not used</th>
</tr>
</thead>
<tbody>
<tr>
<td>clear</td>
<td>Core</td>
<td>None</td>
<td>N/A</td>
<td>None</td>
<td>N/A</td>
<td>No action</td>
</tr>
<tr>
<td>color</td>
<td>Core</td>
<td>Logical color number</td>
<td>Integer</td>
<td>None</td>
<td>N/A</td>
<td>No action</td>
</tr>
<tr>
<td>enable</td>
<td>Core</td>
<td>1 (on)</td>
<td>0 (off)</td>
<td>Integer</td>
<td>None</td>
<td>N/A</td>
</tr>
<tr>
<td>state</td>
<td>Core</td>
<td>1 (on)</td>
<td>0 (off)</td>
<td>Integer</td>
<td>None</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Either both color and state, or clear only must be used or an error is returned. Enable can be used alone or with a color + state.

Binary

Command code: 2064 decimal.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Core or extended</th>
<th>Token number (decimal)</th>
<th>Type</th>
<th>Associated calling value</th>
<th>Associated return value</th>
<th>Default if parameter not used</th>
</tr>
</thead>
<tbody>
<tr>
<td>clear</td>
<td>Core</td>
<td>8</td>
<td>N/A</td>
<td>None</td>
<td>None</td>
<td>No action</td>
</tr>
<tr>
<td>color</td>
<td>Core</td>
<td>9</td>
<td>Integer</td>
<td>Logical color number</td>
<td>None</td>
<td>No action</td>
</tr>
<tr>
<td>enable</td>
<td>Core</td>
<td>20</td>
<td>Integer</td>
<td>1 (on)</td>
<td>0 (off)</td>
<td>None</td>
</tr>
<tr>
<td>state</td>
<td>Core</td>
<td>42</td>
<td>Integer</td>
<td>1 (on)</td>
<td>0 (off)</td>
<td>None</td>
</tr>
</tbody>
</table>

Either both color and state, or clear only must be used or an error is returned. Enable can be used alone or with a color + state.

Description

Summary

vmSetTrans sets logical colors to transparent or opaque and turns physical transparency on and off.

Clear

The clear parameter sets the transparency state of all logical colors to zero (off). Note that this not only turns transparency off but also changes the values of color attributes. Use the enable parameter (see below) to turn transparency off without changing the transparency settings of the colors.
Section 7. Visual-management commands (vm)

vmSetTrans

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).

**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 the **enable** parameter (see below).

**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.

**vmSetTrans enable=0** makes all colors physically opaque regardless of their transparency settings. None of the video plane is visible. However, transparent colors keep their transparency settings and will again be physically transparent after a subsequent **vmSetTrans enable=1**.

**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).

**Notes**

1. **vmGetState logcolors** returns the total number of available logical colors. **vmGetState color** returns the transparency setting for a single specified color.

2. **vmGetState transcolors** returns the number of logical colors that can be made transparent. Using **vmSetTrans** to try to set more than **transcolors** to transparent returns error 51 (Parameter invalid or out of range).

Compliant systems must 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.
Recommended Practices for Interactive Video Portability

vmSetTrans

Returns

ASCII

On success: “OK”.

On failure: “ERROR n...”.

Binary

On success: AX = 0.

On failure: AX = error number.

See also: vmGetState.

Examples

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”

Binary

Designate color 3 as transparent and make it physically transparent
AX 2064 ; vmSetTrans decimal ID
  ES:DI[0] 9 ; color decimal ID
  ES:DI[4] 3 ; color number
  ES:DI[8] 42 ; state decimal ID
  ES:DI[10] 20 ; enable decimal idea
  ES:DI[14] 1 ; turn physical transparency on

After return
AX 0 ; returns 0 if successful (nonzero if not)

Make all colors opaque
AX 2064 ; vmSetTrans decimal ID
  BX 1 ; number of parameters
  ES:DI[0] 8 ; clear decimal ID

After return
AX 0 ; returns 0 if successful (nonzero if not)
Section 7. Visual-management commands (vm)

vmSetVideo

Current: R 1.0
Type: core

Parameters

### ASCII

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Core or extended</th>
<th>Associated calling value</th>
<th>Type as ASCII</th>
<th>Associated return value</th>
<th>Type as ASCII</th>
<th>Default If parameter not used</th>
</tr>
</thead>
<tbody>
<tr>
<td>defsource</td>
<td>Core</td>
<td>Input source, 0–15</td>
<td>Integer</td>
<td>None</td>
<td>N/A</td>
<td>No action</td>
</tr>
<tr>
<td>vmode</td>
<td>Core</td>
<td>0 (native)</td>
<td>1 (NTSC)</td>
<td>2 (PAL)</td>
<td>Integer</td>
<td>N/A</td>
</tr>
</tbody>
</table>

At least one parameter is required or an error is returned.

### Binary

Command code: 2065 decimal.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Core or extended</th>
<th>Token number (decimal)</th>
<th>Type</th>
<th>Associated calling value</th>
<th>Associated return value</th>
<th>Default If parameter not used</th>
</tr>
</thead>
<tbody>
<tr>
<td>defsource</td>
<td>Core</td>
<td>13</td>
<td>Integer</td>
<td>Input source, 0–15</td>
<td>None</td>
<td>No action</td>
</tr>
<tr>
<td>vmode</td>
<td>Core</td>
<td>53</td>
<td>Integer</td>
<td>0 (native)</td>
<td>1 (NTSC)</td>
<td>2 (PAL)</td>
</tr>
</tbody>
</table>

At least one parameter is required or an error is returned.

Description

Summary

vmSetVideo sets the video mode and selects the video input source if more than one source is available.

Defsource parameter

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

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. vmode=1 sets NTSC-U.S.; vmode=2 sets PAL-European. vmSetVideo vmode=0 (native) sets the system to the functionality and appearance that the computer would use if it were not an IV system. This setting also turns overlay off without affecting any other parameters relating to overlay.
vmSetVideo

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

Notes

1. 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 installation/configuration time. (See Section 4.1.2 for more information.)

2. After a player is selected with `vdSet defdevice` (see Section 8), it must be activated as a video source with `vmSetVideo defsource`.

Returns

ASCII

On success: “OK”.

On failure: “ERROR n...”.

Binary

On success: AX = 0.

On failure: AX = error number.

See also: `vdSet`, `vmGetState`.

Examples

ASCII

Set the standard to NTSC  

`vmSetVideo vmode=1  

(returns) "OK"`

Make video source one the default  

`vmSetVideo defsource=1  

(returns) "OK"`

Binary

Set mode to NTSC  

<table>
<thead>
<tr>
<th>AX</th>
<th>2065</th>
<th>; <code>vmSetVideo</code> decimal command ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>BX</td>
<td>1</td>
<td>; number of parameters</td>
</tr>
<tr>
<td>ES:DI[0]</td>
<td>53</td>
<td>; <code>vmode</code> decimal ID</td>
</tr>
<tr>
<td>ES:DI[4]</td>
<td>1</td>
<td>; sets mode to NTSC (1)</td>
</tr>
</tbody>
</table>

After return  

| AX | 0    | ; returns 0 if successful (nonzero if not) |

7-34 Release R 1.0 April 15, 1990
8 Videodisc commands (vd)

This section 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 8–1 lists the commands covered in this section, their token numbers, and their types.

<table>
<thead>
<tr>
<th>ASCII command name</th>
<th>Binary Interface token number (decimal)</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>vdGetState</td>
<td>3078</td>
<td>Core</td>
</tr>
<tr>
<td>vdInit</td>
<td>3079</td>
<td>Core</td>
</tr>
<tr>
<td>vdPassThru</td>
<td>3080</td>
<td>Core</td>
</tr>
<tr>
<td>vdPlay</td>
<td>3081</td>
<td>Core</td>
</tr>
<tr>
<td>vdScan</td>
<td>3083</td>
<td>Core</td>
</tr>
<tr>
<td>vdSearch</td>
<td>3084</td>
<td>Core</td>
</tr>
<tr>
<td>vdSet</td>
<td>3085</td>
<td>Core</td>
</tr>
<tr>
<td>vdStep</td>
<td>3090</td>
<td>Core</td>
</tr>
<tr>
<td>vdStill</td>
<td>3091</td>
<td>Core</td>
</tr>
</tbody>
</table>

1 Upper or lower case for command names is not significant.
2 Compliant implementations must support "Core" commands.

8.1 General information and assumptions

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

8.1.1 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. The individual
command descriptions indicate commands and parameters that apply to CLV videodiscs only.

Current support for CLV videodiscs is a subset of CAV functions. We do not rule out adding extended commands and parameters to provide more sophisticated CLV support in future revisions of this document.

Compliant players must support both CAV and CLV videodiscs.

8.1.2 Play and scan speeds

Play speeds are expressed as a multiples of 1.0, which is defined as the normal speed of either 25 frames per second for PAL or 30 frames per second for NTSC. For players in CAV mode, applications can assume that speed 1.0, at least one speed slower than 1.0, and at least one speed faster than 1.0 are available. For players in CLV mode, applications cannot assume play speeds other than 1.0. (Section 8.2 explains how VDI Managements round speeds when requested speeds cannot be matched exactly by a player.)

Scan speeds vary among players. All an application can assume about a scan speed is that it is faster than normal speed.

Note: Because applications cannot assume that values other than 1.0 will be matched exactly, they should not try to calculate videodisc position based on timing and frame speed at any speed other than 1.0. Instead, they should use vdGetState frame (see vdGetState command).

8.1.3 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.

8.1.4 Fields, frames, and chapters

All frame numbers assume a standard format of two fields per frame. The command set does not support accessing individual fields. The command set assumes that frame numbers are always available from CAV and never from CLV and that chapter numbers may be available from either.
8.2 Rounding methods for player speeds

Player speeds are represented by real numbers, with 1.0 representing normal speed. Values below 1.0 represent speeds below normal; values above 1.0 represent speeds above normal. A value other than 1.0 calls for a speed in frames per second (fps) that equals the product of the value and the default number of frames per second. For example, on an NTSC system, a speed of 0.5 specifies a rate of $0.5 \times 30$ fps or 15 fps.

Videodisc players are usually limited to a finite range of speeds. If a requested speed is not 1.0, 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 0.0 or 1.0 except for players in CLV mode (see below). This method lets applications guarantee use of the fastest fast and slowest slow speeds available. The following tables show how speeds are rounded for the Sony 2000 and Pioneer 4200, respectively. Note that speed requests of 0.0 are errors.

<table>
<thead>
<tr>
<th>Requested speed</th>
<th>Actual speed as multiple of normal</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>Error</td>
</tr>
<tr>
<td>0.00001–0.99999</td>
<td>0.2</td>
</tr>
<tr>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>1.00001 and up</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Requested speed</th>
<th>Actual speed as multiple of normal</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>Error</td>
</tr>
<tr>
<td>0.00001–0.14999</td>
<td>0.1</td>
</tr>
<tr>
<td>0.15–0.34999</td>
<td>0.2</td>
</tr>
<tr>
<td>0.35–0.64999</td>
<td>0.5</td>
</tr>
<tr>
<td>0.65–0.99999</td>
<td>0.8</td>
</tr>
<tr>
<td>1.0</td>
<td>1</td>
</tr>
<tr>
<td>1.0001–2.4999</td>
<td>2</td>
</tr>
<tr>
<td>2.5–3.4999</td>
<td>3</td>
</tr>
<tr>
<td>3.5 and up</td>
<td>4</td>
</tr>
</tbody>
</table>

For players in CAV mode, applications can assume that normal speed, 1.0, at least one speed slower than 1.0, and at least one speed faster than 1.0 are
available. Given this availability and the rounding algorithm, which never rounds to 0.0 or 1.0, the following table lists speed parameter values for specifying several convenient speeds without knowing the exact speeds available from a given player.

<table>
<thead>
<tr>
<th>Speed parameter value</th>
<th>Resulting player speed</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>Error</td>
</tr>
<tr>
<td>0.00001</td>
<td>Slowest available speed</td>
</tr>
<tr>
<td>0.99999</td>
<td>Fastest speed that is slower than normal</td>
</tr>
<tr>
<td>1.00001</td>
<td>Slowest speed that is faster than normal</td>
</tr>
<tr>
<td>9999</td>
<td>Fastest available speed</td>
</tr>
</tbody>
</table>

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 zero are rounded to one. However, if the player supports multiple speeds in CLV mode, VDI Management applies normal rounding rules.

Note: After requesting a play speed, a query for that speed returns the actual speed if it differs from the requested speed because of rounding.
### Section 8. Videodisc commands (vd)

**VdGetState**

**Last revision:** R 1.0

**Type:** core

#### Parameters

**ASCII**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Core or extended</th>
<th>Associated calling value</th>
<th>Type as ASCII</th>
<th>Associated return value</th>
<th>Type as ASCII</th>
<th>Default if parameter not used</th>
</tr>
</thead>
<tbody>
<tr>
<td>audio1</td>
<td>Core</td>
<td>None</td>
<td>N/A</td>
<td>1 (on)</td>
<td>0 (off)</td>
<td>Integer</td>
</tr>
<tr>
<td>audio2</td>
<td>Core</td>
<td>None</td>
<td>N/A</td>
<td>1 (on)</td>
<td>0 (off)</td>
<td>Integer</td>
</tr>
<tr>
<td>cdisplay</td>
<td>Core</td>
<td>None</td>
<td>N/A</td>
<td>1 (on)</td>
<td>0 (off)</td>
<td>Integer</td>
</tr>
<tr>
<td>chapter</td>
<td>Core</td>
<td>None</td>
<td>N/A</td>
<td>Current chapter number</td>
<td>Integer</td>
<td>No action</td>
</tr>
<tr>
<td>defdevice</td>
<td>Core</td>
<td>None</td>
<td>N/A</td>
<td>Default player, 0–15</td>
<td>Integer</td>
<td>No action</td>
</tr>
<tr>
<td>device</td>
<td>Core</td>
<td>Logical player, 0–15</td>
<td>Integer</td>
<td>None</td>
<td>N/A</td>
<td>Default player</td>
</tr>
<tr>
<td>discotype</td>
<td>Core</td>
<td>None</td>
<td>N/A</td>
<td>1 (CLV)</td>
<td>0 (CAV)</td>
<td>Integer</td>
</tr>
<tr>
<td>door</td>
<td>Extended</td>
<td>None</td>
<td>N/A</td>
<td>1 (open)</td>
<td>0 (closed)</td>
<td>Integer</td>
</tr>
<tr>
<td>frame¹</td>
<td>Core</td>
<td>None</td>
<td>N/A</td>
<td>Current frame number</td>
<td>Integer</td>
<td>No action</td>
</tr>
<tr>
<td>idxdisplay</td>
<td>Core</td>
<td>None</td>
<td>N/A</td>
<td>1 (on)</td>
<td>0 (off)</td>
<td>Integer</td>
</tr>
<tr>
<td>motion</td>
<td>Core</td>
<td>None</td>
<td>N/A</td>
<td>1 (on)</td>
<td>0 (off)</td>
<td>Integer</td>
</tr>
<tr>
<td>remote</td>
<td>Extended</td>
<td>None</td>
<td>N/A</td>
<td>1 (on)</td>
<td>0 (off)</td>
<td>Integer</td>
</tr>
<tr>
<td>speed</td>
<td>Core</td>
<td>None</td>
<td>N/A</td>
<td>Current player speed or 999 if scanning</td>
<td>Real</td>
<td>No action</td>
</tr>
<tr>
<td>spin</td>
<td>Core</td>
<td>None</td>
<td>N/A</td>
<td>1 (up)</td>
<td>0 (down)</td>
<td>Integer</td>
</tr>
<tr>
<td>tdevices</td>
<td>Core</td>
<td>None</td>
<td>N/A</td>
<td>Total installed for, 0–15</td>
<td>Integer</td>
<td>No action</td>
</tr>
<tr>
<td>video</td>
<td>Core</td>
<td>None</td>
<td>N/A</td>
<td>1 (on)</td>
<td>0 (off)</td>
<td>Integer</td>
</tr>
</tbody>
</table>

At least one parameter must be specified or an error occurs. If device is specified, at least one other parameter must be specified.

¹Supported for CAV videodiscs only. All other parameters apply to both CAV and CLV videodiscs.
### vdGetState

**Binary**

**Command code:** 3078 decimal.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Core or extended</th>
<th>Token number (decimal)</th>
<th>Type</th>
<th>Associated calling value</th>
<th>Associated return value</th>
<th>Default if parameter not used</th>
</tr>
</thead>
<tbody>
<tr>
<td>audio1</td>
<td>Core</td>
<td>2</td>
<td>Integer</td>
<td>Any value</td>
<td>1 (on)</td>
<td>0 (off)</td>
</tr>
<tr>
<td>audio2</td>
<td>Core</td>
<td>3</td>
<td>Integer</td>
<td>Any value</td>
<td>1 (on)</td>
<td>0 (off)</td>
</tr>
<tr>
<td>cdisplay</td>
<td>Core</td>
<td>6</td>
<td>Integer</td>
<td>Any value</td>
<td>1 (on)</td>
<td>0 (off)</td>
</tr>
<tr>
<td>chapter</td>
<td>Core</td>
<td>7</td>
<td>Integer</td>
<td>Any value</td>
<td>Current chapter number</td>
<td>No action</td>
</tr>
<tr>
<td>defdevice</td>
<td>Core</td>
<td>12</td>
<td>Integer</td>
<td>Any value</td>
<td>Default player, 0-15</td>
<td>No action</td>
</tr>
<tr>
<td>device</td>
<td>Core</td>
<td>14</td>
<td>Integer</td>
<td>Logical player, 0-15</td>
<td>None</td>
<td>Default player</td>
</tr>
<tr>
<td>disctype</td>
<td>Core</td>
<td>16</td>
<td>Integer</td>
<td>Any value</td>
<td>1 (CLV)</td>
<td>0 (CAV)</td>
</tr>
<tr>
<td>door</td>
<td>Extended</td>
<td>18</td>
<td>Integer</td>
<td>Any value</td>
<td>1 (open)</td>
<td>0 (closed)</td>
</tr>
<tr>
<td>frame(^1)</td>
<td>Core</td>
<td>23</td>
<td>Integer</td>
<td>Any value</td>
<td>Current frame number</td>
<td>No action</td>
</tr>
<tr>
<td>idxdisplay</td>
<td>Core</td>
<td>29</td>
<td>Integer</td>
<td>Any value</td>
<td>1 (on)</td>
<td>0 (off)</td>
</tr>
<tr>
<td>motion</td>
<td>Core</td>
<td>35</td>
<td>Integer</td>
<td>Any value</td>
<td>1 (on)</td>
<td>0 (off)</td>
</tr>
<tr>
<td>remote</td>
<td>Extended</td>
<td>39</td>
<td>Integer</td>
<td>Any value</td>
<td>1 (on)</td>
<td>0 (off)</td>
</tr>
<tr>
<td>speed</td>
<td>Core</td>
<td>40</td>
<td>Real</td>
<td>Any value</td>
<td>Current player speed or 999 if scanning</td>
<td>No action</td>
</tr>
<tr>
<td>spin</td>
<td>Core</td>
<td>41</td>
<td>Integer</td>
<td>Any value</td>
<td>1 (up)</td>
<td>0 (down)</td>
</tr>
<tr>
<td>tdevices</td>
<td>Core</td>
<td>45</td>
<td>Integer</td>
<td>Any value</td>
<td>Total installed for, 0-15</td>
<td>No action</td>
</tr>
<tr>
<td>video</td>
<td>Core</td>
<td>51</td>
<td>Integer</td>
<td>Any value</td>
<td>1 (on)</td>
<td>0 (off)</td>
</tr>
</tbody>
</table>

At least one parameter must be specified or an error occurs. If `device` is specified, at least one other parameter must be specified.

\(^1\)Supported for CAV videodiscs only. All other parameters apply to both CAV and CLV videodiscs.
vdGetState

Description

Summary

vdGetState returns information about the videodisc player specified by the device parameter or the default player if no device number is specified.

Audio1 and audio2 parameters

The audio1 and audio2 parameters return one if the respective audio channel is on and zero if it is off.

Cdisplay parameter

The cdisplay parameter returns one if the player's chapter number display is on and zero if it is not. Using cdisplay with videodiscs that do not have chapter numbers returns error 88 (Unable to return requested information).

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.

Defdevice parameter

The defdevice parameter returns the default logical player number as set by vdSet defdevice or vdInit. VDI Management directs all videodisc commands to this player unless a command contains a device parameter (see below) directing it to a different player.

Device parameter

The device parameter directs vdGetState to the specified logical player number regardless of the current player number as set by vdSet defdevice. Because, in general, device affects the command with which it is associated only, the parameter does not affect the return value for defdevice (see above) when the two parameters are used together.

Specifying device with no other parameter returns error 49 (Insufficient parameters). Specifying a nonexistent or uninstalled player returns error 160 (Invalid device number). Specifying an uninitialized player returns error 81 (Device not initialized).

Disctype parameter

The disctype parameter returns one if the videodisc is a CLV disc and zero if it is a CAV disc.
vdGetState

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. 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).

Door is an extended parameter. Using an unimplemented extended parameter causes error 48 (Unknown parameter).

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 discs.

Idxdisplay parameter The idxdisplay parameter returns one if player's frame number (CAV) or time (CLV) display is on and zero if it is not.

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.

Remote parameter The remote parameter returns one if the player's remote control unit is on and zero if it is off. If a VDI implementation supports the remote parameter but the player does not support a remote control unit, remote returns zero.

Remote is an extended parameter. Using an unimplemented extended parameter causes error 48 (Unknown parameter).

Speed parameter The speed parameter returns the actual player speed after any necessary rounding or 999 if the player is in scan mode. A return of zero indicates the player is parked or on a still frame. (See Section 8.2 for information on rounding speed values.)

Spin parameter The spin parameter returns zero if the player is parked or one if the videodisc is spinning and the player is ready to accept motion commands.

Tdevices parameter The tdevices parameter returns the total number of logical players for which VDI Management was configured when it was installed. If only one player is connected, it is numbered zero, and tdevices returns one.
Section 8. Videodisc commands (vd)

vdGetState

Video parameter
The video parameter returns one if the player’s video channel is on and zero if it is not.

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.

Notes
1. vdGetState can be successfully issued any time VDI Management can accept commands. The current state of the player does not affect whether the command can be issued. However, the player’s state can affect the ability to return specific parameter values, and therefore cause errors. For example, vdGetState frame returns error 88 (Unable to return requested information) if the player is parked.

2. Trying to queue vdGetState causes error 177 (Command cannot be queued) at the time of the attempt.

Returns

ASCII
On success: Comma-separated list of values for requested parameters as described above.

On failure: “ERROR n...”.

Binary
On success: AX = 0. Values associated with requested parameters are 32-bit values of the types given in the binary parameter table above.

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

See also: syGetState, vdInit, vdSet, vmGetState, xyGetState.

Examples

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
vdGetState

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 currently selected player
vdGetState device=1,video,defdevice (returns) "1,2"
; player 1 video is on,
; default player is logical number 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 motion value after return

After return
AX 0 ; returns 0 if successful (nonzero if not)
ES:DI[4] 1 ; value for motion, player is playing or scanning
vdInit

**Parameters**

**ASCII**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Core or extended</th>
<th>Associated calling type</th>
<th>Type as ASCII</th>
<th>Associated return value</th>
<th>Type as ASCII</th>
<th>Default if parameter not used</th>
</tr>
</thead>
<tbody>
<tr>
<td>device&lt;sup&gt;1&lt;/sup&gt;</td>
<td>Core</td>
<td>Logical player, 0-15</td>
<td>Integer</td>
<td>None</td>
<td>N/A</td>
<td>Default player</td>
</tr>
</tbody>
</table>

This command can be issued with no parameters.

<sup>1</sup>This parameter applies to both CAV and CLV videodiscs as does vdInit with no parameters.

**Binary**

Command code: 3079 decimal.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Core or extended</th>
<th>Token number (decimal)</th>
<th>Type</th>
<th>Associated calling type</th>
<th>Associated return value</th>
<th>Default if parameter not used</th>
</tr>
</thead>
<tbody>
<tr>
<td>device&lt;sup&gt;1&lt;/sup&gt;</td>
<td>Core</td>
<td>14</td>
<td>Integer</td>
<td>Logical player, 0-15</td>
<td>None</td>
<td>Default player</td>
</tr>
</tbody>
</table>

This command can be issued with no parameters.

<sup>1</sup>This parameter applies to both CAV and CLV videodiscs as does vdInit with no parameters.

**Description**

**Summary**

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 will not be able to issue vdInit until the motion command is complete.

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.
vdlnit

The resulting display after vdlnit 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.

Device parameter

The device parameter specifies the logical player number to be initialized. If device is omitted, vdlnit initializes the default player as set by vdSet defdevice. If vdSet has not been used to set a default player, the default player is defined to be number zero. vdlnit does not change the default player if a device other than the default is specified. To change the default, use vdSet defdevice.

Specifying a nonexistent or uninstalled player causes error 160 (Invalid device number).

Conditions set by vdlnit

vdlnit sets the parameters in the following table to the specified values.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Command reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>audio1</td>
<td>1 (on)</td>
<td>vdSet</td>
</tr>
<tr>
<td>audio2</td>
<td>1 (on)</td>
<td>vdSet</td>
</tr>
<tr>
<td>cdisplay</td>
<td>0 (off) or undefined</td>
<td>vdSet</td>
</tr>
<tr>
<td>door†</td>
<td>0 (closed)</td>
<td>vdSet</td>
</tr>
<tr>
<td>frame</td>
<td>First available on disc</td>
<td>vdPlay, vdSet</td>
</tr>
<tr>
<td>ldxdisplay</td>
<td>0 (off)</td>
<td>vdSet</td>
</tr>
<tr>
<td>motion</td>
<td>0 (off)</td>
<td>vdGetState</td>
</tr>
<tr>
<td>remote†</td>
<td>0 (off)</td>
<td>vdSet</td>
</tr>
<tr>
<td>spin</td>
<td>1 (up)</td>
<td>vdSet</td>
</tr>
<tr>
<td>tdevices</td>
<td>Total installed, 0–15</td>
<td>none</td>
</tr>
<tr>
<td>video</td>
<td>1 (on)</td>
<td>vdSet</td>
</tr>
</tbody>
</table>

†If supported by implementation—this is an extended parameter.

Notes

1. vdlnit 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 Note 4 below).

2. vdGetState tdevices returns the total number of players for which VDI Management was installed. This command can be used after the first vdlnit to determine the number of additional devices that can be initialized.

3. If the player supports a character generator, vdlnit turns it off.
Section 8. Videodisc commands (vd)

vdInit

4. Trying to queue vdInit causes error 177 (Command cannot be queued) at the time of the attempt.

5. 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.

6. Spinning the videodisc up also updates the disctype parameter, and sets edisplay to undefined for videodiscs that do not support chapter numbers.

7. If vdInit returns an error, the parameters listed in the table above have undefined values.

Returns

ASCII

On success: “OK”.

On failure: “ERROR n...”.

Binary

On success: AX = 0.

On failure: AX = error number.

See also: syInit, vdGetstate, vdSet, vmInit, xyInit.

Examples

ASCII

Initialize player 0 and vd service group

vdInit (returns) “OK” ; first time command is issued

Initialize player 1

vdInit device=1 (returns) “OK”
vdInit

Binary

Initialize player 0
AX 3079 ; vdInit decimal ID
BX 0 ; number of parameters
and vd service
After return AX 0 ; returns 0 if successful (nonzero if not)
group

Initialize player 1
AX 3079 ; vdInit decimal ID
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 (nonzero if not)
Section 8. Videodisc commands (vd)

vdPassThru

Last revision: R 1.0

Type: core

Parameters

ASCII
No named parameters (see discussion of device and pmsg parameters below). Applies to both CAV and CLV videodiscs.

Binary

Command code: 3080 decimal.

<table>
<thead>
<tr>
<th>Parameter^1</th>
<th>Core or extended</th>
<th>Token number (decimal)</th>
<th>Type</th>
<th>Associated calling value</th>
<th>Associated return value</th>
<th>Default if parameter not used</th>
</tr>
</thead>
<tbody>
<tr>
<td>device</td>
<td>Core</td>
<td>14</td>
<td>Integer</td>
<td>Logical player, 0–15</td>
<td>None</td>
<td>Default player</td>
</tr>
<tr>
<td>pmsg</td>
<td>Core</td>
<td>37</td>
<td>Pointer</td>
<td>Pointer to player message string</td>
<td>Player response string</td>
<td>Causes error</td>
</tr>
</tbody>
</table>

This command must be issued with the pmsg parameter or an error is returned.

^1 All parameters apply to both CAV and CLV videodiscs.

Description

Caution

vdPassThru is provided to allow nonportable access to special features of videodisc players. It is included to as a convenience to developers who want to use the command set for portable applications and do not want to switch to a different command environment for development efforts that require access to nonportable player functions that are not provided by other commands in the videodisc service group. Therefore, although it is required, it is supplied for convenience only and SHOULD NOT be used for developing portable applications.

Summary

vdPassThru communicates directly with a player, bypassing the standard videodisc service group commands and parameters. It is provided to allow access to specific player features that are not supported by other VDI videodisc commands. vdPassThru passes an string of printable ASCII characters to the player and waits for the player's response. This response is returned to the application. vdPassThru does not return application control until it receives a response from the player or detects an error.
vdPassThru

**Device parameter**
The device parameter directs `vdPassThru` to the specified logical player number regardless of the default player number as set by `vdSet defdevice`. This parameter applies to the binary interface only.

Specifying a nonexistent or uninstalled player causes error 160 (Invalid device number). Specifying an uninitialized player causes error 81 (Device not initialized).

The ASCII interface does not accept a device parameter because of potential difficulties in distinguishing the parameter label and associated value from the string that should be passed to the player. Therefore, the ASCII interface always sends the command string to the default player. (To set the default player, see the `vdSet` command.)

**Pmsg parameter**
The pmsg parameter value is a series of printable ASCII characters to be passed through to the player without modification by VDI Management. However, implementors may need to implement VDI Management so that it supplies a specific terminator if required by a specific supported player.

The binary interface passes a pointer to a null-terminated player message string. For the ASCII interface, all characters from the delimiter following the “u” in `vdPassThru` up to the terminating CR make up the message string. The ASCII interface does not use a pmsg parameter label.

**Returns**

**ASCII**
On success: Player response + CR/LF if CR/LF is not automatically returned by the player.

On failure: “ERROR n...”.

**Binary**
On success: AX = 0. String pointed to by the pmsg parameter contains the player response + NULL.

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

**Examples**

**ASCII**
Send a string to the player
`vdPassThru THIS IS A COMMAND`
(returs) "THIS IS A PLAYER RESPONSE"
Section 8. Videodisc commands (vd)

vdPassThru

**Binary**

<table>
<thead>
<tr>
<th>Send a command string to player 2</th>
<th>AX</th>
<th>3080</th>
<th>; vdPassThru decimal ID</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>BX</td>
<td>2</td>
<td>; number of parameters</td>
</tr>
<tr>
<td></td>
<td>ES:DI[0]</td>
<td>14</td>
<td>; device decimal ID</td>
</tr>
<tr>
<td></td>
<td>ES:DI[4]</td>
<td>1</td>
<td>; send message to player 1</td>
</tr>
<tr>
<td></td>
<td>ES:DI[8]</td>
<td>37</td>
<td>; pmsg decimal ID</td>
</tr>
<tr>
<td>After return</td>
<td>AX</td>
<td>0</td>
<td>; returns 0 if successful (nonzero if not)</td>
</tr>
<tr>
<td></td>
<td>ES:DI[C]</td>
<td>pointer</td>
<td>; pointer to player response string</td>
</tr>
</tbody>
</table>

April 15, 1990

Release R 1.0

8-17
vdPlay

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Core or extended</th>
<th>Associated calling value</th>
<th>Type as ASCII</th>
<th>Associated return value</th>
<th>Type as ASCII</th>
<th>Default if parameter not used</th>
</tr>
</thead>
<tbody>
<tr>
<td>chapter</td>
<td>Core</td>
<td>Chapter number</td>
<td>Integer</td>
<td>None</td>
<td>N/A</td>
<td>No action</td>
</tr>
<tr>
<td>device</td>
<td>Core</td>
<td>Logical player, 0–15</td>
<td>Integer</td>
<td>None</td>
<td>N/A</td>
<td>Default player</td>
</tr>
<tr>
<td>direction</td>
<td>Core</td>
<td>1 (fwd)</td>
<td>0 (back)</td>
<td>Integer</td>
<td>None</td>
<td>N/A</td>
</tr>
<tr>
<td>from</td>
<td>Core</td>
<td>Starting frame number</td>
<td>Integer</td>
<td>None</td>
<td>N/A</td>
<td>Current frame</td>
</tr>
<tr>
<td>speed</td>
<td>Core</td>
<td>Play speed, &gt;0</td>
<td>Real</td>
<td>None</td>
<td>N/A</td>
<td>1.0</td>
</tr>
<tr>
<td>to</td>
<td>Core</td>
<td>Ending frame number</td>
<td>Integer</td>
<td>None</td>
<td>N/A</td>
<td>Disc limit</td>
</tr>
<tr>
<td>wait</td>
<td>Core</td>
<td>None</td>
<td>N/A</td>
<td>None</td>
<td>N/A</td>
<td>No wait</td>
</tr>
</tbody>
</table>

This command may be issued with no parameters. The text describes illegal usage.

Supported for CAV videodiscs only. All other parameters apply to both CAV and CLV videodiscs as does vdPlay with no parameters.

Binary

Command code: 3081 decimal.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Core or extended</th>
<th>Token number (decimal)</th>
<th>Type</th>
<th>Associated calling value</th>
<th>Associated return value</th>
<th>Default if parameter not used</th>
</tr>
</thead>
<tbody>
<tr>
<td>chapter</td>
<td>Core</td>
<td>7</td>
<td>Integer</td>
<td>Chapter number</td>
<td>None</td>
<td>No action</td>
</tr>
<tr>
<td>device</td>
<td>Core</td>
<td>14</td>
<td>Integer</td>
<td>Logical player, 0–15</td>
<td>None</td>
<td>Default player</td>
</tr>
<tr>
<td>direction</td>
<td>Core</td>
<td>15</td>
<td>Integer</td>
<td>1 (fwd)</td>
<td>0 (back)</td>
<td>None</td>
</tr>
<tr>
<td>from</td>
<td>Core</td>
<td>24</td>
<td>Integer</td>
<td>Starting frame number</td>
<td>None</td>
<td>Current frame</td>
</tr>
<tr>
<td>speed</td>
<td>Core</td>
<td>40</td>
<td>Real</td>
<td>Play speed, &gt;0</td>
<td>Actual speed after rounding if required</td>
<td>1.0</td>
</tr>
<tr>
<td>to</td>
<td>Core</td>
<td>48</td>
<td>Integer</td>
<td>Ending frame number</td>
<td>None</td>
<td>Disc limit</td>
</tr>
<tr>
<td>wait</td>
<td>Core</td>
<td>54</td>
<td>N/A</td>
<td>None</td>
<td>None</td>
<td>No wait</td>
</tr>
</tbody>
</table>

This command may be issued with no parameters. The text describes illegal usage.

Supported for CAV videodiscs only. All other parameters apply to both CAV and CLV videodiscs as does vdPlay with no parameters.
### Section 8. Videodisc commands (vd)

## vdPlay

### Description

**Summary**  
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.

**No parameters**  
vdPlay issued with no parameters causes the player to start playing forward from the current frame at a speed of 1.0 and continues until interrupted by a subsequent vdInit, vdPlay, vdScan, vdSearch, vdSet spin=down, vdStep, or vdStill command, or until the player reaches the end of the videodisc.

**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 to return application control until the chapter has been played.

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).

Compatible parameters—device, speed, and wait. Other parameters cause error 50 (Parameters cannot be used together).

**Device parameter**  
The device parameter directs vdPlay to the specified logical player number regardless of the default player number as 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).

Compatible parameters—all, assuming other parameters are compatible with each other.

**Direction parameter**  
The direction parameter sets the direction of motion (1 = forward, 0 = backward) for play sequences that do not include to frames. Specifying a direction with no from frame or chapter 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. Specifying a direction with a chapter is discussed above (see the chapter parameter above).
vdPlay

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 at best redundant if it agrees with the predetermined direction, and at worst conflicting if it opposes the predetermined direction.

Compatible parameters—from, device, speed, and wait. Other parameters cause error 50 (Parameters cannot be used together).

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 and 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.

A from parameter with no to parameter starts an unbounded play. The play sequence continues until interrupted by another vdPlay, or a vdInit, vdScan, vdSearch, vdSet spin=down, vdStep, or vdStill command, or until the player reaches the edge of the videodisc.

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).

Compatible parameters—device, direction OR to, speed, and wait. Other parameters or illegal combinations of compatible parameters cause error 50 (Parameters cannot be used together).

Speed parameter

The speed parameter specifies the speed of play. VDI Management maps requested speeds as closely as possible to available player speeds. 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 and vdGetState speed returns the actual speed after any necessary rounding. For CAV mode, speeds are never rounded to zero or one. (See Section 8.2 for detailed information on rounding speed values including boundary conditions.)

Specifying a speed less than or equal to zero causes error 51 (Parameter value invalid or out of range).

Compatible parameters—device, direction OR to, from, chapter but not with from or to, and wait. Other parameters or illegal combinations of compatible parameters cause error 50 (Parameters cannot be used together).
Section 8. Videodisc commands (vd)

**vdPlay**

**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 displaying the frame.

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.

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). Specifying a to frame with a direction is covered above (see Direction parameter).

Compatible parameters—device, from, speed, and wait. Other parameters cause error 50 (Parameters cannot be used together).

**Wait parameter**
The effect of the wait parameter depends on the parameters that accompany it. The following table lists when vdPlay wait... returns application control based on accompanying parameters. Obviously, if the player returns an error, the command will return when the error state is detected instead of at the time given in the table.

<table>
<thead>
<tr>
<th>Additional parameter</th>
<th>Returns control when?</th>
</tr>
</thead>
<tbody>
<tr>
<td>none</td>
<td>After the default player is playing normally</td>
</tr>
<tr>
<td>chapter</td>
<td>After the player has played the specified chapter</td>
</tr>
<tr>
<td>device only</td>
<td>After the specified player is playing normally</td>
</tr>
<tr>
<td>from with any other legal parameters except to</td>
<td>After the player has searched to the specified from frame</td>
</tr>
<tr>
<td>speed only</td>
<td>After the player is playing normally at the specified speed</td>
</tr>
<tr>
<td>to with any other legal parameters</td>
<td>After the player reached the specified to frame</td>
</tr>
</tbody>
</table>

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, `syCheckErr` must be used to determine if the player entered an error state while either accepting or trying to execute the command.
vdPlay

syCheckErr may also 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. syCheckErr is required to detect any error state that occurs after the player has reached frame 1000.

Note that without wait, a subsequent vdPlay, vdInit, vdScan, vdSearch, vdSet spin=down, vdStep or vdStill command immediately interrupts an executing play sequence, even if the play sequence specifies a target—either a to frame or the end of a chapter.

Compatible parameters—all, assuming other parameters are compatible with each other.

Notes

1. vdPlay can be successfully issued any time spin=1 (up) as set by vdSet.
2. 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 the to frame has been reached.
3. All parameters apply to the current vdPlay only. For example, vdPlay to=1000, speed=.5 does not set the speed to a default of 0.5. A subsequent vdPlay without a speed will play at speed 1.0, not 0.5.

Returns

ASCII

On success: "OK".
On failure: "ERROR n...".

Binary

On success: AX = 0. Value associated with speed parameter is a 32-bit real that gives the actual speed that will be set after rounding if required.
On failure: AX = error number.

See also:

syCheckErr, vdGetState, vdScan, vdSearch vdStep, vdStill.

Examples

ASCII

Play forward at speed 1.0 from current frame
vdPlay
  (returns) "OK"
vdPlay

Play backward
from frame 1000
at the slowest pos-
sible speed
vdPlay speed=.00001,direction=0,from=1000
(returns) “OK”

Play from current
frame to 2000,
don’t 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”
or
vdPlay to=100,from=200
(returns) “OK”

Play backward
from the current
frame
vdPlay direction=0
(returns) “OK”

Play all of chapter
2 at speed 1.0
vdPlay chapter=2
(returns) “OK”

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 0 if successful (nonzero if not)
vdScan

Parameters

ASCII

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Core or extended</th>
<th>Associated calling value</th>
<th>Type as ASCII</th>
<th>Associated return value</th>
<th>Type as ASCII</th>
<th>Default if parameter not used</th>
</tr>
</thead>
<tbody>
<tr>
<td>device</td>
<td>Core</td>
<td>Logical player, 0-15</td>
<td>Integer</td>
<td>None</td>
<td>N/A</td>
<td>Default player</td>
</tr>
<tr>
<td>direction</td>
<td>Core</td>
<td>1 (fwd)</td>
<td>0 (back)</td>
<td>Integer</td>
<td>None</td>
<td>N/A</td>
</tr>
<tr>
<td>wait</td>
<td>Core</td>
<td>None</td>
<td>N/A</td>
<td>None</td>
<td>N/A</td>
<td>No wait</td>
</tr>
</tbody>
</table>

This command may be issued with no parameters.

1Supported for CAV videodiscs only. All other parameters apply to both CAV and CLV videodiscs as does vdScan with no parameters.

Binary

Command code: 3083 decimal.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Core or extended</th>
<th>Token number (decimal)</th>
<th>Type</th>
<th>Associated calling value</th>
<th>Associated return value</th>
<th>Default if parameter not used</th>
</tr>
</thead>
<tbody>
<tr>
<td>device</td>
<td>Core</td>
<td>14</td>
<td>Integer</td>
<td>Logical player, 0-15</td>
<td>None</td>
<td>Default player</td>
</tr>
<tr>
<td>direction</td>
<td>Core</td>
<td>15</td>
<td>Integer</td>
<td>1 (fwd)</td>
<td>0 (back)</td>
<td>None</td>
</tr>
<tr>
<td>wait</td>
<td>Core</td>
<td>54</td>
<td>N/A</td>
<td>None</td>
<td>None</td>
<td>No wait</td>
</tr>
</tbody>
</table>

This command may be issued with no parameters.

1Supported for CAV videodiscs only. All other parameters apply to both CAV and CLV videodiscs as does vdScan with no parameters.

Description

Summary

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 vdPlay, vdInit, vdScan, vdSearch, vdSet spin=down, 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 will not be able to issue vdScan until the motion command is complete.
Section 8. Videodisc commands (vd)

vdScan

No parameters  
vdScan with no parameters starts scanning forward from the current frame.

Device parameter  
The device parameter directs vdScan to the specified logical player number regardless of the default player number as 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).

Direction parameter  
The direction parameter sets the direction of motion for the scan, either one (forward) or zero (backward).

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, syCheckErr must be used to determine if the player entered an error state while either accepting or trying to execute the command.

Notes  
1. Because vdScan often results in displaying parts of frames and does not accept parameters to limit the area of the videodisc that is scanned, the command is typically used during application development only.

Returns  

ASCII  
On success: “OK”.
On failure: “ERROR n…”

Binary  
On success: AX = 0.
On failure: AX = error number.

See also:  
vdPlay, vdInit, vdScan, vdSearch, vdSet, vdStep, vdStill.
vdScan

Examples

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"

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 ; direction in which to play (backward)

After return
AX 0 ; returns 0 if successful (nonzero if not)
Section 8. Videodisc commands (vd)

vdSearch

Last revision: R 1.0
Type: core

Parameters

ASCII

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Core or extended</th>
<th>Associated calling value</th>
<th>Type as ASCII</th>
<th>Associated return value</th>
<th>Type as ASCII</th>
<th>Default if parameter not used</th>
</tr>
</thead>
<tbody>
<tr>
<td>chapter</td>
<td>Core</td>
<td>Chapter number</td>
<td>Integer</td>
<td>None</td>
<td>N/A</td>
<td>No action</td>
</tr>
<tr>
<td>device</td>
<td>Core</td>
<td>Logical player, 0–15</td>
<td>Integer</td>
<td>None</td>
<td>N/A</td>
<td>Default player</td>
</tr>
<tr>
<td>frame¹</td>
<td>Core</td>
<td>Frame number</td>
<td>Integer</td>
<td>None</td>
<td>N/A</td>
<td>No action</td>
</tr>
<tr>
<td>wait</td>
<td>Core</td>
<td>None</td>
<td>N/A</td>
<td>None</td>
<td>N/A</td>
<td>No wait</td>
</tr>
</tbody>
</table>

This command must include a chapter or frame or an error is returned. If device or wait is specified, at least one other parameter must be specified.

¹Supported for CAV videodiscs only. All other parameters apply to both CAV and CLV.

Binary

Command code: 3084 decimal.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Core or extended</th>
<th>Token number (decimal)</th>
<th>Type</th>
<th>Associated calling value</th>
<th>Associated return value</th>
<th>Default if parameter not used</th>
</tr>
</thead>
<tbody>
<tr>
<td>chapter</td>
<td>Core</td>
<td>7</td>
<td>Integer</td>
<td>Chapter number</td>
<td>None</td>
<td>No action</td>
</tr>
<tr>
<td>device</td>
<td>Core</td>
<td>14</td>
<td>Integer</td>
<td>Logical player, 0–15</td>
<td>None</td>
<td>Default player</td>
</tr>
<tr>
<td>frame¹</td>
<td>Core</td>
<td>23</td>
<td>Integer</td>
<td>Frame number</td>
<td>None</td>
<td>No action</td>
</tr>
<tr>
<td>wait</td>
<td>Core</td>
<td>54</td>
<td>N/A</td>
<td>None</td>
<td>None</td>
<td>No wait</td>
</tr>
</tbody>
</table>

This command must include a chapter or frame or an error is returned. If device or wait is specified, at least one other parameter must be specified.

¹Supported for CAV videodiscs only. All other parameters apply to both CAV and CLV.

Description

Summary

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 will not be able to issue vdSearch until the motion command is complete.
vdSearch

The resulting display after vdSearch varies with videodisc type. With CAV videodiscs, video remains visible. 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.

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).

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).

Device parameter

The device parameter directs vdSearch to the specified logical player number regardless of the default player number as 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).

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).

Wait parameter

The wait parameter causes VDI Management to wait until the specified chapter or frame has been reached 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, syCheckErr must be used to determine if the player entered an error state while either accepting or trying to execute the command.

Returns

ASCII

On success: "OK".
On failure: "ERROR n..."

Binary

On success: AX = 0.
On failure: AX = error number.

See also:

vdPlay, vdSet, vdStep, vdStill.

8-28
April 15, 1990
Section 8. Videodisc commands (vd)

vdSearch

Examples

**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"

**Binary**

**Search for frame 10356**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3084</td>
<td>1</td>
<td>23</td>
<td>10356</td>
</tr>
</tbody>
</table>

; vdSearch decimal ID

; number of parameters

; frame decimal ID

; frame number for which to search

**After return**

<table>
<thead>
<tr>
<th>AX</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
</tbody>
</table>

; returns 0 if successful (nonzero if not)
Recommended Practices for Interactive Video Portability

vdSet

Last revision: R 1.0
Type: core

Parameters

<table>
<thead>
<tr>
<th>Parameter1</th>
<th>Core or extended</th>
<th>Associated calling value</th>
<th>Type as ASCII</th>
<th>Associated return value</th>
<th>Type as ASCII</th>
<th>Default if parameter not used</th>
</tr>
</thead>
<tbody>
<tr>
<td>audio1</td>
<td>Core</td>
<td>1 (on)</td>
<td>0 (off)</td>
<td>Integer</td>
<td>None</td>
<td>N/A</td>
</tr>
<tr>
<td>audio2</td>
<td>Core</td>
<td>1 (on)</td>
<td>0 (off)</td>
<td>Integer</td>
<td>None</td>
<td>N/A</td>
</tr>
<tr>
<td>cdisplay</td>
<td>Core</td>
<td>1 (on)</td>
<td>0 (off)</td>
<td>Integer</td>
<td>None</td>
<td>N/A</td>
</tr>
<tr>
<td>defdevice</td>
<td>Core</td>
<td>Logical player, 0-15</td>
<td>Integer</td>
<td>None</td>
<td>N/A</td>
<td>No action</td>
</tr>
<tr>
<td>device</td>
<td>Core</td>
<td>Logical player, 0-15</td>
<td>Integer</td>
<td>None</td>
<td>N/A</td>
<td>Default player</td>
</tr>
<tr>
<td>door</td>
<td>Extended</td>
<td>1 (open)</td>
<td>0 (closed)</td>
<td>Integer</td>
<td>None</td>
<td>N/A</td>
</tr>
<tr>
<td>idxdisplay</td>
<td>Core</td>
<td>1 (on)</td>
<td>0 (off)</td>
<td>Integer</td>
<td>None</td>
<td>N/A</td>
</tr>
<tr>
<td>remote</td>
<td>Extended</td>
<td>1 (on)</td>
<td>0 (off)</td>
<td>Integer</td>
<td>None</td>
<td>N/A</td>
</tr>
<tr>
<td>spin</td>
<td>Core</td>
<td>1 (up)</td>
<td>0 (down)</td>
<td>Integer</td>
<td>None</td>
<td>N/A</td>
</tr>
<tr>
<td>video</td>
<td>Core</td>
<td>1 (on)</td>
<td>0 (off)</td>
<td>Integer</td>
<td>None</td>
<td>N/A</td>
</tr>
<tr>
<td>wait</td>
<td>Core</td>
<td>None</td>
<td>N/A</td>
<td>None</td>
<td>N/A</td>
<td>No action (wait)</td>
</tr>
</tbody>
</table>

At least one parameter is required or an error is returned. If device or wait is specified, at least one other parameter must be specified.

1 All parameters apply to both CAV and CLV.
Section 8. Videodisc commands (vd)

vdSet

Binary

Command code: 3085 decimal.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Core or extended</th>
<th>Token number (decimal)</th>
<th>Type</th>
<th>Associated calling value</th>
<th>Associated return value</th>
<th>Default if parameter not used</th>
</tr>
</thead>
<tbody>
<tr>
<td>audio1</td>
<td>Core</td>
<td>2</td>
<td>Integer</td>
<td>1 (on)</td>
<td>0 (off)</td>
<td>None</td>
</tr>
<tr>
<td>audio2</td>
<td>Core</td>
<td>3</td>
<td>Integer</td>
<td>1 (on)</td>
<td>0 (off)</td>
<td>None</td>
</tr>
<tr>
<td>cddisplay</td>
<td>Core</td>
<td>6</td>
<td>Integer</td>
<td>1 (on)</td>
<td>0 (off)</td>
<td>None</td>
</tr>
<tr>
<td>defdevice</td>
<td>Core</td>
<td>12</td>
<td>Integer</td>
<td>Logical player, 0–15</td>
<td>None</td>
<td>No action</td>
</tr>
<tr>
<td>device</td>
<td>Core</td>
<td>14</td>
<td>Integer</td>
<td>Logical player, 0–15</td>
<td>None</td>
<td>Default player</td>
</tr>
<tr>
<td>door</td>
<td>Extended</td>
<td>18</td>
<td>Integer</td>
<td>1 (open)</td>
<td>0 (closed)</td>
<td>None</td>
</tr>
<tr>
<td>idxdisplay</td>
<td>Core</td>
<td>29</td>
<td>Real</td>
<td>1 (on)</td>
<td>0 (off)</td>
<td>None</td>
</tr>
<tr>
<td>remote</td>
<td>Extended</td>
<td>39</td>
<td>Integer</td>
<td>1 (on)</td>
<td>0 (off)</td>
<td>None</td>
</tr>
<tr>
<td>spin</td>
<td>Core</td>
<td>41</td>
<td>Integer</td>
<td>1 (up)</td>
<td>0 (down)</td>
<td>None</td>
</tr>
<tr>
<td>video</td>
<td>Core</td>
<td>51</td>
<td>Integer</td>
<td>1 (on)</td>
<td>0 (off)</td>
<td>None</td>
</tr>
<tr>
<td>wait</td>
<td>Core</td>
<td>54</td>
<td>N/A</td>
<td>None</td>
<td>None</td>
<td>No wait</td>
</tr>
</tbody>
</table>

At least one parameter is required or an error is returned. If device or wait is specified, at least one other parameter must be specified.

1 All parameters apply to both CAV and CLV.

Description

Summary

vdSet sets the default logical player number and other player conditions including the state of the audio and video channels, the frame 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.

Audio1 and audio2 parameters

The audio1 and audio2 parameters enable and disable the player's stereo outputs. Setting both audio1 and audio2 to zero turns off all player audio.

Note: 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.

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.
Recommended Practices for Interactive Video Portability

**vdSet**

**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 below) directing it to a different player number.

**Device parameter**

The `device` parameter directs `vdSet` to the specified logical player number regardless of the default player number as 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).

**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). 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.

*Door* is an extended parameter. Using an unimplemented extended parameter causes error 48 (Unknown parameter).

**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.

**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 48 (Unknown parameter).

**Spin parameter**

The `spin` parameter spins the disc up and down. `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.

`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 will not be able to issue `vdSet spin=0` until the motion command is complete.

---

8-32  Release R 1.0  April 15, 1990

---
### Section 8. Videodisc commands (vd)

**vdSet**

<table>
<thead>
<tr>
<th>Video parameter</th>
<th>The video parameter enables and disables the player's video output channel.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wait parameter</td>
<td>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. No error checking is done to determine if the player actually accepts the command or acts on it properly. Therefore, syCheckErr must be used to determine if the player entered an error state while either accepting or trying to execute the command. Specifying wait with no other parameter results in error 49 (Insufficient parameters).</td>
</tr>
</tbody>
</table>

#### Notes

1. vdSet can be successfully issued any time the player can accept commands, except that vdSet door=1 (open) can be issued only when spin=0 (down) and the player has actually completed the spin down sequence.

#### Returns

**ASCII**

On success: “OK”.

On failure: “ERROR n...”.

**Binary**

On success: AX = 0.

On failure: Ax = error number.

#### See also:

syCheckErr, vdGetState.

#### Examples

**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”
vdSet

Disable audio channel 1

\[ \text{vdSet audio1=0} \]

(returns) "OK"

Binary

<table>
<thead>
<tr>
<th>Turn on player index display</th>
<th>AX</th>
<th>3085</th>
<th>; vdSet decimal ID</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>BX</td>
<td>1</td>
<td>; number of parameters</td>
</tr>
<tr>
<td></td>
<td>ES:DI[0]</td>
<td>29</td>
<td>; idxdisplay decimal ID</td>
</tr>
<tr>
<td></td>
<td>ES:DI[4]</td>
<td>1</td>
<td>; set value to 1 (on)</td>
</tr>
</tbody>
</table>

After return

| AX   | 0    | ; returns 0 if successful (nonzero if not) |

Set video on, audio channel 1 on, audio channel 2 off

<table>
<thead>
<tr>
<th>AX</th>
<th>3085</th>
<th>; vdSet decimal ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>BX</td>
<td>3</td>
<td>; number of parameters</td>
</tr>
<tr>
<td>ES:DI[0]</td>
<td>51</td>
<td>; video decimal ID</td>
</tr>
<tr>
<td>ES:DI[4]</td>
<td>1</td>
<td>; set value to 1 (on)</td>
</tr>
<tr>
<td>ES:DI[8]</td>
<td>2</td>
<td>; audio1 decimal ID</td>
</tr>
<tr>
<td>ES:DI[C]</td>
<td>1</td>
<td>; set value to 1 (on)</td>
</tr>
<tr>
<td>ES:DI[10]</td>
<td>3</td>
<td>; audio2 decimal ID</td>
</tr>
<tr>
<td>ES:DI[14]</td>
<td>0</td>
<td>; set value to 0 (off)</td>
</tr>
</tbody>
</table>

After return

| AX   | 0    | ; returns 0 if successful (nonzero if not) |
vdStep

Parameters

ASCII

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Core or extended</th>
<th>Associated calling value</th>
<th>Type as ASCII</th>
<th>Associated return value</th>
<th>Type as ASCII</th>
<th>Default if parameter not used</th>
</tr>
</thead>
<tbody>
<tr>
<td>device</td>
<td>Core</td>
<td>Logical player, 0–15</td>
<td>Integer</td>
<td>None</td>
<td>N/A</td>
<td>Default player</td>
</tr>
<tr>
<td>direction</td>
<td>Core</td>
<td>1 (fwd) 0 (back)</td>
<td>Real</td>
<td>None</td>
<td>N/A</td>
<td>1</td>
</tr>
</tbody>
</table>

This command can be issued with no parameters.

1 No parameters apply to CLV videodiscs, nor does vdStep alone.

Binary

Command code: 3090 decimal.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Core or extended</th>
<th>Token number (decimal)</th>
<th>Type</th>
<th>Associated calling value</th>
<th>Associated return value</th>
<th>Default if parameter not used</th>
</tr>
</thead>
<tbody>
<tr>
<td>device</td>
<td>Core</td>
<td>14</td>
<td>Integer</td>
<td>Logical player, 0–15</td>
<td>None</td>
<td>Default player</td>
</tr>
<tr>
<td>direction</td>
<td>Core</td>
<td>15</td>
<td>Integer</td>
<td>1 (fwd) 0 (back)</td>
<td>None</td>
<td>1</td>
</tr>
</tbody>
</table>

This command can be issued with no parameters.

1 No parameters apply to CLV videodiscs, nor does vdStep alone.

Description

Summary

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 will not be able to issue vdPlay until the motion command is complete.

No parameters

With no parameters, vdStep steps forward one frame and freezes.

Device parameter

The device parameter directs vdStep to the specified logical player number regardless of the default player number as set by vdSet device. Specifying a nonexistent or uninstalled player causes error 160 (Invalid device number); specifying an uninitialized player causes error 81 (Device not initialized).
Recommended Practices for Interactive Video Portability

vdStep

Direction parameter

The direction parameter sets the direction of motion for the step, either one (forward) or zero (backward).

Notes

1. vdStep can be successfully issued any time the 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.

Returns

ASCII

On success: “OK”.

On failure: “ERROR n...”.

Binary

On success: AX = 0.

On failure: AX = error number.

See also:

syCheckerr, vdPlay, vdSet.

Examples

ASCII

Step forward 1 frame

vdStep

(returns) “OK”

Step backward one frame

vdStep direction=0

(returns) “OK”

Binary examples

Step forward one frame

AX 3090 ; vdStep decimal command ID

BX 0 ; number of parameters

After return

AX = 0 ; returns 0 if successful (non-zero if not)
Section 8. Videodisc commands (vd)

vdStill

Last revision: R 1.0
Type: core

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Core or extended</th>
<th>Associated calling value</th>
<th>Type as ASCII</th>
<th>Associated return value</th>
<th>Type as ASCII</th>
<th>Default if parameter not used</th>
</tr>
</thead>
<tbody>
<tr>
<td>device¹</td>
<td>Core</td>
<td>Logical player, 0–15</td>
<td>Integer</td>
<td>None</td>
<td>N/A</td>
<td>Default player</td>
</tr>
</tbody>
</table>

This command can be issued with no parameters.

¹This parameter applies to both CAV and CLV videodiscs as does vdStill with no parameters.

Binary

Command code: 3091 decimal.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Core or extended</th>
<th>Token number (decimal)</th>
<th>Type</th>
<th>Associated calling value</th>
<th>Associated return value</th>
<th>Default if parameter not used</th>
</tr>
</thead>
<tbody>
<tr>
<td>device¹</td>
<td>Core</td>
<td>14</td>
<td>Integer</td>
<td>Logical player, 0–15</td>
<td>None</td>
<td>Default player</td>
</tr>
</tbody>
</table>

This command can be issued with no parameters.

¹This parameter applies to both CAV and CLV videodiscs as does vdStill with no parameters.

Description

Summary

vdStill causes the videodisc player to immediately stop on the current frame and sets the motion parameter returned by vdGetState to zero. 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.

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.

Device parameter

The device parameter directs vdStill to the specified logical player number regardless of the default player number as 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).
vdStill

Returns

**ASCII**
On success: “OK”.
On failure: “ERROR n...”.

**Binary**
On success: AX = 0.
On failure: Ax= error number.

See also: vdPlay, vdScan, vdSet.

Examples

**ASCII**
Stop player motion vdStill
(returs) “OK”

**Binary**
Stop player motion AX 3091 ; vdStill decimal ID
BX 0 ; number of parameters
After return AX 0 ; returns 0 if successful (nonzero if not)
9 XY-input commands (xy)

This section describes commands that relate to XY-input devices such as mice, touchscreens, and light pens. These commands provide a uniform way to obtain information from these devices and define coordinate spaces. Table 9-1 lists the commands covered in this section, their token numbers, and their types.

<table>
<thead>
<tr>
<th>ASCII command</th>
<th>Binary Interface token number (decimal)</th>
<th>Type²</th>
</tr>
</thead>
<tbody>
<tr>
<td>xyGetInput</td>
<td>4100</td>
<td>Core</td>
</tr>
<tr>
<td>xyGetState</td>
<td>4102</td>
<td>Core</td>
</tr>
<tr>
<td>xyInit</td>
<td>4103</td>
<td>Core</td>
</tr>
<tr>
<td>xySet</td>
<td>4109</td>
<td>Core</td>
</tr>
</tbody>
</table>

¹ Upper or lower case for command names is not significant.
² Compliant implementations must support “Core” commands.

9.1 General information and assumptions

The general information and assumptions in this subsection were used in the definition of the XY-input commands.

9.1.1 Device mapping

Typically, each physical XY-input device is treated independently and mapped to a unique logical device number. However, VDI implementers may opt 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 both devices return the same value for the same screen position to the application based on the application-coordinate space established with the xySet command.
All mapping must be done when VDI Management is installed. Devices cannot be remapped at run-time and mapping is not under application control. Mapping of multiple devices to a single logical device allows a user to use, for example, a mouse and a touchscreen that both appear to be the same device from the application’s viewpoint.

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 recommended practices.

Note that the xy service group keeps sets of all parameters that can be returned by xyGetInput and xyGetState for each logical device.

9.1.2 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 active for a device such as a mouse to display a cursor for making menu selections and similar tasks.

Although some XY-input devices such as touchscreens allow input beyond the limits of active graphics, applications should limit active XY-input areas to the active graphics plane. Again, this is necessary for devices such as mice that rely on the graphics plane for cursor display.

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.

9.1.3 Coordinate space mapping

The alignment of specific XY-coordinate values versus graphics is an implementation issue. However, the minimum and maximum values for X and Y always map to the edges of the active graphics area with the upper left corner of the graphics area as the origin, which is equal to xmin and ymin.

For example, if the minimum value for X is 0 and the maximum is 10, these map to the left and right edges of active graphics (typically 0 and 319 or 0 and 639), respectively. If the minimum and maximum values are -100 and +100, these still map to the left and right edges of active graphics.

The 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.

For relative positioning devices such as mice, VDI Management ignores changes in position which take the cursor outside of the clipping area. For absolute positioning devices such as touchscreens, VDI Management ignores
button presses outside of the clipping area. Application authors should note that the behavior differs between the two device classes and should consider testing applications against both.

Calibrating the XY-coordinate space to the active graphics area is an implementation and application issue. Typically, if a device such as a touchscreen requires calibration, the device comes with software to support its calibration at installation.

9.1.4 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 touchscreen or pressing a physical button on a mouse. The command set supports devices with multiple buttons. However, applications should assume single-button devices for maximum portability.

The command set supports reporting only whether a button has been pressed. It does not distinguish touchdown, liftoff, or intensity (Z dimension). Supporting these variations is an application issue and is nonportable.

9.2 Stream-mode and point-mode devices

XY-input devices fall into two broad categories based on how they make positional information available—stream-mode and point-mode. Some devices support one mode only, while others support both depending on configuration.

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.

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.

To treat both true stream-mode devices and point-mode devices as stream-mode devices, the reported coordinates will be one of:

- the current coordinates from a true stream-mode device; or
- the coordinates at the time the button was last pressed for a point-mode device; or
- 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.
xyGetInput

Parameters

ASCII

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Core or extended</th>
<th>Associated calling value</th>
<th>Type as ASCII</th>
<th>Associated return value</th>
<th>Type as ASCII</th>
<th>Default if parameter not used</th>
</tr>
</thead>
<tbody>
<tr>
<td>buttons</td>
<td>Core</td>
<td>None</td>
<td>N/A</td>
<td>Integer sum of bit field, 1 (closed)</td>
<td>Integer</td>
<td>No action</td>
</tr>
<tr>
<td>device</td>
<td>Core</td>
<td>Logical input device, 0–15</td>
<td>Integer</td>
<td>None</td>
<td>N/A</td>
<td>Default device</td>
</tr>
<tr>
<td>xpos</td>
<td>Core</td>
<td>None</td>
<td>N/A</td>
<td>Current X value</td>
<td>Integer</td>
<td>No action</td>
</tr>
<tr>
<td>ypos</td>
<td>Core</td>
<td>None</td>
<td>N/A</td>
<td>Current Y value</td>
<td>Integer</td>
<td>No action</td>
</tr>
</tbody>
</table>

At least one parameter is required or an error is returned. If device is specified, at least one other parameter must be specified.

Binary

Command code: 4100 decimal.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Core or extended</th>
<th>Token number (decimal)</th>
<th>Type</th>
<th>Associated calling value</th>
<th>Associated return value</th>
<th>Default if parameter not used</th>
</tr>
</thead>
<tbody>
<tr>
<td>buttons</td>
<td>Core</td>
<td>5</td>
<td>Integer</td>
<td>Any value</td>
<td>Bit field, 1 (closed)</td>
<td>No action</td>
</tr>
<tr>
<td>device</td>
<td>Core</td>
<td>14</td>
<td>Integer</td>
<td>Logical input device, 0–15</td>
<td>None</td>
<td>Default device</td>
</tr>
<tr>
<td>xpos</td>
<td>Core</td>
<td>61</td>
<td>Integer</td>
<td>Any value</td>
<td>Current X value</td>
<td>No action</td>
</tr>
<tr>
<td>ypos</td>
<td>Core</td>
<td>67</td>
<td>Integer</td>
<td>Any value</td>
<td>Current Y value</td>
<td>No action</td>
</tr>
</tbody>
</table>

At least one parameter is required or an error is returned. If device is specified, at least one other parameter must be specified.

Description

Summary

xyGetInput returns the current position and button status of the XY-input device.
Section 9. XY-Input commands

xyGetInput

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 0–31.

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.

The ASCII interface returns the same bit field as an integer value. For the example above, the ASCII interfaces would return "5" (4 + 0 + 1).

Device parameter

The device parameter directs xyGetInput to the specified logical XY-input device regardless of the default device number as set by xySet device.

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).

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.

Notes

1. xyGetState tbuttons returns the number of buttons available on a device.
2. Trying to queue xyGetInput causes error 177 (Command cannot be queued) at the time of the attempt.

Returns

ASCII

On success: Comma-separated list of values for requested parameters as described above.

On failure: "ERROR n...".

Binary

On success: AX = 0. Values associated with requested parameters are 32-bit values of the types given in the binary parameter table above.

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

See also:

xyGetState, xySet.
xyGetlnput

Examples

ASCII

<table>
<thead>
<tr>
<th>Get current X position</th>
<th>xyGetlnput xpos</th>
<th>(returns) &quot;43&quot; ; the current X coordinate is 43</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get current XY positions and state of buttons</td>
<td>xyGetlnput xpos,ypos,buttons</td>
<td>(returns) &quot;43,110,1&quot; ; XY coordinates are 43,110 and button 1 is closed</td>
</tr>
<tr>
<td>Get XY positions for input device 2</td>
<td>xyGetlnput device=2,xpos,ypos</td>
<td>(returns) &quot;128,145&quot; ; device 2 XY coordinates are 128, 145</td>
</tr>
</tbody>
</table>

Binary

<table>
<thead>
<tr>
<th>Get current XY and buttons values</th>
<th>AX 4100 ; xyGetlnput decimal ID</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>BX 3 ; number of parameters</td>
</tr>
<tr>
<td></td>
<td>ES:DI[0] 61 ; xpos decimal ID</td>
</tr>
<tr>
<td></td>
<td>ES:DI[4] any value ; place holder for xpos value after return</td>
</tr>
<tr>
<td></td>
<td>ES:DI[8] 67 ; ypos decimal ID</td>
</tr>
<tr>
<td></td>
<td>ES:DI[10] 5 ; buttons decimal ID</td>
</tr>
<tr>
<td></td>
<td>ES:DI[14] any value ; place holder for button bit field after return</td>
</tr>
<tr>
<td>After return</td>
<td>AX 0 ; returns 0 if successful (nonzero if not)</td>
</tr>
<tr>
<td></td>
<td>ES:DI[4] X position ; X position</td>
</tr>
<tr>
<td></td>
<td>ES:DI[8] Y position ; Y position</td>
</tr>
<tr>
<td></td>
<td>ES:DI[14] bit field ; button status bit field</td>
</tr>
</tbody>
</table>

9-6 Release R 1.0 April 15, 1990
### Section 9. XY-input commands

**xyGetState**

Last revision: R 1.0  
Type: core

#### Parameters

<table>
<thead>
<tr>
<th>ASCII</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Parameter</strong></td>
<td><strong>Core or extended</strong></td>
</tr>
<tr>
<td>cursor</td>
<td>Extended</td>
</tr>
<tr>
<td>defdevice</td>
<td>Core</td>
</tr>
<tr>
<td>device</td>
<td>Core</td>
</tr>
<tr>
<td>tbuttons</td>
<td>Core</td>
</tr>
<tr>
<td>tdevices</td>
<td>Core</td>
</tr>
<tr>
<td>xmax</td>
<td>Core</td>
</tr>
<tr>
<td>xmaxclip</td>
<td>Core</td>
</tr>
<tr>
<td>xmin</td>
<td>Core</td>
</tr>
<tr>
<td>xminclip</td>
<td>Core</td>
</tr>
<tr>
<td>ymax</td>
<td>Core</td>
</tr>
<tr>
<td>ymaxclip</td>
<td>Core</td>
</tr>
<tr>
<td>ymin</td>
<td>Core</td>
</tr>
<tr>
<td>yminclip</td>
<td>Core</td>
</tr>
</tbody>
</table>

At least one parameter is required or an error is returned. If device is specified, at least one other parameter must be specified.
xyGetState

Binary

Command code: 4102 decimal.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Core or extended</th>
<th>Token number (decimal)</th>
<th>Type</th>
<th>Associated calling value</th>
<th>Associated return value</th>
<th>Default if parameter not used</th>
</tr>
</thead>
<tbody>
<tr>
<td>cursor</td>
<td>Extended</td>
<td>11</td>
<td>Integer</td>
<td>Any value</td>
<td>1 (on)</td>
<td>0 (off)</td>
</tr>
<tr>
<td>defdevice</td>
<td>Core</td>
<td>12</td>
<td>Integer</td>
<td>Any value</td>
<td>Default input device, 0–15</td>
<td>No action</td>
</tr>
<tr>
<td>device</td>
<td>Core</td>
<td>14</td>
<td>Integer</td>
<td>Logical input device, 0–15</td>
<td>None</td>
<td>Default device</td>
</tr>
<tr>
<td>tbuttons</td>
<td>Core</td>
<td>44</td>
<td>Integer</td>
<td>Any value</td>
<td>Total available for device</td>
<td>No action</td>
</tr>
<tr>
<td>tdevlces</td>
<td>Core</td>
<td>45</td>
<td>Integer</td>
<td>Any value</td>
<td>Total installed for, 0–15</td>
<td>No action</td>
</tr>
<tr>
<td>xmax</td>
<td>Core</td>
<td>56</td>
<td>Integer</td>
<td>Any value</td>
<td>Maximum possible X value</td>
<td>No action</td>
</tr>
<tr>
<td>xmaxclip</td>
<td>Core</td>
<td>57</td>
<td>Integer</td>
<td>Any value</td>
<td>Current maximum X value</td>
<td>No action</td>
</tr>
<tr>
<td>xmin</td>
<td>Core</td>
<td>58</td>
<td>Integer</td>
<td>Any value</td>
<td>Minimum possible X value</td>
<td>No action</td>
</tr>
<tr>
<td>xminclip</td>
<td>Core</td>
<td>59</td>
<td>Integer</td>
<td>Any value</td>
<td>Current minimum X value</td>
<td>No action</td>
</tr>
<tr>
<td>ymax</td>
<td>Core</td>
<td>62</td>
<td>Integer</td>
<td>Any value</td>
<td>Maximum possible Y value</td>
<td>No action</td>
</tr>
<tr>
<td>ymaxclip</td>
<td>Core</td>
<td>63</td>
<td>Integer</td>
<td>Any value</td>
<td>Current maximum Y value</td>
<td>No action</td>
</tr>
<tr>
<td>ymin</td>
<td>Core</td>
<td>64</td>
<td>Integer</td>
<td>Any value</td>
<td>Minimum possible Y value</td>
<td>No action</td>
</tr>
<tr>
<td>yminclip</td>
<td>Core</td>
<td>65</td>
<td>Integer</td>
<td>Any value</td>
<td>Current minimum Y value</td>
<td>No action</td>
</tr>
</tbody>
</table>

At least one parameter is required or an error is returned. If device is specified, at least one other parameter must be specified.

Description

Summary

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.
**Section 9. XY-Input commands**

**xyGetState**

**Cursor parameter**  The *cursor* parameter returns one if the graphics cursor is visible. *Cursor* returns zero 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 48 (Unknown parameter).

**Defdevice parameter**  The *defdevice* parameter returns the logical number of the default XY-input device as by *xySet defdevice*. VDI management directs all XY commands to this device unless a command includes a *device parameter* (see below) directing it to a different input device.

**Device parameter**  The *device* parameter directs *xyGetState* to the specified logical device number regardless of the default device number as set by *xySet defdevice*. Because, in general, *device* affects the command with which it is associated only, the parameter does not affect the return value for *defdevice* (see above) when the two parameters are used together.

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).

**Tbuttons parameter**  The *tbuttons* parameter returns the total number of buttons available for the default or specified XY-input device.

**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, for example a mouse and touch screen.

**Xmin, ymin, xmax, ymax parameters**  The *xmin*, *ymin*, *xmax*, *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.
xyGetState

Xminclip, yminclip, xmaxclip, and ymaxclip parameters

The xminclip, yminclip, xmaxclip, and ymaxclip parameters return the area within the XY-coordinate space within which changes of position are reported.

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.

Notes

1. Trying to queue xyGetState causes error 177 (Command cannot be queued) at the time of the attempt.

Returns

ASCII

On success: Comma-separated list of values for requested parameters as described above.

On failure: "ERROR n...".

Binary

On success: AX = 0. Values associated with requested parameters are 32-bit values of the types given in the binary parameter table above.

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

See also:

syGetState, vdGetState, vmGetState, xyGetInput, xyInit, xySet.

Examples

ASCII

Get total devices, and default device

xyGetState tdevices,defdevice

(return "2,1" ; two devices, number 1 is selected

Get current XY-coordinate space

xyGetState xmin,ymin,xmax,ymax

(return "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

(return "3" ; three buttons available
Section 9. XY-input commands

xyGetState

Binary

<table>
<thead>
<tr>
<th>Determine If cursor is on</th>
<th>AX</th>
<th>4102</th>
<th>; xyGetState decimal ID</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>BX</td>
<td>1</td>
<td>; number of parameters</td>
</tr>
<tr>
<td>ES:DI[0]</td>
<td>11</td>
<td></td>
<td>; cursor decimal ID</td>
</tr>
<tr>
<td>ES:DI[4]</td>
<td>any value</td>
<td></td>
<td>; place holder for cursor value after return</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>After return</th>
<th>AX</th>
<th>0</th>
<th>; returns 0 if successful (nonzero if not)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ES:DI[4]</td>
<td>1</td>
<td>; graphics cursor is on (0 = off)</td>
</tr>
</tbody>
</table>
xyInit

Parameters

ASCII

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Core or extended</th>
<th>Associated calling value</th>
<th>Type as ASCII</th>
<th>Associated return value</th>
<th>Type as ASCII</th>
<th>Default if parameter not used</th>
</tr>
</thead>
<tbody>
<tr>
<td>device</td>
<td>Core</td>
<td>Logical input device, 0-15</td>
<td>Integer</td>
<td>None</td>
<td>N/A</td>
<td>Default device</td>
</tr>
</tbody>
</table>

This command can be issued with no parameters.

Binary

Command code: 4103 decimal.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Core or extended</th>
<th>Token number (decimal)</th>
<th>Type</th>
<th>Associated calling value</th>
<th>Associated return value</th>
<th>Default if parameter not used</th>
</tr>
</thead>
<tbody>
<tr>
<td>device</td>
<td>Core</td>
<td>14</td>
<td>Integer</td>
<td>Logical input device, 0-15</td>
<td>None</td>
<td>Default device</td>
</tr>
</tbody>
</table>

This command can be issued with no parameters.

Description

Summary

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.

Device parameter

The device parameter specifies the logical number of the XY-input device to be initialized. If device is omitted, xyInit initializes the default device as set by xySet defdevice. If xySet has not been used to set a default input device, the default device is defined to be number zero. xyInit does not change the default input device if a device other than the default is specified. To change the default, use xySet defdevice.

Specifying a nonexistent or uninstalled device returns error 160 (Invalid device number).
Section 9. XY-input commands

xyInit

Conditions set by xyInit

xyInit sets the parameters in the following table to the specified values.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Command reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>tbuttons</td>
<td>Total available for device being initialized</td>
<td>none</td>
</tr>
<tr>
<td>tdevices</td>
<td>Total installed, 0–15</td>
<td>none</td>
</tr>
<tr>
<td>xmax</td>
<td>639</td>
<td>xySet</td>
</tr>
<tr>
<td>xmaxclip</td>
<td>639</td>
<td>xySet</td>
</tr>
<tr>
<td>xmin</td>
<td>0</td>
<td>xySet</td>
</tr>
<tr>
<td>xminclip</td>
<td>0</td>
<td>xySet</td>
</tr>
<tr>
<td>xpos</td>
<td>0</td>
<td>xySet</td>
</tr>
<tr>
<td>ymax</td>
<td>199</td>
<td>xySet</td>
</tr>
<tr>
<td>ymaxclip</td>
<td>199</td>
<td>xySet</td>
</tr>
<tr>
<td>ylin</td>
<td>0</td>
<td>xySet</td>
</tr>
<tr>
<td>ylinclip</td>
<td>0</td>
<td>xySet</td>
</tr>
<tr>
<td>ypos</td>
<td>0</td>
<td>xySet</td>
</tr>
</tbody>
</table>

Notes

1. `xyGetState tdevices` returns the total number of 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.

2. Trying to queue `xyInit` causes error 177 (Command cannot be queued) at the time of the attempt.

Returns

ASCII

On success: "OK".

On failure: "ERROR n...".

Binary

On success: AX = 0.

On failure: AX = error number.

See also: `xyInit, vdInit, vmInit, xyGetState, xySet`.

April 15, 1990
Release R 1.0

9–13
Recommended Practices for Interactive Video Portability

**xylInit**

**Examples**

**ASCII**

<table>
<thead>
<tr>
<th>Initialize device 0 and xy service group</th>
<th>xylInit</th>
<th>(returns) &quot;OK&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>; first time command is issued</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Initialize device 1</th>
<th>xylInit device=1</th>
<th>(returns) &quot;OK&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>; device 1 successfully initialized</td>
</tr>
</tbody>
</table>

**Binary**

<table>
<thead>
<tr>
<th>Initialize device 0 and xy service group</th>
<th>AX</th>
<th>4103</th>
<th>; xylInit decimal ID</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>BX</td>
<td>0</td>
<td>; number of parameters</td>
</tr>
</tbody>
</table>

| After return | AX | 0    | ; returns 0 if successful (nonzero if not) |

9-14 Release R 1.0 April 15, 1990
### xySet

**Parameters**

**ASCII**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Core or extended</th>
<th>Associated calling value</th>
<th>Type as ASCII</th>
<th>Associated return value</th>
<th>Type as ASCII</th>
<th>Default if parameter not used</th>
</tr>
</thead>
<tbody>
<tr>
<td>cursor</td>
<td>Extended</td>
<td>1 (on)</td>
<td>0 (off)</td>
<td>Integer</td>
<td>None</td>
<td>N/A</td>
</tr>
<tr>
<td>defdevice</td>
<td>Core</td>
<td>Logical input device, 0–15</td>
<td>Integer</td>
<td>None</td>
<td>N/A</td>
<td>No action</td>
</tr>
<tr>
<td>device</td>
<td>Core</td>
<td>Logical input device, 0–15</td>
<td>Integer</td>
<td>None</td>
<td>N/A</td>
<td>Default device</td>
</tr>
<tr>
<td>xmax</td>
<td>Core</td>
<td>Maximum possible X value</td>
<td>Integer</td>
<td>None</td>
<td>N/A</td>
<td>No action</td>
</tr>
<tr>
<td>xmaxclip</td>
<td>Core</td>
<td>Current maximum X value</td>
<td>Integer</td>
<td>None</td>
<td>N/A</td>
<td>No action</td>
</tr>
<tr>
<td>xmin</td>
<td>Core</td>
<td>Minimum possible X value</td>
<td>Integer</td>
<td>None</td>
<td>N/A</td>
<td>No action</td>
</tr>
<tr>
<td>xminclip</td>
<td>Core</td>
<td>Current minimum X value</td>
<td>Integer</td>
<td>None</td>
<td>N/A</td>
<td>No action</td>
</tr>
<tr>
<td>xpos</td>
<td>Core</td>
<td>X position</td>
<td>Integer</td>
<td>None</td>
<td>N/A</td>
<td>No action</td>
</tr>
<tr>
<td>ymax</td>
<td>Core</td>
<td>Maximum possible Y value</td>
<td>Integer</td>
<td>None</td>
<td>N/A</td>
<td>No action</td>
</tr>
<tr>
<td>ymaxclip</td>
<td>Core</td>
<td>Current maximum Y value</td>
<td>Integer</td>
<td>None</td>
<td>N/A</td>
<td>No action</td>
</tr>
<tr>
<td>ymIn</td>
<td>Core</td>
<td>Minimum possible Y value</td>
<td>Integer</td>
<td>None</td>
<td>N/A</td>
<td>No action</td>
</tr>
<tr>
<td>yminclip</td>
<td>Core</td>
<td>Current minimum Y value</td>
<td>Integer</td>
<td>None</td>
<td>N/A</td>
<td>No action</td>
</tr>
<tr>
<td>ypos</td>
<td>Core</td>
<td>Y position</td>
<td>Integer</td>
<td>None</td>
<td>N/A</td>
<td>No action</td>
</tr>
</tbody>
</table>

At least one parameter is required or an error is returned. If device is specified, at least one other parameter must be specified.
xySet

Binary

Command code: 4109 decimal.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Core or extended</th>
<th>Token number (decimal)</th>
<th>Type</th>
<th>Associated calling value</th>
<th>Associated return value</th>
<th>Default if parameter not used</th>
</tr>
</thead>
<tbody>
<tr>
<td>cursor</td>
<td>Extended</td>
<td>11</td>
<td>Integer</td>
<td>1 (on)</td>
<td>0 (off)</td>
<td>None</td>
</tr>
<tr>
<td>defdevice</td>
<td>Core</td>
<td>12</td>
<td>Integer</td>
<td>Logical input device, 0–15</td>
<td>None</td>
<td>No action</td>
</tr>
<tr>
<td>device</td>
<td>Core</td>
<td>14</td>
<td>Integer</td>
<td>Logical input device, 0–15</td>
<td>None</td>
<td>Default device</td>
</tr>
<tr>
<td>xmax</td>
<td>Core</td>
<td>56</td>
<td>Integer</td>
<td>Maximum possible X value</td>
<td>None</td>
<td>No action</td>
</tr>
<tr>
<td>xmaxclip</td>
<td>Core</td>
<td>57</td>
<td>Integer</td>
<td>Current maximum X value</td>
<td>None</td>
<td>No action</td>
</tr>
<tr>
<td>xmin</td>
<td>Core</td>
<td>58</td>
<td>Integer</td>
<td>Minimum possible X value</td>
<td>None</td>
<td>No action</td>
</tr>
<tr>
<td>xminclip</td>
<td>Core</td>
<td>59</td>
<td>Integer</td>
<td>Current minimum X value</td>
<td>None</td>
<td>No action</td>
</tr>
<tr>
<td>xpos</td>
<td>Core</td>
<td>61</td>
<td>Integer</td>
<td>X position</td>
<td>None</td>
<td>No action</td>
</tr>
<tr>
<td>ymax</td>
<td>Core</td>
<td>62</td>
<td>Integer</td>
<td>Maximum possible Y value</td>
<td>None</td>
<td>No action</td>
</tr>
<tr>
<td>ymaxclip</td>
<td>Core</td>
<td>63</td>
<td>Integer</td>
<td>Current maximum Y value</td>
<td>None</td>
<td>No action</td>
</tr>
<tr>
<td>ymin</td>
<td>Core</td>
<td>64</td>
<td>Integer</td>
<td>Minimum possible Y value</td>
<td>None</td>
<td>No action</td>
</tr>
<tr>
<td>yminclip</td>
<td>Core</td>
<td>65</td>
<td>Integer</td>
<td>Current minimum Y value</td>
<td>None</td>
<td>No action</td>
</tr>
<tr>
<td>ypos</td>
<td>Core</td>
<td>67</td>
<td>Integer</td>
<td>Y position</td>
<td>None</td>
<td>No action</td>
</tr>
</tbody>
</table>

At least one parameter is required or an error is returned. If device is specified, at least one other parameter must be specified.

Description

Summary  xySet defines the XY-coordinate space, sets the default input device, turns the cursor on and off, and sets the current XY coordinates. Each parameter stays in effect for either the current or specified device regardless of the graphics mode until reset with xySet or xyInit.
Section 9. XY-Input commands

xySet

**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. If the input device does not support a cursor, turning the cursor on returns error 87 (Action not supported by device).

> **Cursor** is an extended parameter. Using an unimplemented extended parameter causes error 48 (Unknown parameter).

**Defdevice parameter** The defdevice parameter specifies the default input device to be used when more than one input device is available. Specifying a nonexistent or uninstalled device returns error 160 (Invalid device number). Specifying an uninitialized device causes error 81 (Device not initialized).

**Device parameter** The device parameter directs xySet to the specified logical device number regardless of the default input device as set by xySet defdevice (see above).

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).

**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 -32768–32767. 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.

**Xminclip, yminclip, xmaxclip, and ymaxclip parameters** The xminclip, yminclip, xmaxclip, and ymaxclip parameters define a constrained area within the coordinate space for reporting coordinate movement. Coordinates values are returned only within the defined clip 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 above) returns error 51 (Parameter value invalid or out of range).

**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 the clipping values (see above) sets the coordinate to the limit of the respective clipping range.
xySet

Notes
1. xyGetState tdevices returns the number of input devices that can be selected by xySet defdevice, assuming all devices for which VDI Management was installed are available.

Returns

ASCII
On success: “OK”.
On failure: “ERROR n...”.

Binary
On success: AX = 0.
On failure: AX = error number.

See also: xyGetInput, xyGetState, xyInit.

Examples

ASCII
Set XY position
xySet xpos=40,ypos=50
(returns) “OK” ; XY position is 40,50

Turn device 2
device=2,cursor=1
cursor on
xySet xmin=0,ymin=0,xmax=639,ymax=479
(returns) “OK”

Match coordinate
space to VGA max
xySet xmaxclip=200,ymaxclip=200
(returns) “OK” ; report change if X or Y is less than 200

Binary examples

Turn on XY cursor
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 (nonzero if not)
This appendix explains how to determine the size and position of graphics relative to background video. To ensure the compatibility of hardware, VDI Management software, and 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).

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 sections explain how to determine horizontal and vertical graphics positions for both NTSC and PAL video.

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, the recommended positions in this appendix are guidelines for a nominal video image. 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.

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.
A.1 Terms of reference

Figure A-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.

The remaining sections of this appendix use the following terms:

1. **Active graphics**: The portion of the screen where graphics can appear.
2. **Active video**: The portion of the screen where video can appear. This is the portion of horizontal and vertical video not blanked by horizontal and verticalblanking.
3. **Border**: The portion of active video not covered by active graphics.
4. **Horizontal blanking**: The time period during which the display is blank for horizontal retracement.
Appendix A. Default positions of graphics relative to video

5. **Horizontal sync**: The pulse used to synchronize the horizontal scan of the video monitor.

6. **Vertical blanking**: The time period during which the display is blank for vertical retracement.

7. **Vertical sync**: The pulse used to synchronize the vertical scan of the video monitor.

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–170A and CCIR 470–1, respectively.

A.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.

A.2.1 Differences in signals and timing

CGA and EGA graphics overlay systems typically use standard 15-kHz background video. For these systems, accepted video standards dictate blanking interval widths and the starts of horizontal and vertical sync. Therefore, CGA and EGA systems can use the starts of horizontal and vertical sync as absolute references for graphics positioning and rely on constant horizontal signal widths and vertical timing for a given video standard, either NTSC or PAL.

However, 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 start of sync and positioning must be relative to the active video rather than the total video including blanking.

A.2.2 Differences in the size of active graphics

By increasing pixel width, VGA graphics cover the entire width of active video, leaving borders at the top and bottom of active graphics only. 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 × 200) graphics cover the entire width of the active video, while mode 6 (640 × 200) graphics from a CGA or EGA adapter leave a visible video border around all edges of the active graphics. Therefore, compliant VDI Management implementations for VGA overlay systems must
Recommended Practices for Interactive Video Portability

support both graphics that map to the left and right edges of active video and, for those modes that are CGA/EGA-compatible, emulations of true CGA and EGA systems that leave a border around all edges of active graphics.

A.3 Horizontal positions

This section explains how to determine the start and end of active graphics relative to a horizontal video signal. 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.

Because horizontal sync is the timing reference from which all horizontal components are measured in 15-kHz video, true CGA and EGA graphics use the start of horizontal sync as a reference and 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 expressed as a proportion of active H.

A.3.1 General assumptions

The following general assumptions are used in determining the horizontal positions of active graphics relative to NTSC and PAL video.

1. The optimal position for active graphics is centered horizontally in the active video.

   Basis: Nominal common practice. However, variations in graphics positioning due to monitor centering adjustments cannot be accounted for in the recommendations in this appendix.

2. One horizontal line of graphics is 912 pixels in length. The active graphics consist of a 640-pixel window.

   Basis: Apple II and IBM CGA standards define a 640-pixel active graphics window in a 912-pixel horizontal line. The graphics are about 85% of the total displayable window to allow for monitor overscan. This has become a de facto standard that is also used by EGA graphics. CGA- and EGA-based overlay systems generally follow this standard.

3. 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

   Basis: Original IBM CGA implementation.

3. Graphics with 320-pixel active areas cover exactly the same horizontal area as graphics with 640-pixel active areas. Therefore, starting and
Appendix A. Default positions of graphics relative to video

ending positions based on calculations using 640 pixels are valid for 320-pixel graphics.

**Basis:** 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.

A.3.2 NTSC

Figure A–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 from or derived from the accepted standard for NTSC video. The calculations in this section are based on the values shown in the figure.

**Figure A–2.**

One horizontal line of NTSC video with 640- or 320-pixel overlayed graphics

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 given the timing shown in Figure A–2, use the following equations and assumptions for 640-pixel graphics. Note that the resulting positions hold for 320-pixel graphics.

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:

\[
GH_{\text{start}} = \frac{AV_{\text{start}} + \frac{(P_{\text{active}} - P_{\text{displayed}}) \times P_{\text{width}}}{2}}{H_{\text{total}}}
\]

(1)
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:

\[ \text{GH}_{\text{end}} = \frac{A_{\text{start}} + \frac{(P_{\text{active}} - P_{\text{displayed}}) \times P_{\text{width}}}{2} + (P_{\text{displayed}} \times P_{\text{width}})}{H_{\text{total}}} \]

Where, GH stands for "graphics horizontal," and for NTSC video and 640-pixel resolutions:

1. The nominal start of active video is:
   \[ A_{\text{start}} = 9.4 \mu s \]
   from Figure A-2.

2. The total number of pixels corresponding to the width of active video is:
   \[ P_{\text{active}} = \frac{52.656 \mu s \times 912 \text{ pixels}}{63.556 \mu s} = 756 \text{ pixels} \]
   from Figure A-2 and the de facto standard (see assumption 2 in Section A.3.1).

3. The total number of displayed pixels is:
   \[ P_{\text{displayed}} = 640 \text{ pixels} \]
   from the de facto standard (see assumption 2 in Section A.3.1).

4. The total width of the video signal is:
   \[ H_{\text{total}} = 63.556 \mu s \]
   from Figure A-2.

5. The width of one pixel is
   \[ P_{\text{width}} = \frac{63.556 \mu s}{912} = 0.06969 \mu s \]

Solving equation 1 for the starting position of 640-pixel graphics yields:

\[ G_{\text{Hstart}} = \frac{9.4 + (756 - 640) \times 0.06969}{2} = 0.2115 H_{\text{total}} \]

Solving equation 2 for the ending position of 640-pixel graphics yields:

\[ G_{\text{Hend}} = \frac{9.4 + (756 - 640) \times 0.06969 + (640 \times 0.06969)}{2} = 0.9133 H_{\text{total}} \]
For NTSC video these values equate to left and right border widths of 4.042 µs and a start of active graphics at approximately 13.4 µs after the start of horizontal sync. The latter value can be reliably verified with an accurate oscilloscope.

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 A–2, use the following equations and assumption for 640-pixel graphics. Note that the resulting positions hold for 320-pixel graphics.

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:

\[
GH_{\text{start}} = \frac{(P_{\text{active}} - P_{\text{displayed}}) \times \text{width}}{2H_{\text{active}}}
\]

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:

\[
GH_{\text{end}} = \frac{(P_{\text{active}} - P_{\text{displayed}}) \times \text{width} + (P_{\text{displayed}} \times \text{width})}{2H_{\text{active}}}
\]

These equations are identical to equations 1 and 2 in the previous section except that \(AV_{\text{start}}\) is now equal to zero and therefore dropped from the equations and that \(H_{\text{total}}\) has been changed to \(H_{\text{active}}\) where:

\[
H_{\text{active}} = 52.656 \, \mu s
\]

from Figure A–2.

Solving equation 3 for the starting position of 640-pixel graphics yields:

\[
GH_{\text{start}} = \frac{(756 - 640) \times 0.06969}{52.656} = 0.0768 \, \frac{H_{\text{active}}}{H_{\text{active}}}
\]
Solving equation 4 for the ending position of 640-pixel graphics yields:

\[ GH_{\text{end}} = \frac{(756 - 640) \times 0.06969}{2} + (640 \times 0.06969) \]
\[ = \frac{52.656}{52.656} = 0.924 H_{\text{active}} \]

**A.3.3 PAL**

Figure A–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 timing values in the figure are either taken directly from or derived from the accepted standard for PAL video. The calculations in this section are based on the values shown in the figure.

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.

**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 A–3, use the following equations and assumptions for 640-pixel graphics. Note that the resulting positions hold for 320-pixel graphics.
Appendix A. Default positions of graphics relative to video

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 horizontal sync as a reference can be expressed as:

\[
GH_{\text{start}} = \frac{AV_{\text{start}} + \frac{(P_{\text{active}} - P_{\text{displayed}}) \times P_{\text{width}}}{2}}{H_{\text{total}}}
\]

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:

\[
GH_{\text{end}} = \frac{AV_{\text{end}} + \frac{(P_{\text{active}} - P_{\text{displayed}}) \times P_{\text{width}}}{2} + (P_{\text{displayed}} \times P_{\text{width}})}{H_{\text{total}}}
\]

Where, for PAL video and 640-pixel resolutions:

1. The nominal start of active video is:
   \[AV_{\text{start}} = 10.5 \mu s\]
   from Figure A–3.

2. The total number of pixels corresponding to the width of active video is:
   \[P_{\text{active}} = \frac{51.95 \mu s}{64.0 \mu s} \times 912 \text{ pixels} = 740 \text{ pixels}\]
   from Figure A–3 and the de facto standard (see assumption 2 in Section A.3.1).

3. The total number of displayed pixels is:
   \[P_{\text{displayed}} = 640 \text{ pixels}\]
   from the de facto standard (see assumption 2 in Section A.3.1).

4. The total width of the video signal is
   \[H_{\text{total}} = 64.0 \mu s\]
   from Figure A–3.

5. The width of one pixel is
   \[P_{\text{width}} = \frac{64.0 \mu s}{912} = 0.07018 \mu s\]

Solving equation 5 for the starting position of 640-pixel graphics yields:

\[
GH_{\text{start}} = \frac{10.5 + \frac{(740 - 640) \times 0.07018}{2}}{64.0} = 0.2189 H_{\text{total}}
\]
Recommended Practices for Interactive Video Portability

Solving equation 6 for the ending position of 640-pixel graphics yields:

\[
GH_{\text{end}} = \frac{10.5 + \frac{(740 - 640) \times 0.07018}{2} + (640 \times 0.07018)}{64.0} = 0.9207 \cdot H_{\text{total}}
\]

For PAL video these values equate to left and right border widths of 3.509 μs and a start of active graphics at approximately 14.0 μs after the start of horizontal sync. The latter value can be reliably verified with an accurate oscilloscope.

**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 A-3, use the following equations and assumptions for 640-pixel graphics. Note that the resulting positions hold for 320-pixel graphics.

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:

\[
GH_{\text{start}} = \frac{(P_{\text{active}} - P_{\text{displayed}}) \times P_{\text{width}}}{2} \cdot H_{\text{active}}
\]

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:

\[
GH_{\text{end}} = \frac{(P_{\text{active}} - P_{\text{displayed}}) \times P_{\text{width}}}{2} + (P_{\text{displayed}} \times P_{\text{width}}) \cdot H_{\text{active}}
\]

These equations are identical to equations 5 and 6 in the previous section except that \(A V_{\text{start}}\) is now equal to zero and therefore dropped from the equations and that \(H_{\text{total}}\) has been changed to \(H_{\text{active}}\) where:

\[H_{\text{active}} = 51.95 \mu s\]

from Figure A-3.
Solving equation 7 for the starting position of 640-pixel graphics yields:

\[
GH_{\text{start}} = \frac{(740 - 640) \times 0.07018}{2} = 0.0675 H_{\text{active}}
\]

Solving equation 8 for the ending position of 640-pixel graphics yields:

\[
GH_{\text{end}} = \frac{(740 - 640) \times 0.07018}{2} + (640 \times 0.07018) = 0.9321 H_{\text{active}}
\]

A.4 Vertical positions

This section explains 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.

A.4.1 General assumptions

The following general assumptions are used in determining the vertical positions of active graphics relative to NTSC and PAL video.

1. The optimal position for active graphics is centered vertically in the active video.

   **Basis:** Nominal common practice. However, note that variations in graphics positioning due to monitor centering adjustments cannot be accounted for in the recommendations in this appendix.

2. Vertical resolution is expressed relative to fields, not frames. VGA 480-line mode converts to 240 lines for single-field for computations.

   **Basis:** Observation.

A.4.2 NTSC

Figure A-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 from or derived from the accepted standard for NTSC video. The calculations in this section are based on the values shown in the figure.
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 the start of vertical sync for a 15.734-kHz, NTSC, video signal given the timing shown in Figure A-4, use the following equations and assumptions. Note that the calculations can easily be modified for different numbers of lines.

Note: 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.

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:

\[
GV_{\text{start}} = AV_{\text{start}} + \frac{(V_{\text{active}} - G_{\text{active}})}{2}
\]

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:

\[
GV_{\text{end}} = AV_{\text{start}} + \frac{(V_{\text{active}} - G_{\text{active}})}{2} + G_{\text{active}}
\]

Where GV stands for "graphics vertical," and for NTSC video:

1. The nominal start of active video is:

\[
AV_{\text{start}} = 18 \text{ lines}
\]

from Figure A-4.
2. The number of active video lines is:
   \[ V_{active} = 241.5 \text{ lines} \]
   from Figure A-4.

3. The number of active graphics lines is either:
   \[ G_{active} = 200 \text{ lines} \]
   for \(640 \times 200\) and \(320 \times 200\) graphics, or:
   \[ G_{active} = 240 \text{ lines} \]
   for \(640 \times 480\) graphics (see assumption 2 in Section A.4.1.)

**Starting and ending positions for 200-line graphics**

Solving equation 9 for the start of 200-line graphics yields:

\[ GV_{start} = 18 + \frac{241.5 - 200}{2} = 39 \text{ lines} \]

Solving equation 10 for the end of 200-line graphics yields

\[ GV_{end} = 18 + \frac{241.5 - 200}{2} + 200 = 239 \text{ lines} \]

**Starting and ending positions for 240-line (640 \times 480) graphics**

Solving equation 9 for the start of 240-line graphics yields:

\[ GV_{start} = 18 + \frac{241.5 - 240}{2} = 18.75 \text{ lines} \]

Solving equation 10 for the end of 240-line graphics yields

\[ GV_{end} = 18 + \frac{241.5 - 240}{2} + 240 = 258.75 \text{ lines} \]

Given the derived values above, VGA 240-line modes actually leave borders 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.

**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:

\[ GV_{start} = \frac{AV_{start} + \frac{(V_{active} - G_{active})}{2}}{V_{total}} \]
Recommended Practices for Interactive Video Portability

\[
GV_{end} = AV_{start} + \frac{(V_{active} - G_{active})}{2} + G_{active} \div V_{total}
\]

(12)

All terms are defined in the previous section except \(V_{total}\), which for NTSC video is:

\[V_{total} = 262.5 \text{ lines}\]

from Figure A-4.

**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:

\[GV_{start} = \frac{39}{262.5} = 0.1486 \, V_{total}\]

\[GV_{end} = \frac{239}{262.5} = 0.9105 \, V_{total}\]

**Starting and ending positions for 240-line (640 x 480) graphics**

Borrowing from the previous section, the starting and ending positions for 240-line graphics as proportions of total V are simply:

\[GV_{start} = \frac{18.75}{262.5} = 0.0714 \, V_{total}\]

\[GV_{end} = \frac{258.75}{262.5} = 0.9857 \, V_{total}\]

**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:

\[
GV_{start} = \frac{(V_{active} - G_{active})}{2} \div V_{active}
\]

(13)

\[
GV_{end} = \frac{(V_{active} - G_{active}) + G_{active}}{2} \div V_{active}
\]

(14)
All terms are defined in previous sections except $V_{active}$, which for NTSC video is:

$$V_{active} = 241.5 \text{ lines}$$

from Figure A-4.

**Starting and ending positions for 200-line graphics**

Borrowing from previous sections, the starting and ending positions for 200-line graphics as proportions of active $V$ are simply:

$$GV_{start} = \frac{2}{241.5} = 0.0859 \cdot V_{active}$$

$$GV_{end} = \frac{241.5 - 200}{241.5} = 0.9141 \cdot V_{active}$$

**Starting and ending positions for 240-line (640 x 480) graphics**

Borrowing from previous sections, the starting and ending positions for 240-line graphics as proportions of active $V$ are simply:

$$GV_{start} = \frac{2}{241.5} = 0.0031 \cdot V_{active}$$

$$GV_{end} = \frac{241.5 - 240}{241.5} = 0.9969 \cdot V_{active}$$

**A.4.3 PAL**

Figure A-5 shows the timing for 15-kHz PAL video including the position of 200-line graphics. The timing values in the figure are either taken directly from or derived from the accepted standard for PAL video. The calculations in this section are based on the values shown in the figure.
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 section.

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 given the timing shown in Figure A-5, use the following equations and assumptions. Note that the calculations can easily be modified for different numbers of graphics lines.

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:

\[
GV_{\text{start}} = AV_{\text{start}} + \frac{(V_{\text{active}} - G_{\text{active}})}{2}
\]

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:

\[
GV_{\text{end}} = AV_{\text{start}} + \frac{(V_{\text{active}} - G_{\text{active}})}{2} + G_{\text{active}}
\]

Where GV stands for "graphics vertical," and for PAL video:

1. The nominal start of active video is:

\[
AV_{\text{start}} = 22.5 \text{ lines}
\]

from Figure A-5.
2. The number of active video lines is:

\[ V_{\text{active}} = 287.5 \text{ lines} \]

from Figure A-5.

3. The number of active graphics lines is either:

\[ G_{\text{active}} = 200 \text{ lines} \]

for 640 x 200 and 320 x 200 graphics, or:

\[ G_{\text{active}} = 240 \text{ lines} \]

for 640 x 480 graphics and various special PAL modes (see assumption 2 in Section A.4.1.)

**Starting and ending positions for 200-line graphics**

Solving equation 15 for the start of 200-line graphics yields:

\[ GV_{\text{start}} = 22.5 + \frac{287.5 - 200}{2} = 66 \text{ lines} \]

Solving equation 16 for the end of 200-line graphics yields

\[ GV_{\text{end}} = 22.5 + \frac{287.5 - 200}{2} + 200 = 266 \text{ lines} \]

**Starting and ending positions for 240-line graphics**

Solving equation 15 for the start of 240-line graphics yields:

\[ GV_{\text{start}} = 22.5 + \frac{287.5 - 240}{2} = 46 \text{ lines} \]

Solving equation 16 for the end of 240-line graphics yields

\[ GV_{\text{end}} = 22.5 + \frac{287.5 - 240}{2} + 240 = 286 \text{ lines} \]

**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 15 and 16 to yield:

\[ GV_{\text{start}} = \frac{AV_{\text{start}} + (V_{\text{active}} - G_{\text{active}})}{2} \]

\[ GV_{\text{end}} = \frac{AV_{\text{end}} + (V_{\text{active}} - G_{\text{active}})}{2} \]
All terms are defined in previous sections except $V_{active}$, which for PAL video is:

$$V_{active} = 287.5 \text{ lines}$$

from Figure A-5.

**Starting and ending positions for 200-line graphics**

Borrowing from previous sections, the starting and ending positions for 240-line graphics as proportions of active $V$ are simply:

$$GV_{start} = \frac{287.5 - 200}{287.5} = 0.1522 \times V_{active}$$

$$GV_{end} = \frac{287.5 - 200 + 200}{287.5} = 0.8478 \times V_{active}$$

**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 simply:

$$GV_{start} = \frac{287.5 - 240}{287.5} = 0.0826 \times V_{active}$$

$$GV_{end} = \frac{287.5 - 240 + 240}{287.5} = 0.9174 \times V_{active}$$
Table B-1 lists standard graphics modes returned by BIOS interrupt 10H, service OFH for IBM and compatible personal computers. 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 non-standard manner may make a system noncompliant.

Note that modes 0–3 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.

Table B-1. IBM-compatible graphics modes

<table>
<thead>
<tr>
<th>Mode (^1) (decimal)</th>
<th>Type</th>
<th>Resolution</th>
<th>Colors</th>
<th>Overlay mode</th>
<th>Adapter (^2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0, 1</td>
<td>Text</td>
<td>40 x 25</td>
<td>16</td>
<td>Yes</td>
<td>CGA, EGA, MCGA, VGA</td>
</tr>
<tr>
<td>2, 3</td>
<td>Text</td>
<td>80 x 25</td>
<td>16</td>
<td>Yes</td>
<td>CGA, EGA, MCGA, VGA</td>
</tr>
<tr>
<td>4, 5</td>
<td>Graphics</td>
<td>320 x 200</td>
<td>4</td>
<td>Yes</td>
<td>CGA, EGA, MCGA, VGA</td>
</tr>
<tr>
<td>6</td>
<td>Graphics</td>
<td>640 x 200</td>
<td>2</td>
<td>Yes</td>
<td>CGA, EGA, MCGA, VGA</td>
</tr>
<tr>
<td>7</td>
<td>Text</td>
<td>80 x 25</td>
<td>Mono</td>
<td>No</td>
<td>EGA, VGA</td>
</tr>
<tr>
<td>13</td>
<td>Graphics</td>
<td>320 x 200</td>
<td>16</td>
<td>Yes</td>
<td>EGA, VGA</td>
</tr>
<tr>
<td>14</td>
<td>Graphics</td>
<td>640 x 200</td>
<td>16</td>
<td>Yes</td>
<td>EGA, VGA</td>
</tr>
<tr>
<td>15</td>
<td>Graphics</td>
<td>640 x 350</td>
<td>Mono</td>
<td>No</td>
<td>EGA, VGA</td>
</tr>
<tr>
<td>16</td>
<td>Graphics</td>
<td>640 x 350</td>
<td>16</td>
<td>No</td>
<td>EGA, VGA</td>
</tr>
<tr>
<td>17</td>
<td>Graphics</td>
<td>640 x 480</td>
<td>2</td>
<td>Yes</td>
<td>MCGA, VGA</td>
</tr>
<tr>
<td>18</td>
<td>Graphics</td>
<td>640 x 480</td>
<td>16</td>
<td>Yes</td>
<td>VGA</td>
</tr>
<tr>
<td>19</td>
<td>Graphics</td>
<td>320 x 200</td>
<td>256</td>
<td>Yes</td>
<td>MCGA, VGA</td>
</tr>
</tbody>
</table>

\(^1\) Does not include modes exclusive to the IBM PCjr and internal BIOS modes.

\(^2\) CGA = Color Graphics Adapter, EGA = Enhanced Graphics Adapter, MCGA = Multicolor Graphics Array, VGA = Video Graphics Array
Note: 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 mode 19 will not be portable to a compliant system that is limited to EGA modes.

Any system that requires applications to use a nonstandard graphics mode is noncompliant. Any application that uses a nonstandard mode is noncompliant.
C Application programming examples

This subsection gives 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. Therein lies the makings of another document.

C.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 below use the Microsoft GWBASIC interpreter.

For clarity and brevity, these programs do not determine which service groups are present or check for errors after issuing commands. However, well-behaved applications should do both.

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
Recommended Practices for Interactive Video Portability

Programming systems with library facilities for software interrupts can use direct software interrupt calls for issuing commands. The following example uses Microsoft C 5.1. It is a code fragment, not a complete program, and, as such, is not compilable.

```c
#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;
```

C.2 Using software interrupt calls

The next section of code would parse $R\$, which is in the form 

```plaintext
<value>,<value>,<value>
```

Now shut it down

```c
OPEN "O",#1,"ivdev"
PRINT #1,"syStop"
CLOSE #1
```

```
END
```
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 = COLORPARAM;
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);
C.3 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.

```c
#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 */

    ...
}
```

C-4 Release R 1.0 April 15, 1990
C.4 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
```
D Error handling

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 that VDI Management can return to an application.

D.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).

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. However, implementors should try to be complete and should not omit error codes simply for convenience.

VDI Management implementations may supply textual error messages. (See syErrorMsg in Section 6.) Although this is not a compliance requirement, if an implementation does support textual error messages, it must use the summary messages given in this appendix. Although this appendix presents summary messages in mixed case for legibility, VDI Management returns summary messages (and all other return strings) in all capital letters.

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.

1 Filing-system error codes are included primarily for use by future digital audio commands.
Note: VDI Management cannot handle the error of a user forgetting to install VDI Management or the correct interface used by an application. Applications should guard against this by confirming that VDI Management and the proper interface are installed (see Sections 3.2.2 and 3.3.2).

D.2 Error listings

The following subsections list error numbers in numerical order with summary messages and brief explanations. Related errors are grouped for convenience and do not necessarily imply a corresponding programming structure in VDI Management implementations. However, implementations should use the error numbers as they are defined.

D.2.1 Command problems

1 Service group not installed

The VDI 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.

2 Unknown command

The command does not exist.

Compliant VDI implementations cannot return this error in response to any core command for a supported service group.

3 System not initialized

The command was issued before the application issued syInit, or syStop was followed by a command other than syInit.

15 General command error

A command error occurred that is not listed above or about which no information is available.
D Error handling

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 that VDI Management can return to an application.

D.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).

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. However, implementors should try to be complete and should not omit error codes simply for convenience.

VDI Management implementations may supply textual error messages. (See syErrorMsg in Section 6.) Although this is not a compliance requirement, if an implementation does support textual error messages, it must use the summary messages given in this appendix. Although this appendix presents summary messages in mixed case for legibility, VDI Management returns summary messages (and all other return strings) in all capital letters.

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.

1 Filing-system error codes are included primarily for use by future digital audio commands.
Note: VDI Management cannot handle the error of a user forgetting to install VDI Management or the correct interface used by an application. Applications should guard against this by confirming that VDI Management and the proper interface are installed (see Sections 3.2.2 and 3.3.2).

D.2 Error listings

The following subsections list error numbers in numerical order with summary messages and brief explanations. Related errors are grouped for convenience and do not necessarily imply a corresponding programming structure in VDI Management implementations. However, implementations should use the error numbers as they are defined.

D.2.1 Command problems

1 Service group not installed

The VDI 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.

2 Unknown command

The command does not exist.

Compliant VDI implementations cannot return this error in response to any core command for a supported service group.

3 System not initialized

The command was issued before the application issued syInit, or syStop was followed by a command other than syInit.

15 General command error

A command error occurred that is not listed above or about which no information is available.
D.2.2 ASCII interface problems

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 (see Section D.2.4).

17 Command too long
The command was longer than 255 characters and will be ignored in its entirety. (The terminal carriage return counts but redundant delimiters do not.)

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 xxGet-State command to request too much information.

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 illegal 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 has issued an syStop before exiting.

31 General ASCII interface error
An ASCII interface error has occurred that is not listed above or about which no information is available.

D.2.3 Binary interface problems

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.
33 Invalid parameter packet address
On 80x86-based systems, the ES:DI register pair contains an invalid address for a parameter packet.

34 Invalid pointer in parameter packet
The parameter packet contains a null or invalid pointer.

47 General binary interface error
A binary interface error occurred that is not listed above or about which no information is available.

D.2.4 Parameter problems

48 Unknown parameter
The command included a parameter label that is not valid for any command.

Compliant VDI implementations cannot return this error in response to any core parameter for a supported service group.

49 Insufficient parameters
The command required: a specific parameter that was missing; at least one parameter from a specific group of parameters and was issued without the parameter; or at least one parameter that could have been any parameter in its list and was issued with no parameters.

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.

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 result from the combined effects of two or more parameters and 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` plus one. (`logcolors` is the maximum
52 Parameter invalid for this command

The command included a known but invalid parameter. For example, the application issued syInit with a color parameter.

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.

54 Parameter used more than once

The command included the same parameter more than once. This is never allowed.

79 General parameter error

A parameter error occurred that is not listed above or about which no information is available.

D.2.5 Hardware problems

80 Initialization error

The system could not initialize an attached device. The application can find out which device by examining the failed command.

81 Device not initialized

The application tried to use either an uninitialized device or an uninitialized service group.

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 appropriate state. For example, this error would result from a cable being unplugged after a device has been initialized.
83 Communications error
An error occurred during communications. For example, repeated parity errors that cannot be cleared during asynchronous serial communications cause this error.

84 Device reports error
A peripheral device sent a message indicating that an error occurred that it cannot clear.

85 Device canceled request
A peripheral device sent a message indicating that it has unilaterally canceled a requested service.

86 Device not ready
A peripheral device sent a message indicating that it cannot be made operational.

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.

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.

111 General hardware error
A hardware error occurred that is not listed above or about which no information is available.

D.2.6 System resources

Some systems may not be able to return some errors in this group.

112 Insufficient memory
VDI Management could not access enough memory to perform the requested service.
Appendix D. Error messages

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.

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.

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.

116 Needed timer not available
VDI Management requires the use of a timer resource that is not available.

127 General resources error
VDI Management requires additional system resources that are not listed above or about which no information is available.

D.2.7 Filing system problems

128 Invalid filename
The command used a filename that was invalid for the operating system. (A legal filename that cannot be opened should return error 132.)

129 Invalid path
The command used a path name that was invalid for the operating system.

130 Invalid drive
The command specified a drive that is not recognized by the operating system.

131 Invalid file number
The command used a file number that was not recognized by the operating system.

132 Cannot open or create file
The operating system could not open or create a requested file.
133 Cannot close file
The operating system could not close a requested file.

134 File already open
The command tried to open a file that was already open.

135 File already exists
The command tried to create a file that already exists.

136 File does not exist
The command tried to access a file that does not exist.

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.

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.

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.

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.

141 Disk read error
A data error occurred while reading the disk.

142 Disk write error
A data error occurred while writing to the disk.
159 General filing-system error
A filing-system error occurred that is not listed above or about which no information is available.

D.2.8 Miscellaneous problems

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.

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.

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 system vendor.

163 Copy protection error
A copy protected version of VDI Management has declined to run because its protection scheme has been violated. Legitimate users should discuss this problem with the system vendor.

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 system vendor.

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.)

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) in that 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 implementors should not use this error number before carefully considering whether a more informative error code could be used. This is an error of last resort.

D.2.9 System group problems

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.

177 Command cannot be queued
The application tried to queue a command that cannot be queued. This indicates an application problem such as failing to turn syQueue off at the appropriate time.

191 General system error
A problem occurred within the system group that is not listed above or about which no further information is available.

D.2.10 Visual-management problems

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.

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.

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, or issuing vmSetGraphics mode=14, which requires an EGA...
Appendix D. Error messages

graphics adapter, on a CGA system would cause this error. (See Appendix A for more information on VGA native and emulation modes.)

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.

D.2.11 Videodisc problems

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.

209 Disc not spun up

The application issued a command such as vdPlay that requires the videodisc to be spun up when it has not been spun up.

210 Disc not spun down

The application issued a command such as vdSet door=1 that requires the videodisc to be spun down when it has not been spun down.

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.

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.

213 Bad disc section

It was impossible to seek to the required frame because of a physical problem with the videodisc.
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.

215 Invalid frame number
The command specified a frame that is not present on the videodisc.

216 Invalid chapter number
The command specified a chapter that is not present on the videodisc.

217 Invalid time code
The command specified a time code that is not present on the videodisc.

239 General videodisc player error
A videodisc error occurred that is not listed above or about which no information is available.

D.2.12 XY-input device problems

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.

241 Invalid coordinate
A coordinate was specified that is outside the acceptable range. This normally indicates an application problem.

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.

255 General XY-input error
An XY-input error occurred that is not listed above or about which no information is available.
Index

80x86
See: Intel 80x86

Active graphics, A-2
Active video, A-2
Addressing conventions and formats
See: Intel 80x86
See: parameter packets

Apple
II, 2-2, A-4
Macintosh, 2-2

Application interface
See: ASCII interface
See: binary interface
See: interfaces

Application portability
See: compliance requirements
See: portability

Array parameter
vmGetPalette, 7-12–7-13
vmSetPalette, 7-28

ASCII interface, 3-7–3-9, 4-4–4-9
See also: command strings
See also: commands
See also: interfaces
See also: parameters
See also: response strings

Basic characteristics, 3-1
Buffers, 4-8, 6-3
Command name summary, 5-3
Command strings, 3-8
Confirming existence, 3-7–3-8
Device driver name, 3-7
Error return, 3-7

Formal syntax, 4-5–4-6
Format of command examples, 1-4
Format of comments, 1-4
Format of strings, 4-4–4-5
Formats of parameter values, 4-7–4-8
General procedure, 3-7
Multiple commands in one write operation, 4-5
Parameter name summary, 5-5
Reasons for using, 3-1
Response strings, 3-8–3-9
Success return, 3-7

Audio1 parameter
default, 8-12
vdGetState, 8-7
vdSet, 8-31

Audio2 parameter
default, 8-12
vdGetState, 8-7
vdSet, 8-31

Authoring systems, 2-1
Autoexec.bat, 4-10
AX register
See: Intel 80x86

B

B parameter
vmGetPalette, 7-12
vmSetPalette, 7-27

Background processing, 4-4, 7-8
Backus Naur Form, 4-5

BASIC
See: programming examples

Binary interface, 3-3–3-6, 4-9–4-12
See also: commands
See also: Intel 80x86
See also: interfaces
See also: parameter packets
Index

See also: parameters
80x86 register contents upon return, 3-5
80x86 register contents when called, 3-3
basic characteristics, 3-1
buffers, 6-3
calling from device driver, 4-4
command prefix values, 5-1
command token number summary, 5-3
command word values, 5-2
confirming existence, 3-4
default interrupt, 4-9
error return, 3-5
format of comments, 1-4
format of examples, 1-4
format of pointers, 4-11--4-12
formats of parameter values, 4-10--4-12
general procedure, 3-3
parameter packets, 3-4--3-6
parameter token number summary, 5-5
range of interrupts, 4-9
reasons for using, 3-1
requesting information, 3-6
return strings, 3-6
signature, 3-4
success return, 3-5
token numbers, 3-2--3-6, 5-1--5-5
version number, 3-4
binary numeric format, 1-4
BIOS
functions, 4-3
interrupt 10H, 7-2, 7-18, 7-22, 7-24, B-1
interrupt vectors, 4-3
requirements, 2-2
bit fields
See also: buttons parameter
See also: parameters
See also: support parameter
analysis with BASIC, C-5
buttons on XY-input devices, 9-5
supported service groups, 6-9--6-10
bit pad
See: XY-input devices
blanking
horizontal, A-2
vertical, A-3
blue
See: b parameter
border, A-2
buffers
See: ASCII interface
See: binary interface
See: errors
See: VDI Management
buttons
See: buttons parameter
See: tbuttons parameter
See: XY-input devices
buttons parameter
xyGetInput, 9-5
BX register
See: Intel 80x86

C

C language
See: programming examples
case significance, 1-4, 3-2
CAV
See: videodiscs
CBT authoring systems
See: authoring systems
cdisplay parameter
default, 8-12
vdGetState, 8-7
vdSet, 8-31

CGA

See also: graphics adapter
active graphics area, A-3--A-4
background video, A-3
graphics position reference, A-3
horizontal registration to NTSC video, A-5--A-7
horizontal registration to PAL video, A-8--A-10
line width, A-4
pixel size, A-4
reference signal, A-4
standard modes, B-1
vertical registration to NTSC video, A-12--A-15
vertical registration to PAL video, A-16--A-19
chapter
See: chapter parameter
See: vdPlay
See: vdSearch
See: videodiscs
chapter display
See: cdisplay parameter
See: vdSet
chapter parameter
vdGetState, 8-7
vdPlay, 8-19
vdSearch, 8-28
clear parameter
syQueue, 6-15
vmSetTrans, 7-30–7-31
CLV
See: videodiscs
color arrays
See: color parameter
See: parameters
Color Graphics Adapter
See: CGA
color parameter
vmGetPalette, 7-12–7-13
vmGetState, 7-17
vmSetPalette, 7-27–7-28
vmSetTrans, 7-31
colors
See also: color parameter
See also: logcolors parameter
See also: phycolors parameter
See also: transcolors parameter
See also: vmGetPalette
See also: vmSetPalette
See also: vmSetTrans
logical, 4-12, 7-2, 7-4, 7-12
logical-to-physical conversion, 7-2
physical, 7-2, 7-4, 7-12
setting multiple, 4-11–4-12
transparent, 7-17, 7-31
command parameter
syCheckError, 6-3
command queue
See: syQueue
command strings, 3-8
buffering, 4-8
delimiters, 4-5
length limits, 4-5–4-6
syntax, 4-5–4-6
terminator, 4-4
tokens, 4-4
commands
See also: individual command name entries
ASCII name summary, 5-3
binary token number summary, 5-3
core, 2-4–2-5, 5-3
deriving token numbers, 5-2
extended, 2-4–2-5, 5-3
format of names, 1-4
mixing ASCII and binary, 3-9
organization, 2-4
prefix values, 5-1
system (sy) summary, 6-1
unqueueable, 6-16
videodisc (vd) summary, 8-1
visual-management (vm) summary, 7-1
word values, 5-2
XY-input (xy) summary, 9-1
comments
See also: ASCII interface
substitution procedure, 1-5
compatibility
See: compliance requirements
See: device interoperability
See: platform independence
See: portability
compliance requirements
core commands, 2-4
core parameters, 2-4
device numbers, 4-2
dynamic repositioning of graphics, A-1
error messages, D-1
extended commands, 2-5
extended parameters, 2-5
fade and dissolve levels, 7-4
graphics adapters, B-2
graphics modes zero through three, 7-3, B-1
IBM graphics modes, B-1
interface installation, 4-1
interface provision, 3-9
MS–DOS versions, 4-3
service group provision, 2-4
system (sy) commands, 6-1
transparent colors, 7-31
VGA emulation of CGA/EGA, 7-4
VGA emulation of CGA/EGA graphics, A-3–A-4
videodisc (vd) commands, 8-1
videodisc types, 8-2
visual-management (vm) commands, 7-1
XY-input (xy) commands, 9-1
constant angular velocity
See: videodiscs
constant linear velocity
See: videodiscs
cooked mode, 4-9
coordinate space
See: XY-coordinate space
See: xyGet
Index

See: xySet
core commands
   See: commands
   See: compliance requirements
core parameters
   See: compliance requirements
   See: parameters
critical section flag, 4-3
cursor
   See: XY-input devices
cursor keypad
   See: XY-input devices
cursor parameter
   xyGetState, 9-9
   xySet, 9-17

decimal numeric format, 1-4
defdevice parameter
   vdGetState, 8-7
   vdSet, 8-32
   xyGetState, 9-9
   xySet, 9-17
defsource parameter
   vmGetState, 7-17
   vmSetVideo, 7-33
design criteria
   See: interfaces
device driver, 2-3, 3-1
   See also: ASCII interface
   See also: IVDEV
buffer behavior, 4-8
communications, 3-7
duplicate file names, 3-8
IOCTL functions, 4-9
modes, 4-9
MS-DOS version requirements, 4-3
name, 3-7
reentrancy issues, 4-4
device interoperability, 1-1
device numbers, 2-4, 4-2, 9-1–9-2
   See also: defdevice parameter
   See also: device parameter
   See also: source numbers
   See also: tdevices parameter
   See also: vdSet
   See also: xySet
default, 6-12
mapping logical to physical, 4-2
rules, 4-2
stating requirements, 4-2
device parameter
   vdGetState, 8-7
   vdInit, 8-12
   vdPassThru, 8-16
   vdPlay, 8-19
   vdScan, 8-25
   vdSearch, 8-28
   vdSet, 8-32
   vdStep, 8-35
   vdStill, 8-37
   xyGetInput, 9-5
   xyGetState, 9-9
   xyInit, 9-12
   xySet, 9-17
DI register
   See: Intel 80x86
direction parameter, 8-20
   vdPlay, 8-19
   vdScan, 8-25
   vdStep, 8-36
disc
   See: videodiscs
disc player
   See: videodisc players
disdtype parameter
   vdGetState, 8-7
display screen, A-2
display width
   See: width parameter
dissolve level control
   See: dlevel parameter
   See: vmFade
dissolve unit
   See: video overlay subsystem
dissolves
   See: rounding
   See: vmFade
dlevel parameter
   default, 7-21
   vmFade, 7-7
   vmGetState, 7-17
door parameter
   default, 8-12
   vdGetState, 8-8
   vdSet, 8-32
Index

E

EGA
See also: graphics adapter
active graphics area, A-3–A-4
background video, A-3
graphics position reference, A-3
horizontal registration to NTSC video, A-5–A-7
horizontal registration to PAL video, A-8–A-10
line width, A-4
pixel size, A-4
reference signal, A-4
standard modes, B-1
vertical registration to NTSC video, A-12–A-15
vertical registration to PAL video, A-16–A-19
emulation parameter
default, 7-21
vmGetState, 7-17
vmSetGraphics, 7-24
enable parameter
default, 7-21
vmGetState, 7-17
vmSetTrans, 7-31
Enhanced Graphics Adapter
See: EGA
environment variable
See: IVINT
errno parameter
syCheckError, 6-4
syErrorMsg, 6-6
error messages
ASCII interface, D-3
binary interface, D-3–D-4
commands, D-2
filing system, D-7–D-9
hardware, D-5–D-6
miscellaneous, D-9–D-10
parameters, D-4–D-5
system group, D-10
system resources, D-6–D-7
videodisc, D-11–D-12
visual management, D-10–D-11
XY-input device, D-12
glevel parameter
See also: ASCII interface
See also: binary interface
See also: Intel 80x86, AX register
See also: response strings
buffering, 6-3
from queued commands, 6-16
nonimmediate, 6-2
retrieving descriptions, 6-6
retrieving most recent, 6-2
ES register
See: Intel 80x86
execute parameter
syQueue, 6-15
extended commands
See: commands
extended parameters
See: parameters

F

fader
See: video overlay subsystem
fades
See: rounding
See: vmFade
field
See: videodiscs
file
control blocks, 4-9
handles, 4-9
frame
See: frame parameter
See: videodiscs
frame number display
See: idxdisplay parameter
See: vdSet
frame parameter
default, 8-12
vdGetState, 8-8
vdSearch, 8-28
frame search
See: from parameter
See: vdSearch
from parameter
vdPlay, 8-20

G

g parameter
vmGetPalette, 7-12
vmSetPalette, 7-27
genlock control
See: visual management
glevel parameter
default, 7-21
Index

vmFade, 7-8
vmGetState, 7-17
gmode parameter
default, 7-21
vmGetState, 7-18
vmSetGraphics, 7-24
graphics adapter, 7-3-7-4
See also: CGA
See also: EGA
See also: VGA
requirements, 2-2
graphics cursor
See: XY-input device
graphics horizontal line length, A-4
graphics level control
See: glevel parameter
See: vmFade
graphics modes, 7-2-7-3, B-1-B-2
See also: gmode parameter
See also: vmSetGraphics
graphics position relative to video
See: graphics registration
graphics registration, 7-3, A-1-A-19
See also: vmSetGraphics
See also: width parameter
See also: xoffset parameter
See also: yoffset parameter
horizontal, A-4-A-10
reference frame, A-1
terms of reference, A-2
terms of reference, A-2
vertical, A-11-A-19
graphics resolution, A-4
See also: horzpix parameter
See also: vertpix parameter
green
See: g parameter

H

hardware requirements, 2-2
hexadecimal numeric format, 1-4
horizontal blanking
See: blanking
horizontal sync
See: sync signals
horzpix parameter
default, 7-21
vmGetState, 7-18

idxdisplay parameter
default, 8-12
vdGetState, 8-8
vdSet, 8-32
initialization
See: syInit
See: vdInit
See: vmInit
See: xyInit
installable device driver
See: device driver
installation
See: VDI Management
integers
See: parameters
Intel 80x86, 1-2
AX register, 3-3, 3-5
BX register, 3-3, 3-5
DI register, 1-4, 3-3, 3-5
ES register, 1-4, 3-3, 3-5
offset address, 3-3
register for command token, 3-3
register for number of parameters, 3-3
register for packet offset address, 3-3
register for packet segment address, 3-3
register for return codes, 3-3
segment address, 3-3
interfaces
See also: ASCII interface
See also: binary interface
ASCII and binary compared, 2-3, 3-1
design criteria, 2-2-2-3
reasons for two, 2-3
interoperability
See: device interoperability
interrupt
See also: software interrupts
10H, 7-2, 7-18, 7-22, 7-24, B-1
handlers, 4-3-4-4, 4-10
vectors, 4-3
IOCTI, 4-9
IVDEV, 3-7-3-8
duplicate file name, 3-8
IVINT, 3-4
setting, 4-9-4-10
ivver parameter
syGetstate, 6-9
IVVER signature, 3-4
K
keyboard
  See: XY-input device
keyer
  See: video overlay subsystem

L
length parameter
  vmGetPalette, 7-12–7-13
  vmSetPalette, 7-28
light pen
  See: XY-input devices
logcolors parameter
  default, 7-21
  vmGetState, 7-18
logical colors
  See: colors
  See: logcolors parameter
  See: vmGetPalette
  See: vmGetState
  See: vmSetPalette
logical device numbers
  See: device numbers

M
Macintosh
  See: Apple
manufacturer name
  See: mfgname parameter
manufacturer version
  See: mfgver parameter
MCGA
standard modes, B-1
mfgname parameter
  syGetState, 6-9
mfgver parameter
  syGetState, 6-9
microprocessor
  See: Intel 80x86
Microsoft Windows, 2-2
mode trapping
  See: visual management
  See: vimInit
modes
  See: graphics modes
  See: overlay modes
  See: scan-altered modes
  See: video modes

motion parameter
  default, 8-12
  vdGetState, 8-8
mouse
  See: XY-input devices
MS–DOS
  See also: BIOS
  See also: device driver
  See also: software interrupts
  file control blocks and handles, 4-9
generic definition, 1-4
IOCTL functions, 4-9
reentrancy, 4-3–4-4
requiring specific versions, 4-3
tick routines, 4-3
version requirements, 2-2, 4-3
Multicolor Graphics Array
  See: MCGA

N
NTSC
  See also: vmode parameter
  See also: vmSetVideo
  active video, A-6, A-12
  border widths, A-7
  EIA RS–170A, A-3
  frames per second, 3-2, 8-2
  horizontal signal components, A-5
  horizontal signal width, A-5
  vertical height, A-12
  vertical timing components, A-11–A-12

O
offset address
  See: Intel 80x86
operating systems
  See also: MS–DOS
  supported, 1-2, 2-2, 4-3
OS/2, 2-2
overlay board
  See: video overlay subsystem
  overlay modes, 7-2–7-3, B-1

P
PAL
  See also: vmode parameter
See also: `vmSetVideo`
active video, A-9, A-16
border widths, A-10
CCIR 470-1, A-3
frames per second, 3-2, 8-2
horizontal signal components, A-8
horizontal signal width, A-8
vertical height, A-16
vertical timing components, A-15
palette
See also: `colors`
See also: `video overlay subsystem`
See also: `vmGetPalette`
See also: `vmSetPalette`
memory allocation, 7-28
size, 7-4, 7-18
parameter packets, 3-4–3-6
See also: `parameters`
address specification, 3-3
addressing conventions, 1-4, 3-5
color arrays, 4-11–4-12
contents after rounding, 3-6
contents passed, 3-4–3-5
contents returned, 3-6
memory allocation, 3-5
memory requirements, 3-4–3-5
parameters, 3-1–3-3
See also: `individual parameter name entries`
See also: `parameter packets`
ASCII bit fields, 4-7
ASCII name summary, 5-5
ASCII numbers, 4-7
ASCII text, 4-8
binary bit fields, 4-11
binary color arrays, 4-11–4-12, 7-12–7-13
binary integers, 4-10
binary real numbers, 4-10
binary strings, 4-11
binary token number summary, 5-5
core, 2-4–2-5, 5-4–5-5
extended, 2-4–2-5, 5-4–5-5
format of names, 1-4
maximum possible with one command, 3-3
memory requirements for binary, 3-5
order, 3-2–3-3
return values, 3-6
rounding, 3-6
PC DOS
See: `MS–DOS`

physical colors
See: `colors`

See: `physcolors parameter`
See: `vmGetState`
See: `vmSetPalette`
platform independence, 1-1, 2-1
player
See: `videodisc players`

pmsg parameter

`syErrorMsg`, 6-6

point mode
See: `XY-input devices`

pointer format
See: `binary interface`

portability
See also: `compliance requirements`
CGA and EGA to VGA systems, 7-3
extended commands, 2-4–2-5
extended parameters, 2-4–2-5
level addressed, 1-1
transparent colors, 7-31

POSIX, 2-2

processor
See: `Intel 80x86`

programming examples
BASIC, C-1–C-2, C-5
bit field analysis, C-5
C, C-2–C-4
library calls, C-4
software interrupt calls, C-2–C-3

pmsg parameter

`vdPassThru`, 8-16

Q

queue
See: `syQueue`

R

r parameter

`vmGetPalette`, 7-12
`vmSetPalette`, 7-27

raw mode, 4-9

real numbers
See: `parameters`
recommended practices
See also: ASCII interface
See also: binary interface
See also: compliance requirements
See also: VDI Management
basis, 1-1
benefits, 1-2
general architecture, 2-1
goals, 1-2
hardware requirements, 2-2
intended audience, 1-3
IV system level addressed, 1-1
language requirements, 2-2
operating systems addressed, 1-2, 2-2
portability level addressed, 1-1
processor architecture addressed, 1-2
scope, 1-1-1-2
red
See: r parameter
reentrancy
See: MS-DOS
registers
See: Intel 80x86
registration
See: graphics registration
remote control unit
See: remote parameter
remote parameter
default, 8-12
vdGetState, 8-8
vdSet, 8-32
response strings, 3-8-3-9, 4-5
buffering, 4-8
case, 3-5, 3-9
delimiters, 3-8, 4-5
"ERROR n...", 3-8
"OK", 3-8
syntax, 4-5-4-6
terminator, 3-8, 4-5
RGB
See: b parameter
See: g parameter
See: r parameter
rounding
color components, 7-13, 7-27-7-28
effects on parameter packets, 3-6
fade and dissolve levels, 7-4-7-5
times, 7-4-7-5
videodisc player speeds, 8-3-8-4

S
scan
See: vdScan
See: videodisc players
scan-altered modes, A-3
screen disturbance, 6-14, 7-24-7-25, 7-34, 8-11,
A-13
search
See: vdSearch
See: videodisc players
segment address
See: Intel 80x86
service groups, 2-4
See also: commands
determining if present, 4-7, 4-11, 6-9-6-10
token values of prefixes, 5-1
software architecture
See: recommended practices
software interrupts
See also: interrupt
See also: IVINT
21H, 4-4, 4-9
default, 4-9
numbers available, 3-1, 4-9
programming example, C-2-C-3
user, 4-9
source numbers
See also: defsource parameter
See also: source parameter
See also: tsources parameter
See also: vmSetVideo
default, 7-17
mapping to devices, 7-17
speed parameter
vdGetState, 8-8
vdPlay, 8-20
spin parameter
default, 8-12
vdGetState, 8-8
vdSet, 8-32
standard
See: NTSC
See: PAL
state parameter
default, 7-21
syQueue, 6-15
vmSetTrans, 7-31
still frame
See: vdSearch
Index

See: vdStep
See: vdStill
stream mode
See: XY-input devices
string
See: ASCII interface
See: command strings
See: parameters
See: response strings
support parameter
syGetState, 6-9-6-10
syCheckError, 6-2-6-5
syErrorMsg, 6-6-6-7
syGetState, 6-8-6-11
syInit, 6-12-6-18
sync signals
horizontal, A-2-A-3
vertical, A-2-A-3
system initialization
See: syInit
syStop, 6-19-6-20

T
buttons parameter
default, 9-13
xyGetState, 9-9
tdevices parameter
default, 8-12, 9-13
vdGetState, 8-8
xyGetState, 9-9
tick chain, 4-3
time display
See: idxDisplay parameter
See: vdSet
time parameter
vmFade, 7-8
to parameter
vdPlay, 8-21
token numbers
See: binary interface
See: commands
See: parameters
touch screen
See: XY-input devices
transcolors parameter
default, 7-21
vmGetState, 7-18
transparent colors
See: colors
tsources parameter
vmGetState, 7-18

U
UNIX, 2-2
user interrupt
See: software interrupts

V
vdGetState, 8-5-8-10
VDI Management, 2-1
background processing, 4-4
blanking control, 8-2
communications, 3-1
correction of XY-input information, 9-1
error buffers, 6-3
error handling, D-1
genlock control, 7-3
graphics positioning requirements, A-1
installation, 2-4, 4-1-4-2, 9-2
interrupt handle vector, 4-10
memory allocation for return strings, 4-11
mode trapping, 7-3
MS-DOS calls, 4-3
nonimmediate error detection, 6-2-6-3
parser design, 3-3
rounding, 7-4-7-5, 7-27, 8-3-8-4
treatment of XY-input information, 9-3
user device numbering, 4-2
version number, 3-4
vdInit, 8-11-8-14
vdPassThru, 8-15-8-17
vdPlay, 8-18-8-23
vdScan, 8-24-8-26
vdSearch, 8-27-8-29
vdSet, 8-30-8-34
vdStep, 8-35-8-36
vdStill, 8-37-8-38
version number
See: iver parameter
See: IVVER signature
See: mfgver parameter
See: VDI Management
vertical blanking
See: blanking
vertical sync
See: sync signals
vertpix parameter
  default, 7-21
  vmGetState, 7-18
VGA
  See also: emulation parameter
  See also: graphics adapter
  active graphics area, A-3–A-4
  background video, A-3
  blanking intervals, A-3
  borders, A-13
  emulation of CGA/EGA, 7-3–7-4
  horizontal registration in emulation mode, A-7–A-8, A-10–A-11
  horizontal sync, A-3
  reference signal, A-4
  standard modes, B-1
  vertical registration to NTSC video, A-12–A-15
  vertical registration to PAL video, A-16–A-19
  vertical sync, A-3
video channel
  See: vdSet
  See: video parameter
Video Graphics Array
  See: VGA
video level control
  See: vlevel parameter
  See: vmFade
video modes, 7-3
  See also: vmode parameter
  See also: vmSetVideo
video overlay subsystem
  See also: individual vm command name entries
dissolve unit, 7-2
functionality, 7-1–7-2
keyer, 7-2
palette, 7-2
sources, 7-2
video parameter
  default, 8-12
  vdGetState, 8-9
  vdSet, 8-33
video position relative to graphics
  See: graphics registration
video standard
  See: NTSC
  See: PAL
videodisc player door control
  See: door parameter
videodisc player initialization
  See: vdInit
videodisc players
  See also: individual vd command name entries
  See also: videodiscs
  instant jump, 8-2
  numbering, 4-2
  Pioneer 4200, 8-3
  play speeds, 8-2
  scan speeds, 8-2
  search, 8-2
  Sony 2000, 8-3
  speeds, 8-3–8-4
videodiscs
  See also: disctype parameter
  See also: videodisc players
  CAV, 8-1–8-3
  chapters, 8-2
  CLV, 8-1–8-4
  fields, 8-2
  frames, 8-2
  reference frame, A-1
Virtual Device Interface
  See: VDI Management
visual management
  See also: graphics registration
  See also: individual vm command name entries
  See also: video overlay subsystem
  genlock control, 7-3
  graphics registration, 7-3
  logical and physical colors, 7-4
  mode trapping, 7-3
  overlay modes, 7-2–7-3
  rounding fade and dissolve levels, 7-4–7-5
visual management initialization
  See: vmInit
vlevel parameter
  default, 7-21
  vmFade, 7-8
  vmGetState, 7-17
  vmFade, 7-6–7-10
  vmGetPalette, 7-11–7-14
  vmGetState, 7-15–7-20
  vmInit, 7-21–7-22
vmode parameter
  default, 7-21
  vmGetState, 7-18
  vmSetVideo, 7-33–7-34
VMS, 2-2
vmSetGraphics, 7-23–7-25
vmSetPalette, 7-26–7-29
vmSetTrans, 7-30–7-32
vmSetVideo, 7-33–7-34

See also: xySet

correcting for multiple physical devices, 9-1

XY-input device initialization
See: xyInit

XY-input devices
See also: individual xy command name entries
buttons, 9-3
calibration, 9-3
cursor keypad, 9-2
cursor management, 9-2
graphics plane management, 9-2
keyboard, 9-2
mapping, 9-1–9-2
multiple physical to single logical, 9-1–9-2
numbering, 4-2
point mode, 9-3
requirements, 2-2
stream mode, 9-3

xyGetInput, 9-4–9-6
xyGetState, 9-7–9-11
xyInit, 9-12–9-14
xySet, 9-15–9-18

W

wait parameter
effects on vdPlay, 8-21
vdPlay, 8-21–8-22
vdScan, 8-25
vdSearch, 8-28
vdSet, 8-33
vmFade, 7-8

width parameter
default, 7-21
vmGetState, 7-18
vmSetGraphics, 7-24

See: Microsoft Windows

X

xmax parameter
default, 9-13
xyGetState, 9-9
xySet, 9-17

xmaxclip parameter
default, 9-13
xyGetState, 9-10
xySet, 9-17

xmin parameter
default, 9-13
xyGetState, 9-9
xySet, 9-17

xminclip parameter
default, 9-13
xyGetState, 9-10
xySet, 9-17

xoffset parameter
default, 7-21
vmGetState, 7-19
vmSetGraphics, 7-24

xpos parameter
default, 9-13
xyGetInput, 9-5
xySet, 9-17

XY-coordinate space, 9-2–9-3
See also: xyGetState

Y

ymax parameter
default, 9-13
xyGetState, 9-9
xySet, 9-17

ymaxclip parameter
default, 9-13
xyGetState, 9-10
xySet, 9-17

ymin parameter
default, 9-13
xyGetState, 9-9
xySet, 9-17

yminclip parameter
default, 9-13
xyGetState, 9-10
xySet, 9-17

yoffset parameter
default, 7-21
vmGetState, 7-19
vmSetGraphics, 7-24

ypos parameter
default, 9-13
xyGetInput, 9-5
xySet, 9-17