DPVL-SI Standard

860 Hillview Court, Suite 150
Milpitas, California 95035

Phone: (408) 957-9270
Fax: (408) 957-9277

VESA DPVL SOFTWARE INTERFACE STANDARD

Version 1
February 28. 2006

Purpose

This proposal defines a device independent software interface handler which resides between the graphics subsystem of an operating system and a device specific graphics card driver. This handler allows third party software developers to create a DPVL system that can be used by any graphics board that supports the extensions defined in this proposed standard.

Summary

To help enable the proliferation of DPVL enabled monitors, it is advantageous for a software interface standard to allow the majority of the DPVL software to be developed in a device independent manner. Two software interfaces are defined by the proposed standard. The first interface is between the device independent DPVL-Software Interface (DPVL-SI) handler and the device specific graphics card driver. The second interface is between the device independent DPVL-SI Handler and an application program utilizing features of the DPVL standard. Windows-specific and Linux-specific implementations are included in this standard.

Table of Contents

LIST OF FIGURES

LIST OF TABLES

Intellectual Property

While every precaution has been taken in the preparation of this standard, the Video Electronics Standards Association and its contributors assume no responsibility for errors or omissions, and make no warranties, expressed or implied, of functionality or suitability for any purpose.

Trademarks

Windows is a registered trademark of Microsoft Corporation.

Patents

VESA proposals and standards are adopted by the Video Electronics Standards Association without regard as to whether their adoption may involve any patents or articles, materials, or processes. Such adoption does not assume any liability to any patent owner, nor does it assume any obligation whatsoever to parties adopting the proposals or standards documents.

Other Documents Referenced

Note: Versions identified here are current, but users of this standard are advised to ensure they have the latest versions of referenced standards and documents.

Source	Name	Version / Date
VESA	Digital Packet Video Link (DPVL) Standard	Ver. 1, April 18, 2004
VESA	Enhanced Display Data Channel (E-DDC) Standard	Ver. 1.1, March 24, 2004

Table 1-1: Reference Documents

Support for this Standard

Clarifications and application notes to support this standard may be written. To obtain the latest standard and any support documentation, contact VESA.

If you have a product which incorporates any variant of the DPVL system, you should ask the company that manufactured your product for assistance. If you are a manufacturer, VESA can assist with any clarification you may require. All comments or reported errors should be submitted in writing to VESA using one of the following methods.

Fax:	408-957 9277, direct this note to Technical Support at VESA
e-mail:	support@vesa.org
mail:	Technical Support Video Electronics Standards Association 860 Hillview Court, Suite 150 Milpitas, CA 95035

Conformance Glossary – Definition of Terms

The requirements and specifications for the VESA DPVL-SI standard adhere to the following definitions and conventions:

a) Features or functions that are required to be implemented are identified by the word **shall** in bold type. Failure to adhere to a feature or function identified by **shall** may cause application restrictions, result in improper functioning, or hinder operations. For a device to be DPVL-Software Interface compliant it must implement all features or functions identified by **shall**.

b) Features or functions that are desirable, but that are not required to be implemented are identified by the word **should** in bold type. Failure to adhere to a feature or function identified by **should** may inhibit the implementation of some features or functions in specific applications and environments.

c) Features or functions that are optional and are not needed to be implemented are identified by the word **may** in bold type. Such features or functions represent goals to be achieved, and may enhance convenience and utility of the VESA DPVL-Software Interface.

d) A number prefaced with 0x is a hexadecimal number, any number without a preface of 0x is a decimal number.

Revision History

February 28, 2006 – Initial release of the standard

Acknowledgements

This document would not have been possible without the efforts of the VESA Display Systems Standards Committee's DPVL Task Group. In particular, the following individuals and their companies contributed significant time and knowledge to this edition of the standard.

Syed Athar Hussain	ATI Technologies	Marcus Andrews	Microsoft
**Ricardo Baratto	Columbia University	Wayne Young	NVidia
**Jason Nieh	Columbia University	Michael Anderson	Portrait Displays Inc.
Jim Webb	Display Labs	Mark Stockfisch	Quantum Data
Takashi Matsui	EIZO NANAO	Ian Miller	Samsung Electronics
Steve Millman	IBM	Joe Lamm	Tech Source
Kai Schleupen	IBM	Phil Merritt	Tech Source
Eric Wogsberg	Jupiter Systems	Alain d'Hautecourt	ViewSonic
Ameet Chitre	Microsoft		

** non-member participant

1. Overview

This document describes the Digital Packet Video Link Software Interface (DPVL-SI) Standard, which defines software interfaces between a device independent software handler and a device specific graphics driver. The standard also defines a software interface between the device independent software handler and an application layer program.

1.1 Background

Since the DPVL standard was released, companies wanting to develop DPVL-compliant products for general use have had a difficult time writing the software necessary to create a complete system. Primarily this difficulty is due to the necessity of possessing specific knowledge of and the ability to modify the graphics drivers provided by the graphics card companies. The graphics card companies are hesitant to expend the effort needed to create the entire DPVL software system until they can see a clear widespread use, and have multiple hardware implementations to choose from. The monitor companies that have the ability to create DPVL compliant hardware are reluctant to embark on development projects to create the hardware when there is no standard software available. Monitor companies typically do not have the software expertise to create a full DPVL software system. Therefore, to alleviate this problem, a software interface standard is needed to allow the majority of the DPVL software to be developed in a device independent manner. The device specific portion should require minimal modifications to already existing graphics card drivers.

1.2 Objectives

The objective of the DPVL-SI standard is to define two software interfaces. The first interface is between the device independent DPVL-SI Handler and the device specific graphics card driver. The second interface is between the device independent DPVL-SI Handler and an application program wishing to utilize features found in the DPVL standard.

1.3 Document Layout

Section 1 provides an overview and justification of the proposed standard. Section 2 describes the architecture of the software interfaces and how they fit into the overall system. Section 3 describes the interface between the device independent DPVL-SI Handler and the device specific graphics card driver (hereafter called the “Driver Interface”) in an operating system agnostic manner. Section 4 defines the interface between the device independent DPVL-SI Handler and an application program (hereafter called the “Application Interface”) in an operating system agnostic manner. Annex 1 in section 5 defines interfaces in a Microsoft Windows implementation. Annex 2 in section 6 defines the interfaces in a Linux environment.

2 Architecture

The DPVL-SI Handler is a device independent software layer that translates traditional graphics commands from the operating system into commands appropriate for a DPVL system. For example:

> The operating system wants to draw a line on the screen. It issues a draw line command to the DPVL-SI Handler. The DPVL-SI Handler decodes the command and determines that the only action required is to keep track of the portion of the virtual screen that is affected by the command. It then passes the command to the Display Driver.

There are four possible outcomes for what happens when the DPVL-SI Handler receives a command from the operating system:

A	Pass the command unchanged to the display driver.
B	Pass the command unchanged to the display driver, but keep track of what the command was.
C	Swallow the command.
D	Swallow the command, but issue a different set of commands to the display driver.

In the following sections, architecture for the DPVL-SI Handler in both Windows and Linux is described.

2.1 Windows

For Window's architecture, the DPVL-SI Handler may be either a filter driver which resides in kernel mode on top of the display driver or a user mode .dll which resides in user mode and dispatches commands to the display driver. In addition, a system compromised of both a kernel mode filter driver and a user mode component can also be envisioned, where for example the user mode portion is used for USB communication.

A list of the major advantages/disadvantages of each implementation appears below:

A. .dll only

The advantages of using this setup are:

1. Elimination of the need for a filter driver (and the associated problems with verification and signing).
2. Simplicity in the support of Extended Packets.
3. Easier synchronization between drawing and updates (reduces tearing).

4. Easier debugging since it is user mode (fewer reboots).
5. Off-screen buffer can easily be used to transmit packets.
6.

The disadvantages of using this setup are:

1. Portability - Applications would have to be modified to control the DPVL displays.
2. Multiple DPVL applications could not be run on same monitor.
3. May only work in limited application scenarios.

B. filter driver (& .dll)

The advantages of using this setup are:

1. Portability – Drawing commands are captured through the filter driver and applications are not required to be DPVL-aware.
2. Not necessary to rewrite applications so that they are DPVL aware.
3. The filter driver can act as arbiter between multiple DPVL applications.

The disadvantages of using this setup are:

1. May require both a filter driver and a .dll to support Extended Packets.
2. Difficult debugging since it is kernel mode (more system crashes).
3. May require extra space to be reserved in the frame buffer for header.

2.1.1 Filter driver implementation technique (Windows)

In this scenario, the DPVL-SI Handler is inserted as a filter driver between the operating system and the display driver. The architecture of Windows lends itself to the implementation of such a scenario. In this way, the DPVL-SI Handler can listen to all commands normally going to the display driver and decide how to handle each of them. Figure 2-1 shows a block diagram of a typical Windows system implementing the filter driver technique.

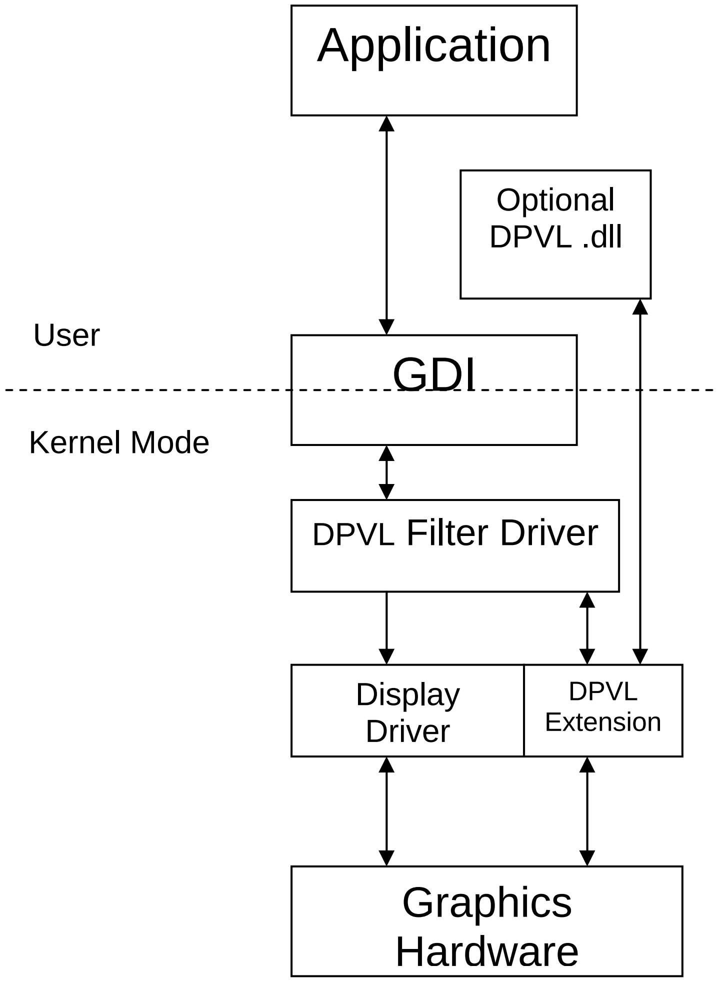

Figure 2-1 - Software Interface Block Diagram for a filter driver implementation under Windows

2.1.2 .dll implementation technique (Windows)

In this scenario, the DPVL-SI Handler resides in user mode and issues DPVL commands to the display driver. Figure 2-2 shows a block diagram of a typical Windows-based system.

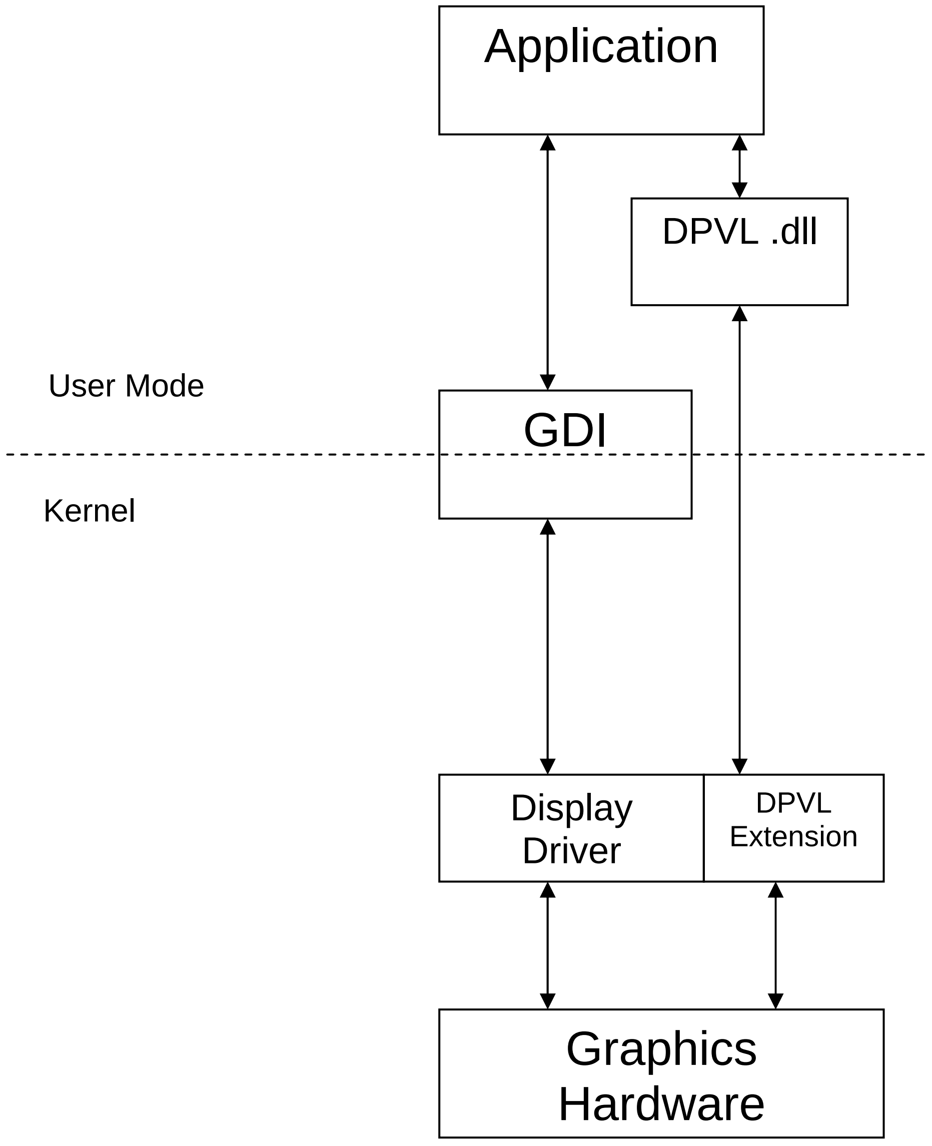

Figure 2-2: Software Interface Block Diagram for a .dll implementation under Windows

2.2 Linux

The graphics system under Linux is somewhat different than it is under Windows. Much of the graphics system resides in user mode.

2.2.1 X Extension implementation technique (Linux)

In this scenario, the DPVL-SI Handler resides in user mode inside an X server, and issues DPVL commands to the display driver. The application issues commands to a DPVL-library which uses a DPVL X extension to communicate with the DPVL-SI Handler. Figure 2-3 shows a block diagram of a typical Linux-based system. Though shown in the figure, the Kernel Driver is typically only used for three dimensional graphics operations and its implementation is optional.

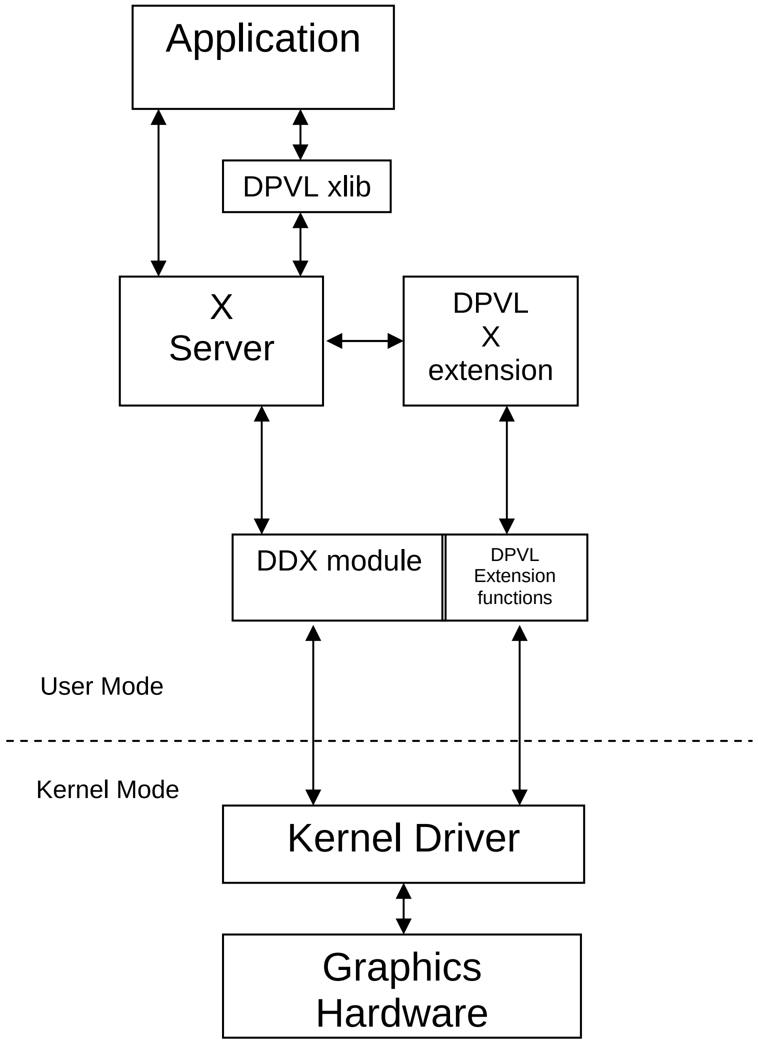

Figure 2-3: Software Interface Block Diagram for a Linux based X extension implementation

3 Driver Interface

This section defines the interface between the device-independent DPVL-SI Handler and the device-specific DPVL extension of the video card driver. Separate annexes (sections 5 and 6) provide platform-specific information for Windows and Linux operating systems.

3.1 Retrieving the Interface

Communication between the DPVL-SI Handler and the video card driver will be handled through a set of defined functions. How access is gained to these functions is operating system dependent and is outlined in the annexes. The actions taken by these functions and the information passed to and from them, however, are not dependent on the operating system, and are described below.

3.2 Interface Functions

3.2.1 DPVLGetInterface

Before the DPVL-SI Handler can begin communicating with the video card driver, it must determine some basic information such as which versions of the DPVL specification are supported by the driver, how this support is implemented, and how many DPVL devices are attached to that adapter. In some cases, this number may be different from the number of devices reported to the operating system. For example, a three-by-three array of nine monitors may be reported to the operating system as one large monitor.

In each of the following functions, the first parameter passed in is “index.” This index corresponds to one of the devices attached to the video card, and may be any value from 0 to the number of devices attached to the video card (as reported by DPVLGetInterface) minus one.

For each operating system, DPVLGetInterface is uniquely defined and returns the information necessary to access the remainder of the functions defined in this specification. If the pointer to any of the following functions is NULL, then that function is considered to be unsupported by the video card driver. Unless that function is marked as optional below, returning a NULL pointer shall be considered an error, though the DPVL-SI Handler may choose to continue if it can operate without that function.

3.2.2 DPVLGetDeviceAttr

This function **should** be called once for each of the DPVL devices attached to the video card adapter (the number returned by DPVLGetInterface). In Table 3-1 the information that is returned for each device is shown.

Parameter	Description
deviceIndex	Indicates the device for which information is being requested. This number must be between zero and one less than the number of devices returned by DPVLGetInterface.
Flags	Any combination of the following: DPVL_USE_HWC – Indicates that the DPVL header can be supplied via the hardware cursor for this device. DPVL_AUTO_HEADER – Indicates that the video card driver is capable of loading the DPVL header into the video stream automatically, based on the DPVLSetCrt call. It may or may not use the hardware cursor. DPVL_PANNING – Supports hardware panning of display surface for this device. DPVL_NO_KERNEL_MODULE – Indicates that this device does not support kernel mode DPVL-SI Handler implementations. DPVL_NO_USER_MODULE – Indicates that this device does not support user mode DPVL-SI Handler implementations.
fbMaxWidth	Maximum width in pixels that the frame buffer can support.
fbMaxHeight	Maximum height in pixels that the frame buffer can support.
fbMaxSize	Maximum number of bytes that may be allocated to the visible frame buffer.
fbOffsetGranularity	Granularity in pixels of changes to the frame buffer start address.
fbWidthGranularity	Granularity in pixels of valid values for width of pixmaps in frame buffer, including desktop.
fbHeightGranularity	Granularity in pixels of valid values for height of pixmaps in frame buffer, including desktop.
hwcWidth	Maximum width in pixels of the hardware cursor.
hwcHeight	Maximum height in pixels of the hardware cursor.

Table 3-1: Parameters for the DPVLGetDeviceAttr function

If the DPVL-SI Handler uses the hardware cursor to load a header, then later switches to using the video card driver's auto-header feature, the DPVL-SI Handler is responsible for first turning off the hardware cursor, which may interfere with the new packet. It is recommended that the DPVL-SI Handler turn off the hardware cursor during initialization to prevent any possible interference. As described in the DPVL standard, device specific limitations are available to the DPVL-SI Handler through E-DDC (VESA Enhanced Display Data Channel Standard, version 1.1, March 24, 2004).

3.2.3 DPVLGetCrtcMap

This function returns an integer for each DPVL device attached to the graphics card (the same number of devices reported by DPVLGetInterface). That integer is an index number indicating which CRT controller is supplying information to that device. Space for this single dimensional array should be allocated by the DPVL-SI Handler prior to calling the function.

For example, if three DPVL devices are attached to two CRT controllers, the array returned may look like: {0, 0, 1}. This indicates that the first two devices are attached to one CRT controller, and the third (device index 2) is attached to another.

3.2.4 DPVLGetCrtcAttr

This function may be called once for each unique integer returned by DPVLGetCrtcMap. In Table 3-2 the parameters returned by calling the DPVLGetCrtcAttr function for each device are listed.

Parameter	Description
iCrtcIndex	Indicates the CRT controller for which information is being requested. This number must be equal to one of the integers returned by DPVLGetCrtcMap.
crtcMaxWidth	Maximum width in pixels of video output that can be programmed into the graphics card's crt controller for this device.
crtcMaxHeight	Maximum height in pixels of video output that can be programmed into the graphics card's crt controller for this device.
crtcMaxTotal	Maximum horizontal total in pixels, including active video and blanking, that can be programmed into the graphics card's crt controller for this device.
crtcMaxSize	Maximum number of bytes of the frame buffer that can be assigned to the active desktop.
crtcWidthGranularity	Granularity in pixels of valid values for width of video output that can be programmed into the graphics card's video controller.
crtcHeightGranularity	Granularity in pixels of valid values for height of video output that can be programmed into the graphics card's video controller.
crtcMaxPanningX	Maximum distance in pixels that may be panned in the X direction by the graphics card's crt controller.
crtcMaxPanningY	Maximum distance in pixels that may be panned in the Y direction by the graphics card's crt controller.

Table 3-2: Parameters for the DPVLGetCrtcAttr function

3.2.5 DPVLEnable

As the name implies, a call to this function shall be made when transitioning between DPVL mode and raster mode. At boot time, all video cards are assumed to be running in raster mode. If the flags parameter passed in is TRUE, the video card driver **shall** turn off raster mode and prepare to send DPVL packets. When the flags parameter passed in is FALSE, the video card driver **shall** stop sending DPVL packets and return to the same raster mode it was running prior to switching to DPVL mode. In Table 3-3 the parameters for the DPVLEnable function are listed.

Parameter	Description
index	Index of the device entering or leaving DPVL mode
flags	Indicates whether the device is entering or leaving DPVL mode

Table 3-3: Parameters for the DPVLEnable function

3.2.6 DPVLSetCrt

This is the workhorse function of the DPVL interface. Whenever the DPVL-SI Handler decides that a particular rectangular area of the screen must be updated, it shall send a description of that rectangle and its position relative to the top left corner of that display device. In Table 3-4 the parameters for the DPVLSetCrt function are listed.

Parameter	Description
index	Index of the device displaying the given rectangle.
flags	Any one of the following: DPVL_WAIT_FOR_UPDATE – Indicates that this function should not return until this and all previous DPVL packets have been sent. DPVL_RETURN_IF_BUSY – Indicates that this function should return if it is unable to send the packet immediately, such as when another packet is still pending. If busy, the request should be discarded. DPVL_ASYNCHRONOUS – Indicates that the function should queue up the given rectangle and return immediately, sending the DPVL packet as appropriate. DPVL_SIGNAL – Indicates that the video card driver should signal the DPVL-SI Handler after this packet has been sent. This overrides the DPVL_SIGNAL_NONE setting. DPVL_AUTO_HEADER – Indicates that the video card driver shall load the header into the video stream automatically. This is only valid if DPVLGetDeviceAttr returned DPVL_AUTO_HEADER for this device. DPVL_PAN_ONLY – Indicates that only the panRight and panDown values have changed since the last DPVL packet was sent.
priority	Indicates how the request should be queued. DPVL_PRIORITY_HIGH – Indicates that the packet should be updated as quickly as possible, taking precedence over refresh operations. DPVL_PRIORITY_NORMAL – Reserved. DPVL_PRIORITY_LOW – Indicates that this is a refresh operation, and may be delayed if necessary.
rclWidth	Width in pixels of the rectangular area of video data that must be updated.
rclHeight	Height in pixels of the rectangular area of video data that must be updated.
rclTop	Top row of the rectangular area of video data that must be updated, relative to the top, left corner of the device indicated by ‘index’.
rclLeft	Leftmost column of the rectangular area of video data that must be updated, relative to the top, left corner of the device indicated by ‘index’.
panRight	Number of pixels that the video card driver should pan to the right before outputting video data.
panDown	Number of pixels that the video card driver should pan down before outputting video data.
ulTimeout	Maximum number of milliseconds the video card driver should attempt to update the rectangle before discarding the request and returning DPVL_SETCRT_TIMED_OUT.

Table 3-4: Parameters for the DPVLSetCRT function

Upon receipt of this call, the video card driver shall send a DPVL packet to the appropriate device, containing the data necessary to update the given rectangle.

When a video card driver is asked to pan a device that does not support hardware panning or when the panRight or panDown values exceed the capabilities of that device, the video card driver should add these panning values to the rclLeft and rclTop values respectively.

If any other parameter exceeds the capabilities of the hardware, the video card driver shall return a failure. Attempts to break the call up into multiple DPVL packets may change the behavior of the DPVL-SI Handler and produce unwanted artifacts or tearing.

Valid return values are:
`DPVL_SETCRT_SUCCESS` – This indicates that the packet request has completed or been queued as requested.
`DPVL_SETCRT_INVALID_INDEX` – No device exists corresponding to the given index value.
`DPVL_SETCRT_UNSUPPORTED_FLAG` – One or more flags passed to the driver is either unsupported or undefined.
`DPVL_SETCRT_INVALID_RECTANGLE` – One or more of the parameters defining the requested rectangle exceed the hardware capabilities of the device.
`DPVL_SETCRT_OUT_OF_BOUNDS` – All or part of the requested rectangle extends beyond the boundaries of the virtual desktop.
`DPVL_SETCRT_BUSY` – DPVL_RETURN_IF_BUSY was set and the video card driver was still processing a previous packet request.
`DPVL_SETCRT_AUTO_HEADER_FAILURE` – Indicates that the video card driver was unable to automatically load the parameters of the requested packet into the header.
`DPVL_SETCRT_TIMED_OUT` – If the video card driver is unable to complete the packet request within the number of milliseconds given in ulTimeout, it shall discard the request and return this value.

3.2.7 DPVLValidateVirtualResolution

Although the DPVL-SI Handler can use the information from DPVLGetDeviceAttr to make assumptions about what resolutions the video card driver can support, it **should** always call DPVLValidateVirtualResolution to verify those assumptions before reporting a resolution to the operating system. In Table 3-5 the parameters for the DPVLValidateResolution function are listed.

Parameter	Description
index	Index of the device being asked to validate the given resolution.
width	Width in pixels of the virtual desktop.
height	Height in pixels of the virtual desktop.

Table 3-5: Parameters for the DPVLValidateResolution function

3.2.8 DPVLSetVirtualResolution

If a DPVL-SI Handler reports additional virtual resolutions to the operating system, it may be necessary to pass that information down to the video card driver so that it can rearrange its video memory or make whatever changes are necessary. In Table 3-6 the parameters for the DPVLSetVirtualResolution function are listed.

Parameter	Description
index	Index of the device whose virtual resolution should be changed.
width	Width in pixels of the virtual desktop.
height	Height in pixels of the virtual desktop.

Table 3-6: Parameters for the DPVLSetVirtualResolution function

An example would be a video card that can only support 640x480 in raster mode. If it is connected to a 1280x1024 DPVL monitor, the DPVL-SI Handler might report 1280x1024 as a valid resolution to the operating system. When the user selects that mode, the DPVL-SI Handler would ask the video card driver to enter DPVL mode, and set its virtual desktop to 1280x1024. The DPVL-SI Handler must not ask the video card driver to send a packet larger than it can handle (640x480 in this example).

The video card driver may refuse to set the requested virtual resolution.

3.2.9 DPVLSetHWCursorShape (optional)

A DPVL-SI Handler may be implemented in user space (e.g. a .dll file), in kernel space (e.g. a filter driver), or a combination of the two. While a filter driver may use standard methods for enabling and reshaping the hardware cursor, a user space implementation may have trouble communicating directly with the video card driver. This function provides a way for user space components to enable the hardware cursor or change its shape.

As described in section 3.1 of the DPVL 1.0 specification, the DPVL header can be implemented by embedding the header in the video stream by using the hardware cursor feature. In this implementation, when in DPVL mode, the video card driver is not expected to be using the hardware cursor for user input. If it is, the driver may fail this call. In Table 3-7 the parameters for the DPVLSetHWCursorShape function are listed.

Parameter	Description
index	Index of the device whose virtual resolution should be changed.
XXXX	OS dependent

Table 3-7: Parameters for the DPVLSetHWCursorShape function

3.2.10 DPVLSetHWCursorPosition (optional)

As with DPVLSetHWCursorShape, this function is provided to allow user space components to manipulate the hardware cursor provided by the video card driver, in this case, its position in the output stream. A DPVL-SI Handler that exists in the same layer as the video card driver (e.g. filter driver) may choose to use standard operating system calls to accomplish the same task. In Table 3-8 the parameters for the DPVLSetHWCursorPosition function are listed.

Parameter	Description
index	Index of the device whose virtual resolution should be changed.
XXXX	OS dependent

Table 3-8: Parameters for the DPVLSetHWCursorPosition function

3.2.11 DPVLSetSignaling

It is often desirable for the DPVL-SI Handler to be notified when a given packet has been successfully sent by the video card driver to the attached device. While the exact mechanism is operating system specific, accommodations should be made to support various standard signaling options. Note that overlapping signaling requests, such as setting both DPVL_SIGNAL_PER_PACKET and the signal option in DPVLSetCrt, should never result in more than one signal being sent per packet. In Table 3-9 the parameters for the DPVLSetSignaling function are listed.

Parameter	Description
index	Indicates the device for which signaling is to be set. This number must be less than the number of devices returned by DPVLGetInterface. A value of -1, however, shall indicate that the setting should be extended across all attached devices.
flags	Any one of the following: DPVL_SIGNAL_NONE – Indicates that no signal should be sent by the video card driver to the DPVL-SI Handler upon completion of a given packet, unless the signal option was set in the DPVLSetCrt call for that packet. DPVL_SIGNAL_PER_PACKET – Indicates that the video card driver should signal the DPVL-SI Handler each time a packet has been sent. DPVL_SIGNAL_QUEUE_EMPTY – Indicates that the video card driver should signal the DPVL-SI Handler after all requested packets have been sent.

Table 3-9: Parameters for the DPVLSetSignaling function

4 Application Interface

4.1 Overview

There are many features defined in the DPVL specification which, if supported by the DPVL-SI Handler, can be taken advantage of by applications. However, unless the software interface is defined and the application can query if a specific capability is supported, there is no guaranteed mechanism for taking advantage of these many features.

In Table 4-1, the features, which an application may take advantage of, as defined in the DPVL version 1.0 specification, are listed along with a short description of each feature. A much fuller description of each feature can be found in the DPVL standard. In addition, the DPVL level number (see section 5 of the DPVL version 1.0 standard) to which the feature applies is also listed along with the section number within the DPVL document that describes the feature. Note that support of some of these features is required for a DPVL-SI Handler to conform to a specific level. See the DPVL standard for more details of this requirement.

	Feature	Short Description	Section	DPVL Level
1	Scene Sync'ing	Reduces tearing artifacts by defining the last packet of a scene and to which image buffer the pixel data is written.	3.9 and 4.3	0,1,2
2	Video Packet	Carries a bitmap image to update a rectangular region of the screen	4.3.1	2
3	Genlock	Useful in multi-monitor, stereo and motion picture applications as a frame synchronization mechanism	4.3.2	2
4	Scaled Video Stream Setup	Used to designate the size of the source bitmap image and the size and position of the destination rectangle	4.3.3.1	
5	Scaled Video Stream	Use the monitor to scale original size image data to monitor unique scale and position values	4.3.3.2	2
6	Gamma Table	Load a downstream Look-Up-Table in the monitor with a table of color values	4.3.4	2
7	Frame Buffer Control	Allows independent control of the displayed frame buffer in both monaural and stereo modes	4.3.5	2
8	BitBlt	Provides a means to scroll or move regions of the screen in any direction	4.3.6.1	2
9	Area Fill	Initialize a region of the frame buffer to a fixed color value	4.3.6.2	2
10	Pattern Fill	Initialize a region of the frame buffer to a pattern	4.3.6.3	2
11	Transparancy Map Packet	Used in conjunction with the Scaled Video Stream Setup packet to indicate how the stream data should be blended with the existing frame buffer contents	4.3.7	2

Table 4-1: List of the features which the application layer supports

4.2 Mechanism for Accessing the Features

As described in section 2 of this document, the DPVL-SI Handler can reside in kernel or user mode. In the case that the handler resides in kernel mode, the operating system, to insure system integrity and security, may prevent an application from directly calling a function in kernel mode. Various mechanisms exist on the different operating systems to bypass this problem. A kernel mode DPVL-SI Handler **shall** implement the mechanism or mechanisms defined in the appropriate annex to allow calling a function in kernel mode, while a user mode handler will not implement this mechanism.

In the case that the handler resides in kernel mode, a unique, operating system specific, mechanism for the handler must be defined. Upon calling this mechanism, the data structure shown in Table 4-2 **shall** be filled in by the kernel mode handler. The structure provides the calling application with the specific mechanisms for each function which is necessary to take advantage of the application interface.

Parameter	Description
Version	Version number
Scene sync’ing	Mechanism code for Scene sync’ing function
Video Packet	Mechanism code for Video Packet function
Genlock	Mechanism code for Genlock function
Scaled Video Stream Setup	Mechanism code for Scaled Video Stream Setup function
Scaled Video Stream	Mechanism code for Scaled Video Stream function
Gamma Table	Mechanism code for Gamma Table function
Frame Buffer Control	Mechanism code for Frame Buffer Control function
BitBlt	Mechanism code for BitBlt function
Area Fill	Mechanism code for Area Fill function
Pattern Fill	Mechanism code for Pattern Fill function
Transparency Map Packet	Mechanism code for Transparency Map Packet function

Table 4-2: Parameters for retrieving the IOCTL codes of the various application interface functions

In the case where the specific feature is not supported by the handler, the value for the mechanism code on return from the handler **shall** be set to 0.

A kernel mode DPVL-SI Handler **shall** implement this call based on the mechanism or mechanisms described in the appropriate annex, while a user mode handler will not implement this call.

In the case that the handler resides in user mode, a direct function call for each feature can be made from the application to the user mode DPVL-SI Handler. A direct function call is the preferable mechanism since it is the most efficient call for system performance. For simplicity of retrieving the user mode function addresses a function shall be defined. Operating systems have a method for locating and calling a function, if its name is known. The function **shall** be named DPVLGetUserFeatures. The function **shall** pass to the handler a pointer to an array of storage locations that are described in Table 4-3.

Parameter	Description
Version	Version number
Scene sync'ing	Address of Scene sync'ing function
Video Packet	Address of Video Packet function
Genlock	Address of Genlock function
Scaled Video Stream Setup	Address of Scaled Video Stream Setup
Scaled Video Stream	Address of Scaled Video Stream function
Gamma Table	Address of Gamma Table function
Frame Buffer Control	Address of Frame Buffer Control function
BitBlt	Address of BitBlt function
Area Fill	Address of Area Fill function
Pattern Fill	Address of Pattern Fill function
Transparency Map Packet	Address of Transparancy Map Packet function

Table 4-3 : Parameters in the DPVLGetUserFeatures function

In the case where the specific feature is not supported, the function address on return **shall** be set to 0.

A user mode DPVL-SI Handler **shall** implement this function call, while a kernel mode handler will not implement this call.

4.3 Feature Function Parameters

4.3.1 Scene Sync'ing

The main purpose of this feature is to give the DPVL compliant host and monitor a tool to reduce tearing artifacts in the reconstruction of one scene within a sequence of scenes. A scene is a rectangular region or a group of rectangular regions on the screen which are to be updated during a single time slot. Examples of a scene could include an animation, a rotating CAD object or a video. For scene sync'ing to be implemented, two parameters one called "image buffer select" and the second called "last packet" must be defined. The image buffer select parameter defines which image buffer the pixel data contained in a packet is written to. Three choices are available in the DPVL standard, either to the image buffer refreshing the display module (this disables scene sync'ing), to an offline image buffer, or to all the image buffers. The last packet parameter defines the last packet of a scene to the monitor. When this parameter is set to indicate the last scene, the host **shall**

1. Immediately start writing the update rectangle within the packet to the current offline image buffer if the preceding image buffer select parameter indicated that image data was written to an offline image buffer
2. Swap the definition of the refresh/offline image buffer and write update rectangles within subsequent packets to the new offline image buffer if the image buffer select parameter is set to write pixel data to an offline buffer.
3. Begin displaying the new refresh image buffer immediately after the subsequent vertical sync signal of the display module

A kernel mode handler will implement this utilizing the operating system defined mechanism and its mechanism code returned from the handler in Table 4-2. When calling the mechanism, values for ImageBufferSelect and LastPacket **shall** be specified in the call.

For a user mode DPVL-SI Handler, the function is defined as follows:

```
SceneSync(imageBufferSelect, lastPacket)
```

It is called using the function address provided in Table 4-3.

4.3.2 Video Packet

The Video Packet carries a bitmap image to update a rectangular region of the screen. The size of this region may be as small as a single pixel or as large as the whole virtual screen. The parameters needed for this function are defined in Table 4-4.

Parameter	Description
Video Data Format	Monochrome, RGB Single Link, RGB Dual Link, RGB Sequential or RGB 5/4
X Left Edge	Left edge of update rectangle
Y Top Edge	Top edge of update rectangle
X Right Edge	Right edge of update rectangle
Y Bottom Edge	Bottom edge of update rectangle
Pixel Data Attr.	Stereo and Buffer control
Hoffset	Number of pixels on the left edge of the video frame to ignore.

Table 4-4 Parameters needed for implementing the Video Packet Function

A kernel mode handler will implement this utilizing the operating system defined mechanism and its mechanism code returned from the handler in Table 4-2. When calling the mechanism, values for the parameters **shall** be specified in the call.

For a user mode DPVL-SI Handler, the function is defined as follows:

```
VideoPacket(videoDataFormat, xLeftEdge, yTopEdge, xRightEdge,
yBottomEdge, pixelDataAttr, hOffset)
```

It is called using the function address provided in Table 4-3.

4.3.3 Genlock

In multi-monitor, stereo and motion picture applications, it is advantageous for the host to know when the monitor is starting a new frame. This standard provides the host with such a method by defining a frame synchronization mechanism using the timing reference signal called Genlock Packet. The parameters needed for this function are defined in Table 4-5.

Parameter	Description
Clock Frequency	Number of ticks per second (constant).
Clocks per Frame	Number of ticks per frame (constant)
Timestamp	Current counter content
Timestamp Accuracy	Timestamp accuracy is +/- this value

Table 4-5: Parameters needed for implementing the Genlock Function

A kernel mode handler will implement this utilizing the operating system defined mechanism and its mechanism code returned from the handler in Table 4-2. When calling the mechanism, values for the parameters **shall** be specified in the call.

For a user mode DPVL-SI Handler, the function is defined as follows:

```
Genlock(clockFrequency, clockPerFrame, timestamp,
timestampAccuracy)
```

It is called using the function address provided in Table 4-3.

4.3.4 Scaled Video Stream Setup

Whether in single-monitor systems or in multi-monitor systems, it is useful to send a video stream of the original size to the monitor(s) and have each monitor scale and position the image using unique values for each monitor. The packets for scaled video streams are slightly different than the Video Packet. The Video Packet consists of both the header section containing the entire context necessary to reconstruct the whole image using each bitmap image, and the body section containing an actual bitmap image. The Scaled Video Stream uses a setup packet to designate the size of the source bitmap image and the size and the position of the destination rectangle, and a separate Scaled Video Stream Packet to transmit the actual bitmap image.

The parameters needed for Scaled Video Stream setup are defined in Table 4-6.

Parameter	Description
Client ID	Monitor number
Stream ID	Unique number to correlate stream setups with streams
X Left Edge	Left edge of update rectangle
Y Top Edge	Top edge of update rectangle
X Right Edge	Right edge of update rectangle
Y Bottom Edge	Bottom edge of update rectangle
Image Size X	Horizontal size of source video image
Image Size Y	Vertical size of source video image
Video Data Format	Monochrome, RGB Single Link, RGB Dual Link, RGB Sequential or RGB 5/4

Table 4-6: Parameters needed for implementing theScaled Video Stream Setup Function

A kernel mode handler will implement this utilizing the operating system defined mechanism and its mechanism code returned from the handler in Table 4-2. When calling the mechanism, values for the parameters **shall** be specified in the call.

For a user mode DPVL-SI Handler, the function is defined as follows:

```
ScaledVideoStreamSetup (clientID, streamID, xLeftEdge, yTopEdge,
xRightEdge, yBottomEdge, imageSizeX, imageSizeY,
videoDataFormat)
```

It is called using the function address provided in Table 4-3.

4.3.5 Scaled Video Stream

The parameters needed for the Scaled Video Stream function are defined in Table 4-7.

Parameter	Description
Stream ID	Unique number to correlate stream setups with streams
Pixel Data Attr.	Stereo and Buffer control
Hoffset	Number of pixels on the left edge of the video frame to ignore.

Table 4-7: Parameters needed for implementing the Scaled Video Stream Function

A kernel mode handler will implement this utilizing the operating system defined mechanism and its mechanism code returned from the handler in Table 4-2. When calling the mechanism, values for the parameters **shall** be specified in the call.

For a user mode DPVL-SI Handler, the function is defined as follows:

```
ScaledVideoStream (streamID, pixelDataAttr, hoffset)
```
It is called using the function address provided in Table 4-3.

4.3.6 Gamma Table

The Gamma Table packet loads a downstream Look-Up-Table (LUT) in the monitor with a table of color values. Logically, the Gamma Table is placed between the frame buffer and the display module, however other implementations are possible.

The format of the table is specified by the *Video Data Format* parameter. The *ColorMap Offset* parameter tells the monitor which entry in the table is the first to be loaded. The *ColorMap Length* parameter tells the monitor how many entries will be loaded.

The parameters needed for the Gamma table function are defined in Table 4-8.

Parameter	Description
Client ID	Monitor number
Video Data Format	Format of colormap data
ColorMap Offset	Position to start writing values into colormap
ColorMap Length	Total number of entries to write into colormap
reserved	reserved, **shall** be set to zero.

Table 4-8: Parameters needed for implementing the Gamma Table Function

A kernel mode handler will implement this utilizing the operating system defined mechanism and its mechanism code returned from the handler in Table 4-2. When calling the mechanism, values for the parameters **shall** be specified in the call.

For a user mode DPVL-SI Handler, the function is defined as follows:

```
GammaTable (clientID, videoDataFormat, colorMapOffset,
colorMapLength, reserved)
```

It is called using the function address provided in Table 4-3.

4.3.7 Frame Buffer Control

The Frame Buffer Control Packet allows independent control of the displayed frame buffer in both monaural and stereo modes. The *Client ID* parameter specifies which monitor(s) the packet affects. The *Buffer Control* parameter specifies the action to be performed by the monitor. This packet enables and disables the scene sync'ing feature. By default, scene sync'ing is enabled (*Buffer Control* set to 0010h).

The parameters needed for the Frame Buffer Control function are defined in Table 4-9.

Parameter	Description
Client ID	Monitor number
Buffer Control	Selects which buffer to view
reserved	reserved, **shall** be set to zero.

Table 4-9: Parameters needed for implementing the Frame Buffer Control Function

A kernel mode handler will implement this utilizing the operating system defined mechanism and its mechanism code returned from the handler in Table 4-2. When calling the mechanism, values for the parameters **shall** be specified in the call.

For a user mode DPVL-SI Handler, the function is defined as follows:

```
FrameBufferControl (clientID, bufferControl, reserved)
```

It is called using the function address provided in Table 4-3.

4.3.8 BitBlt

The Bitmap Block Transfer (BitBlt) Packet provides a means to scroll or move regions of the screen in any direction. The destination region is specified by the update region parameters *(*X *Left Edge, Y Top Edge, X Right Edge and Y Bottom Edge*). The origin of the source rectangle is specified by the top-left source parameters (*Src X Left Edge and Src Y Top Edge*). The size of the source rectangle is defined to be the same size as the destination update rectangle. The *RasterOp* parameter specifies how the source pixels are merged into the existing contents in the frame buffer. If bits 9 and 8 of the *RasterOp* parameter are set to 00b, then the *Transparent Color* and *Transparent Mask* are not used, and all pixels of the source rectangle are copied to the destination. If bits 9 and 8 of the *RasterOp* parameter are set to 10b, then the *Transparent Color* and the *Transparent Mask* parameters specifies the color in the source update rectangle that is not copied to the destination rectangle. If bits 9 and 8 of the *RasterOp* parameter are set to 11b, then the *Transparent Color* and *Transparent Mask* parameters specify the color in the source update rectangle that is copied to the destination rectangle, and all other colors in the source rectangle are not copied.

The parameters needed for the Frame Buffer Control function are defined in Table 4-10.

Parameter	Description
Client ID	Monitor number
Pixel Data Attr.	Stereo and Buffer control
RasterOp	Indicates how to combine pixels ("xor", "or", "and", "copy" etc…)
X Left Edge	Left edge of update rectangle
Y Top Edge	Top edge of update rectangle
X Right Edge	Right edge of update rectangle
Y Bottom Edge	Bottom edge of update rectangle
Src X Left Edge	Left edge of source rectangle
Src Y Top Edge	Top edge of source rectangle
Transparent Color Red	Red value of the source color that is not copied to destination
Transparent Color Green	Green value of the source color that is not copied to destination
Transparent Color Blue	Blue value of the source color that is not copied to destination
Transparent Mask Red	Red value of the color to be used to mask the source color prior to comparison
Transparent Mask Green	Green value of the color to be used to mask the source color prior to comparison
Transparent Mask Blue	Blue value of the color to be used to mask the source color prior to comparison

Table 4-10: Parameters needed for implementing the BitBlt Function

A kernel mode handler will implement this utilizing the operating system defined mechanism and its mechanism code returned from the handler in Table 4-2. When calling the mechanism, values for the parameters **shall** be specified in the call.

For a user mode DPVL-SI Handler, the function is defined as follows:

```
BitBlt (clientID, pixelDataAttr, rasterOp, xLeftEdge, yTopEdge,
xRightEdge, yBottomEdge, srcXLeftEdge, srcYTopEdge,
transparentColorRed, transparentColorGreen,
transparentColorBlue, transparentMaskRed, transparentMaskGreen,
tansparentMaskBlue)
```

It is called using the function address provided in Table 4-3.

4.3.9 Area Fill

The Area Fill Packet provides a means to easily initialize a region of the frame buffer to a fixed color value. The update region is specified by the update rectangle parameters(*X Left Edge, Y Top Edge, X Right Edge and Y Bottom Edge)*. The *Fill Color* parameters specify the fixed color value used in the fill operation. The *RasterOp* parameter specifies how the fill color is applied to the existing contents in the frame buffer.

The parameters needed for the Frame Buffer Control function are defined in Table 4-11.

Parameter	Description
Client ID	Monitor number
Pixel Data Attr.	Stereo and Buffer control
RasterOp	Indicates how to combine pixels ("xor", "or", "and", "copy" etc…)
X Left Edge	Left edge of update rectangle
Y Top Edge	Top edge of update rectangle
X Right Edge	Right edge of update rectangle
Y Bottom Edge	Bottom edge of update rectangle
Fill Color Red	Destination rectangle set to this color.
Fill Color Green	Destination rectangle set to this color
Fill Color Blue	Destination rectangle set to this color
reserved	reserved, **shall** be set to zero.

Table 4-11: Parameters needed for implementing the Area Fill Function

A kernel mode handler will implement this utilizing the Operatingsystem defined mechanism and its mechanism code returned from the handler in Table 4-2. When calling the mechanism, values for the parameters **shall** be specified in the call.

For a user mode DPVL-SI Handler, the function is defined as follows:

```
AreaFill (clientID, pixelDataAttr, rasterOp, xLeftEdge,
yTopEdge, xRightEdge, yBottomEdge, fillColorRed, fillColorGreen,
fillColorBlue, reserved)
```

It is called using the function address provided in Table 4-3.

4.3.10 Pattern Fill

The Bitmap Pattern Fill Packet provides a means to easily initialize a region of the frame buffer to a pattern. The update region is specified by the update rectangle parameters (*X Left Edge, Y Top Edge, X Right Edge and Y Bottom Edge*). The pattern size is specified by the *Pattern Width* and *Pattern Height* parameters. The *Pattern Offset X* and *Pattern Offset Y* parameters specify the location in the pattern that should be aligned with the upper left of the destination rectangle. The format of the pattern data is specified by the *Video Data Format* parameter. If the destination rectangle is smaller than the fill pattern, then the pattern will be truncated to fit the destination rectangle. If the destination rectangle is larger than the fill pattern, the pattern will be replicated across the full destination rectangle. The *RasterOp* parameter specifies how the pattern data are merged into the existing contents in the frame buffer.

The parameters needed for the Frame Buffer Control function are defined in Table 4-12.

Parameter	Description
Client ID	Monitor number
Pixel Data Attr.	Stereo and Buffer control
RasterOp	Indicates how to combine pixels ("xor", "or", "and", "copy" etc…)
X Left Edge	Left edge of update rectangle
Y Top Edge	Top edge of update rectangle
X Right Edge	Right edge of update rectangle
Y Bottom Edge	Bottom edge of update rectangle
Pattern Width	Pixel width of pattern
Pattern Height	Pixel Height of pattern
Pattern Offset X	16 bit X position in pattern to draw at upper left of update rectangle
Pattern Offset Y	16 bit Y position in pattern to draw at upper left of update rectangle
Video Data Format	Monochrome, RGB Single Link, RGB Dual Link, RGB Sequential or RGB 5/4
Reserved	reserved, **shall** be set to zero.

Table 4-12: Parameters needed for implementing the Pattern Fill Function

A kernel mode handler will implement this utilizing the Operatingsystem defined mechanism and its mechanism code returned from the handler in Table 4-2. When calling the mechanism, values for the parameters **shall** be specified in the call.

For a user mode DPVL-SI Handler, the function is defined as follows:

```
PatternFill (clientID, pixelDataAttr, rasterOp, xLeftEdge,
yTopEdge, xRightEdge, yBottomEdge, patternWidth, patternHeight,
patternOffsetX, patternOffsetY, videoDataFormat, reserved)
```

It is called using the function address provided in Table 4-3.

4.3.11 Transparency Map Packet

The Transparency Map packet is used in conjunction with the Scaled Video Stream Setup packet to indicate how the Scaled Video Stream data should be blended with the existing frame buffer contents. The Transparency Map is assumed to be the same size as the visible frame buffer. The origin of the Transparency Map is defined to be the same as that of the Video Packet. The update region is specified by the update rectangle parameters (*X Left Edge*, *Y Top Edge*, *X Right Edge and Y Bottom Edge)*. The *Transp Map Depth* parameter specifies how many bits of transparency information are being sent in the body of this packet

The parameters needed for the Frame Buffer Control function are defined in Table 4-13.

Parameter	Description
Client ID	Monitor number
reserved	reserved, **shall** be set to zero.
X Left Edge	Left edge of update rectangle
Y Top Edge	Top edge of update rectangle
X Right Edge	Right edge of update rectangle
Y Bottom Edge	Bottom edge of update rectangle
Transp Map Depth	Number of bits in the transparency map

Table 4-13: Parameters needed for implementing the Transparency Map Function

A kernel mode handler will implement this utilizing the Operatingsystem defined mechanism and its mechanism code returned from the handler in Table 4-2. When calling the mechanism, values for the parameters **shall** be specified in the call.

For a user mode DPVL-SI Handler, the function is defined as follows:

```
TransparencyMap (clientID, reserved, xLeftEdge, yTopEdge,
xRightEdge, yBottomEdge, transpMapDepth)
```

It is called using the function address provided in Table 4-3.

5 Annex 1 Windows

The following section will describe the operating system dependent methods and structures for supporting DPVL-SI under Microsoft Windows.

5.1 Driver Interface/ Retrieving the Interface

For Microsoft Windows, communication between the DPVL-SI Handler and the video card driver will be handled through DrvEscape() and DrvDrawEscape() (Escape() and DrawEscape()) calls. Only one DrvEscape code is reserved by this specification, and is used by the DPVL-SI Handler to obtain the proprietary implementation defined by the video card driver.

5.1.1 DPVL_QUERY_IMPLEMENTATION (0xFACADE00)

This DrvEscape() code is reserved for the DPVL-SI Handler to obtain proprietary DPVL implementation information from the video card driver. It differs from DPVLGetInterface() in that it should only be called once, as the DPVL-SI Handler is loaded and that the information provided is not dependent on the arrangement of hardware in the system.

The DPVL-SI hander **shall** call DrvEscape() (0xFACADE00), passing the magic number 0x5AFEC0DE (as a safety check) and the desired implementation level as input. The video card driver **shall** verify the input and output block sizes as well as the magic number, then return its implementation level along with its proprietary Escape code for DPVLGetInterface. The input and output parameters are shown in Table 5-1.

Storage Types	Description
Input:	
DWORD	Magic Number
DWORD	Handler Implementation Level
Output:	
DWORD	Driver Implementation Level
DWORD	Escape Code for DPVLGetInterface

Table 5-1: Parameters for obtaining the escape code for DPVLGetInterface

5.1.2 DPVLGetInterface

Using the Escape code for the DPVLGetInterface function returned by DPVL_QUERY_IMPLEMENTATION, the DPVL-SI Handler can now obtain more information about the current state of the video card, number of devices attached, and further proprietary implementation details. This information may change during normal operation (for example by adding and removing hardware) and should be called again whenever changes are made. In Table 5-2, the output parameters obtained when calling the DPVLGetInterface escape code are listed.

Storage Types	Description
DWORD	Current Driver Implementation Level
DWORD	Attached Devices
DWORD	Escape Code for DPVLGetDeviceAttr function
DWORD	Escape Code for DPVLGetCrtcMap function
DWORD	Escape Code for DPVLGetCrtcAttr function
DWORD	Escape Code for DPVLEnable function
DWORD	Escape Code for DPVLSetCrt function
DWORD	Escape Code for DPVLValidateVirtualResolution function
DWORD	Escape Code for DPVLSetVirtualResolution function
DWORD	Escape Code for DPVLSetHWCursorShape function
DWORD	Escape Code for DPVLSetHWCursorPosition function
DWORD	Escape Code for DPVLSetSignaling function

Table 5-2: Parameters for obtaining the escape codes for the Driver interface functions

5.1.3 DPVLGetDeviceAttr

Using the Escape code for the DPVLGetDeviceAttr function returned by DPVLGetInterface, the DPVL-SI Handler may query detailed characteristics of each device attached to the video card. In Table 5-3, the input and output parameters when utilizing the DVLGetDeviceAttr escape call are listed.

Storage Types	Parameter
Input:	
DWORD	deviceIndex
Output:	
DWORD	deviceIndex
DWORD	Flags
DWORD	fbMaxWidth
DWORD	fbMaxHeight
DWORD	fbMaxSize
DWORD	fbWidthGranularity
DWORD	fbHeightGranularity

Storage Types	Parameter
DWORD	hwcWidth
DWORD	hwcHeight

Table 5-3: Parameters for the DPVLGetDeviceAttr Escape Call

In Table 5-4, the value of the flags returned by the DPVLGetDeviceAttr Escape call is shown.

Flag Value	Flag Name
0x00000001	DPVL_USE_HWC
0x00000002	DPVL_AUTO_HEADER
0x00000004	DPVL_PANNING
0x00000008	DPVL_NO_KERNEL_MODULE
0x00000010	DPVL_NO_USER_MODULE

Table 5-4: Flags returned by the DPVLGetDeviceAttr Escape call

5.1.4 DPVLGetCrtcMap

Using the escape code for the DPVLGetCrtcMap function returned by DPVLGetInterface, the DPVL-SI Handler can obtain an array containing information of which devices are attached to which CRT controller. The length of the array in bytes returned is equal to the number of devices returned by the DPVLGetInterface escape code call. Each element of the array is an unsigned byte.

5.1.5 DPVLGetCrtcAttr

Using the Escape code for the DPVLGetCrtcAttr function returned by DPVLGetInterface, the DPVL-SI Handler can call once for each unique integer returned by the DPVLGetCrtcMap function to retrieve parameters for the CRT controller. In Table 5-5, the parameters needed for calling the DPVLGetCrtcAttr escape code call are listed.

Storage Types	Parameter
Input:	
DWORD	iCrtcIndex
Output:	
DWORD	iCrtcIndex
DWORD	crtcMaxWidth
DWORD	crtcMaxHeight
DWORD	crtcMaxTotal
DWORD	crtcMaxSize
DWORD	crtcWidthGranularity

Storage Types	Parameter
DWORD	crtcHeightGranularity
DWORD	crtcMaxPanningX
DWORD	crtcMaxPanningY

Table 5-5: Parameters for the DPVLGetCrtcAttr Escape Call

5.1.6 DPVLEnable

Using the Escape code for the DPVLEnable function returned by DPVLGetInterface, the DPVL-SI Handler can call this Escape code function to transition between DPVL mode and raster mode.
In Table 5-6, the parameters needed for calling the DPVLEnable escape code call are listed.

Storage Types	Parameter
Input:	
DWORD	Index
BOOLEAN	Enable Flag
Output:	
BOOLEAN	Return Flag

Table 5-6: Parameters for the DPVLEnable Escape Call

When the Enable Flag is set to TRUE, the video card driver shall turn off raster mode and prepare to send DPVL packets. When the Enable Flag is set to FALSE, the video card driver shall turn off DPVL mode and transition to raster mode. A Return FLAG set to TRUE is returned if the transition was successful, otherwise the Return Flag shall be set to FALSE.

5.1.7 DPVLSetCrt

Using the Escape code for the DPVLSetCrt function returned by DPVLGetInterface, the DPVL-SI Handler can call this Escape code function to submit a DPVL update rectangle request to the display driver. In Table 5-7, the parameters needed for calling the DPVLSetCrt escape code call are listed.

Storage Types	Parameter
Input:	
DWORD	Index
DWORD	Flags
DWORD	Priority
DWORD	rclWidth

Storage Types	Parameter
DWORD	rclHeight
DWORD	rclTop
DWORD	rclLeft
DWORD	panRight
DWORD	panDown
DWORD	ulTimeout
Output:	
DWORD	ulReturn

Table 5-7: Parameters for the DPVLSetCrt Escape Call

The Flags parameter is defined as follows in Table 5-8.

Flag Value	Flag Name
0x00000000	DPVL_WAIT_FOR_UPDATE
0x00000001	DPVL_RETURN_IF_BUSY
0x00000002	DPVL_ASYNCHRONOUS
0x00000003	Reserved
0x00000004	DPVL_SIGNAL
0x00000008	DPVL_AUTO_HEADER
0x00000010	DPVL_PAN_ONLY

Table 5-8: Definition of the flags parameter in the DPVLSetCrt Escape Call

The Priority parameter is defined as follows in Table 5-9.

Priority Value	Priority Name
0x00000000	DPVL_PRIORITY_HIGH
0x00000001	DPVL_PRIORITY_NORMAL (Reserved)
0x00000002	DPVL_PRIORITY_LOW

Table 5-9: Definition of the priority parameter in the DPVLSetCrt Escape Call

Finally, the return value for the ulReturn parameter is defined as follows in Table 5-10.

Return Value	Return Value Name
0x00000000	DPVL_SETCRT_SUCCESS
0x00000001	DPVL_SETCRT_INVALID_INDEX
0x00000002	DPVL_SETCRT_UNSUPPORTED_FLAG

Return Value	Return Value Name
0x00000003	DPVL_SETCRT_INVALID_RECTANGLE
0x00000004	DPVL_SETCRT_OUT_OF_BOUNDS
0x00000100	DPVL_SETCRT_BUSY
0x00000101	DPVL_SETCRT_AUTO_HEADER_FAILURE
0x00000102	DPVL_SETCRT_TIMED_OUT

Table 5-10: Return Values from the DPVLSetCrt Escape Call

5.1.8 DPVLValidateVirtualResolution

Using the Escape code for the DPVLValidateVirtualResolution function returned by DPVLGetInterface, the DPVL-SI Handler can call this Escape code function to validate whether a given resolution can be supported for a given device. In Table 5-11, the parameters needed for calling the DPVLValidateVirtualResolution escape code call are listed.

Storage Types	Parameter
Input:	
DWORD	index
DWORD	width
DWORD	height
Output:	
Boolean	bReturn

Table 5-11: Parameters for the DPVLValidateVirtualResolution Escape Call

5.1.9 DPVLSetVirtualResolution

Using the Escape code for the DPVLSetVirtualResolution function returned by DPVLGetInterface, the DPVL-SI Handler can call this Escape code function to inform the display driver that the handler supports a given resolution. In Table 5-12, the parameters needed for calling the DPVLSetVirtualResolution escape code call are listed.

Storage Types	Parameter
Input:	
DWORD	index
DWORD	width
DWORD	height

Storage Types	Parameter
Output:	
Boolean	bReturn

Table 5-12: Parameters for the DPVLSetVirtualResolution Escape Call

The bReturn parameter allows the video card driver to accept (TRUE) or refuse to set (FALSE) the requested virtual resolution.

5.1.10 DPVLSetHWCursorShape

Using the Escape code for the DPVLSetHWCursorShape function returned by DPVLGetInterface, the DPVL-SI Handler can call this Escape code function to enable and reshape the hardware cursor. In Table 5-13, the parameters needed for calling the DPVLSetHWCursorShape escape code call are listed.

Storage Types	Parameter
Input:	
DWORD	index
VIDEO_POINTER_ATTRIBUTES	vpa
Output:	
Boolean	bReturn

Table 5-13: Parameters for the DPVLSetHWCursorShape Escape Call

The VIDEO_POINTER_ATTRIBUTES structure is defined in the Windows OS. The bReturn parameter allows the video card driver to accept or refuse to set the requested virtual resolution.

5.1.11 DPVLSetHWCursorPosition

Using the Escape code for the DPVLSetHWCursorPosition function returned by DPVLGetInterface, the DPVL-SI Handler can call this Escape code function to change the position of the hardware cursor. In Table 5-14, the parameters needed for calling the DPVLSetHWCursorPosition escape code call are listed.

Storage Types	Parameter
Input:	
DWORD	index
WORD	x
WORD	y
Output:	
Boolean	bReturn

Table 5-14: Parameters for the DPVLSetHWCursorPosition Escape Call

5.1.12 DPVLSetSignaling

Using the Escape code for the DPVLSetSignaling function returned by DPVLGetInterface, the DPVL-SI Handler can call this Escape code function so that the handler will be notified when a given packet has been successfully sent by the video card driver to the attached device. If signaling is requested the handler must create and provide a handle to an event to the display driver for signaling. In Table 5-15, the parameters needed for calling the DPVLSetSignaling escape code call are listed.

Storage Types	Parameter
Input:	
DWORD	Index
DWORD	flags
HANDLE	handleEvent
Output:	
Boolean	bReturn

Table 5-15: Parameters for the DPVLSetSignaling Escape Call

The flags parameter is defined as follows in Table 5-16.

Flag Value	Flag Name
0x00000000	DPVL_SIGNAL_NONE
0x00000001	DPVL_SIGNAL_PER_PACKET
0x00000002	DPVL_SIGNAL_QUEUE_EMPTY

Table 5-16: Definition of the flags parameter in the DPVLSetSignaling Escape Call

5.2 Application Interface

As described in section 2 of this document, the DPVL-SI Handler can reside in kernel or user mode. In the case that the handler resides in kernel mode, the operating system, to insure system integrity and security, may prevent an application from directly calling a function in kernel mode. Two mechanisms to support the Application interface, which are architected in Windows for communicating between user mode and kernel mode will be described. At least one of these mechanisms **shall** be supported for a DPVL-SI Handler residing in kernel mode. The first method, DrvEscape, has been discussed previously in this annex as the mechanism which **shall** be used for communication between the DPVL-SI Handler and the display device driver. The second method is the IOCTL method where a unique IOCTL is defined in the handler for support of the application interface.

Although both mechanisms are allowed, the decision of which mechanism or mechanisms to support will be left to the DPVL-SI Handler writer. There are advantages and disadvantages to either mechanism. For example, the DrvEscape mechanism is used for proprietary calls to the display driver and thus the magic number chosen here may not be unique. Though only a remote chance, employing this mechanism may break an application which just happens to use such a magic number for communicating with its display driver. On the other hand, utilizing the IOCTL mechanism would force the DPVL-SI Handler to have a standard kernel mode component, since a display driver cannot access the kernel function support for employing and accessing IOCTL calls. Though it may be necessary for the DPVL-SI function to have a kernel component for accessing the USB stack in kernel mode for a level 1 or 2 compliant DPVL monitor, such an implementation may not lead to a fully certified Microsoft driver.

5.2.1 Mechanisms

5.2.1.1 DrvEscape Mechanism

Only one DrvEscape code is reserved by the Application Interface if the DrvEscape mechanism is employed. The DPVL-SI hander **shall** call DrvEscape() (0xFACADE01), passing the magic number 0x5AFEC0DE (as a safety check). Upon receiving such a code the DPVL-SI Handler **shall** verify the input and output block sizes as well as the magic number, then return the proprietary Escape codes for all the supported features in the Application Interface. This DrvEscape **shall** not be forwarded to the display driver.

When calling the above DrvEscape a storage area of 23 words of the following array shown in Table 5-17 must be provided for the output block: If less than 23 words are provided to the handler, the handler **shall** fail this command.

Storage types	Description
WORD	Version number
DWORD	DrvEscape code for Scene sync'ing function
DWORD	DrvEscape code for Video Packet function
DWORD	DrvEscape code for Genlock function
DWORD	DrvEscape code for Scaled Video Stream Setup function
DWORD	DrvEscape code for Scaled Video Stream function
DWORD	DrvEscape code for Gamma Table function
DWORD	DrvEscape code for Frame Buffer Control function
DWORD	DrvEscape code for BitBlt function
DWORD	DrvEscape code for Area Fill function
DWORD	DrvEscape code for Pattern Fill function
DWORD	DrvEscape code for Transparency Map Packet function

Table 5-17: Parameters for retrieving the DrvEscape codes of the various application interface functions

If a specific feature is unsupported a value of 0 **shall** be placed in the DrvEscape code of the function upon return.

5.2.1.2 IOCTL Mechanism

A kernel mode handler may implement a unique IOCTL in the handler such that the application may retrieve the IOCTLs necessary to access the application interface. A 32 bit unique IOCTL is nominally defined in windows using the MACRO, CTL_CODE. The MACRO is defined as follows:

```
#define CTL_CODE(DeviceType, Function, Method, Access) (

  ((DeviceType) << 16) | ((Access) << 14) | ((Function) << 2) | (Method)
```

)

The first parameter, DeviceType, is a 16 bit parameter. For devices not typically found in Windows system the Microsoft defined value, FILE_DEVICE_UNKNOWN is typically used. It is defined in the Microsoft header files as follows:

#define FILE_DEVICE_UNKNOWN 22h

DeviceType **shall** be defined as FILE_DEVICE_UNKNOWN.

The second parameter Access defines the access check value for any access. This **shall** be defined as FILE_ANY_ACCESS. It is defined in the Microsoft header files as follows:

#define FILE_ANY_ACCESS 0h

The third parameter, function defines an action within the device category. It is a 12 bit value. Microsoft reserves values between 800h and FFFh for OEMs and IHVs. This parameter **shall** be defined as 800h.

The fourth and final parameter is Method. It **shall** be defined as METHOD_BUFFERED which is typically used in IOCTL calls. It is defined in the Microsoft header files as follows:

#define METHOD_BUFFERED 0h.

When calling the above IOCTL, a storage area of 23 words of the following array shown in Table 5-18 must be provided:

Storage types	Description
WORD	Version number
DWORD	IOCTL code for Scene sync'ing function
DWORD	IOCTL code for Video Packet function
DWORD	IOCTL code for Genlock function
DWORD	IOCTL code for Scaled Video Stream Setup function
DWORD	IOCTL code for Scaled Video Stream function
DWORD	IOCTL code for Gamma Table function
DWORD	IOCTL code for Frame Buffer Control function
DWORD	IOCTL code for BitBlt function
DWORD	IOCTL code for Area Fill function
DWORD	IOCTL code for Pattern Fill function
DWORD	IOCTL code for Transparency Map Packet function

Table 5-18: Parameters for retrieving the IOCTL codes of the various application interface functions

If less than 23 words of data are sent to the handler, the handler **shall** fail this command.

If a specific feature is unsupported a value of 0 **shall** be placed in the IOCTL code of the function upon return.

5.2.1.3 User Mode Handler

A user mode handler is required to implement the function DPVLGetUserFeatureAddr. The function **shall** pass a pointer to the following data array:

Table Headings	Table Headings
WORD	Version number
void *	Address of Scene sync'ing function
void *	Address of Video Packet function
void *	Address of Genlock function
void *	Address of Scaled Video Stream Setup function
void *	Address of Scaled Video Stream function
void *	Address of Gamma Table function
void *	Address of Frame Buffer Control function
void *	Address of BitBlt function
void *	Address of Area Fill function
void *	Address of Pattern Fill function
void *	Address of Transparency Map Packet function

Table 5-19 : Parameters in the DPVLGetUserFeatures function

If a specific feature is unsupported a value of 0 **shall** be placed in the address of the function upon return.

5.2.2 Feature Function Parameters

In the sections below the parameters needed for the feature functions described in Section 4-3 as they would appear in a Windows implementation are described.

5.2.2.1 Scene Sync'ing

The parameters used in the SceneSync function are defined in Table 5-20.

Storage Types	Parameter
byte	ImageBufferSelect ‘0’ - pixel data is written to the image buffer refreshing the display module (= scene sync’ing for the packet is disabled) “1” - pixel data is written to an offline image buffer (= scene sync’ing for the packet is enabled) “3”: pixel data is written to all the image buffers
byte	LastBuffer “1” – set (last Buffer) “0” – not set (not last buffer)

Table 5-20 Parameters used in the SceneSync function

For a kernel mode implementation, two bytes of data, the first for ImageBufferSelect and the second for LastBuffer must be specified using one of the mechanisms described earlier. If less than two bytes of data are sent to the handler, the handler shall fail this command.

For a user mode implementation, the function is called as follows:

```
int SceneSync(byte ImageBufferSelect,byte LastPacket)
```

The function returns 1 upon successful completion and 0 otherwise.

5.2.2.2 Video Packet

The parameters used in the VideoPacket function are defined in Table 5-21.

Storage Types	Parameter
WORD	Video Data Format
WORD	X Left Edge
WORD	Y Top Edge
WORD	X Right Edge
WORD	Y Bottom Edge
WORD	Pixel Data Attr.
WORD	Hoffset

Table 5-21 Parmeters used in the VideoPacket function

For a kernel mode implementation seven words of data must be specified using one of the mechanisms described earlier. If less than seven words of data are sent to the handler, the handler **shall** fail this command.

For a user mode implementation, the function is called as follows:

```
int VideoPacket(word VideoDataFormat, WORD XLeftEdge, WORD
YTopEdge, WORD XRightEdge, WORD YBottomEdge, WORD PixelDataAttr,
WORD HOffset);
```

The function returns 1 upon successful completion and 0 otherwise.

5.2.2.3 Genlock

The parameters used in the Genlock function are defined in Table 5-22.

Storage Types	Parameter
DWORD	Clock Frequency
DWORD	Clocks per Frame
DWORD	Timestamp
WORD	Timestamp Accuracy

Table 5-22: Parmeters used in the Genlock function

For a kernel mode implementation seven words of data must be specified using one of the mechanisms described earlier. If less than seven words of data are sent to the handler, the handler **shall** fail this command.

For a user mode implementation, the function is called as follows:

```
int Genlock(DWORD ClockFrequency, DWORD ClockPerFrame, DWORD
Timestamp, WORD TimestampAccuracy)
```

The function returns 1 upon successful completion and 0 otherwise.

5.2.2.4 Scaled Video Stream Setup

The parameters used in the ScaledVideoStreamSetup function are defined in Table 5-23

Storage Types	Parameter
WORD	Client ID
WORD	Stream ID
WORD	X Left Edge
WORD	Y Top Edge
WORD	X Right Edge
WORD	Y Bottom Edge
WORD	Image Size X

Storage Types	Parameter
WORD	Image Size Y
WORD	Video Data Format

Table 5-23: Parameters used in the ScaledVideoStreamSetup function

For a kernel mode implementation nine words of data must be specified using one of the mechanisms described earlier. If less than nine words of data are sent to the handler, the handler **shall** fail this command.

For a user mode implementation, the function is called as follows:

```
int ScaledVideoStreamSetup (WORD ClientID, WORD StreamID, WORD
XLeftEdge, WORD YTopEdge, WORD XRightEdge, WORD YBottomEdge,
WORD ImageSizeX, WORD ImageSizeY, WORD VideoDataFormat)
```

The function returns 1 upon successful completion and 0 otherwise.

5.2.2.5 Scaled Video Stream

The parameters used in the ScaledVideoStream function are defined in Table 5-24

Storage Types	Parameter
WORD	Stream ID
WORD	Pixel Data Attr.
WORD	Hoffset

Table 5-24: Parameters used in the ScaledVideoStream function

For a kernel mode implementation three words of data must be specified using one of the mechanisms described earlier. If less than three words of data are sent to the handler, the handler **shall** fail this command.

For a user mode implementation, the function is called as follows:

```
int ScaledVideoStream (WORD StreamID, WORD PixelDataAttr, WORD
Hoffset)
```

The function returns 1 upon successful completion and 0 otherwise.

5.2.2.6 Gamma Table

The parameters used in the GammaTable function are defined in Table 5-25.

Storage Types	Parameter
WORD	Client ID
WORD	Video Data Format
DWORD	ColorMap Offset
DWORD	ColorMap Length
WORD	reserved

Table 5-25: Parameters used in the GammaTable function

For a kernel mode implementation seven words of data must be specified using one of the mechanisms described earlier. If less than seven words of data are sent to the handler, the handler **shall** fail this command.

For a user mode implementation, the function is called as follows:

```
int GammaTable (WORD ClientID, WORD VideoDataFormat, DWORD
ColorMapOffset, DWORD ColorMapLength, WORD reserved)
```

The function returns 1 upon successful completion and 0 otherwise.

5.2.2.7 Frame Buffer Control

The parameters used in the Frame Buffer Control function are defined in Table 5-26.

Storage Types	Parameter
WORD	Client ID
WORD	Buffer Control
WORD	reserved

Table 5-26: Parameters used in the FrameBufferControl function

For a kernel mode implementation three words of data must be specified using one of the mechanisms described earlier. If less than three words of data are sent to the handler, the handler **shall** fail this command.

For a user mode implementation, the function is called as follows:

```
int FrameBufferControl (WORD ClientID, word BufferControl, word
reserved)
```

The function returns 1 upon successful completion and 0 otherwise.

5.2.2.8 BitBlt

The parameters used in the BitBlt function are defined in Table 5-27.

Storage Types	Parameter
WORD	Client ID
WORD	Pixel Data Attr.
WORD	RasterOp
WORD	X Left Edge
WORD	Y Top Edge
WORD	X Right Edge
WORD	Y Bottom Edge
WORD	Src X Left Edge
WORD	Src Y Top Edge
WORD	Transparent Color Red
WORD	Transparent Color Green
WORD	Transparent Color Blue
WORD	Transparent Mask Red
WORD	Transparent Mask Green
WORD	Transparent Mask Blue

Table 5-27: Parameters used in the BitBlt function

For a kernel mode implementation fifteen words of data must be specified using one of the mechanisms described earlier. If less than fifteen words of data are sent to the handler, the handler **shall** fail this command.

For a user mode implementation, the function is called as follows:

```
int BitBlt (WORD ClientID, WORD PixelDataAttr, WORD RasterOp,
WORD XLeftEdge, WORD YTopEdge, WORD XRightEdge, WORD
YBottomEdge, WORD SrcXLeftEdge, WORD SrcYTopEdge, WORD
TransparentColorRed, WORD TransparentColorGreen, WORD
TransparentColorBlue, WORD TransparentMaskRed, WORD
TransparentMaskGreen, WORD TransparentMaskBlue)
```

The function returns 1 upon successful completion and 0 otherwise.

5.2.2.9 Area Fill

The parameters used in the Area Fill function are defined in Table 5-28.

Storage Types	Parameter
WORD	Client ID
WORD	Pixel Data Attr.
WORD	RasterOp

Storage Types	Parameter
WORD	X Left Edge
WORD	Y Top Edge
WORD	X Right Edge
WORD	Y Bottom Edge
WORD	Fill Color Red
WORD	Fill Color Green
WORD	Fill Color Blue
WORD	reserved

Table 5-28: Parameters used in the Area Fill function

For a kernel mode implementation eleven words of data must be specified using one of the mechanisms described earlier. If less than eleven words of data are sent to the handler, the handler **shall** fail this command.

For a user mode implementation, the function is called as follows:

```
int AreaFill (WORD ClientID, WORD PixelDataAttr, WORD RasterOp,
WORD XLeftEdge, WORD YTopEdge, WORD XRightEdge, WORD
YBottomEdge, WORD FillColorRed, WORD FillColorGreen, WORD
FillColorBlue, WORD reserved)
```

The function returns 1 upon successful completion and 0 otherwise.

5.2.2.10 Pattern Fill

The parameters used in the Pattern Fill function are defined in Table 5-29.

Storage Types	Parameter
WORD	Client ID
WORD	Pixel Data Attr.
WORD	RasterOp
WORD	X Left Edge
WORD	Y Top Edge
WORD	X Right Edge
WORD	Y Bottom Edge
WORD	Pattern Width
WORD	Pattern Height
WORD	Pattern Offset X

Storage Types	Parameter
WORD	Pattern Offset Y
WORD	Video Data Format
WORD	reserved

Table 5-29: Parameters used in the Pattern Fill function

For a kernel mode implementation thirteen words of data must be specified using one of the mechanisms described earlier. If less than thirteen words of data are sent to the handler, the handler **shall** fail this command.

For a user mode implementation, the function is called as follows:

```
int PatternFill (WORD ClientID, WORD PixelDataAttr, WORD
RasterOp, WORD XLeftEdge, WORD YTopEdge, WORD XRightEdge, WORD
YBottomEdge, WORD PatternWidth, WORD PatternHeight, WORD
PatternOffsetX, WORD PatternOffsetY, WORD VideoDataFormat, WORD
reserved)
```

The function returns 1 upon successful completion and 0 otherwise.

5.2.2.11 Transparency Map Packet

The parameters used in the Transparency Map function are defined in Table 5-30.

Storage Types	Parameter
WORD	Client ID
WORD	Reserved
WORD	X Left Edge
WORD	Y Top Edge
WORD	X Right Edge
WORD	Y Bottom Edge
WORD	Transp Map Depth

Table 5-30: Parameters used in the Transparency Map function

For a kernel mode implementation seven words of data must be specified using one of the mechanisms described earlier. If less than seven words of data are sent to the handler, the handler **shall** fail this command.

For a user mode implementation, the function is called as follows:

int TransparencyMap (WORD ClientID, WORD reserved, WORD XLeftEdge, WORD YTopEdge, WORD XRightEdge, WORD YBottomEdge, WORD TranspMapDepth)

The function returns 1 upon successful completion and 0 otherwise.

6 Annex 2 Linux

6.1 Driver Interface

For Linux, as shown in Figure 2-3, both the DPVL-SI Handler and DPVL extension functions in the video card driver module reside in user mode, as part of the X server. The DPVL-SI Handler will be implemented as an extension module to the X server, which can be either be loaded automatically or through a “Load” directive in the X server configuration file. The DPVL-SI Handler **shall** implement and export the function DPVLInitDriverInterface. The function shall be called as follows:

```
int DPVLInitDriverInterface(DWORD HandlerImplementationLevel,
DPVL_GET_INTERFACE * DPVLGetInterface);
```

The DPVL_GET_INTERFACE structure is defined in Table 6-1. When the video card driver is initialized, it **shall** fill in the DPVL_GET_INTERFACE structure, and pass it to the DPVL-SI Handler using DPVLInitDriverInterface. After DPVLInitDriverInterface is called, the DPVL-SI Handler **shall** use the information passed in the DPVL_GET_INTERFACE structure to perform all future communication with the video card driver.

Storage Types	Description
DWORD	Current Driver Implementation Level
DWORD	Attached Devices
void *	Address of DPVLGetDeviceAttr function
void *	Address of DPVLGetCrtcMap function
void *	Address of DPVLGetCrtcAttr function
void *	Address of DPVLEnable function
void *	Address of DPVLSetCrt function
void *	Address of DPVLValidateVirtualResolution function
void *	Address of DPVLSetVirtualResolution function
void *	Address of DPVLSetHWCursorShape function
void *	Address of DPVLSetHWCursorPosition function
void *	Address of DPVLSetSignaling function

Table 6-1: Definition of the DPVL_GET_INTERFACE data structure

6.1.1 DPVLGetDeviceAttr

Using the function address for the DPVLGetDeviceAttr function returned by the DPVLGetDriverInterfaceAddr function, the DPVL-SI Handler may query detailed characteristics of each device attached to the video card. The DPVLGetDeviceAttr function is called as follows:

```
int DPVLGetDeviceAttr(DWORD deviceIndex,
DPVL_GET_DEVICE_ATTR * DPVLGetDeviceAttr);
```

The DPVL_GET_DEVICE_ATTR structure is defined in Table 6-2.

Storage Types	Parameter
DWORD	deviceIndex
DWORD	Flags
DWORD	fbMaxWidth
DWORD	fbMaxHeight
DWORD	fbMaxSize
DWORD	fbWidthGranularity
DWORD	fbHeightGranularity
DWORD	hwcWidth
DWORD	hwcHeight

Table 6-2: Definition of the DPVL_GET_INTERFACE data structure

In Table 6-3, the value of the flags returned by the DPVLGetDeviceAttr function call is shown.

Flag Value	Flag Name
0x00000001	DPVL_USE_HWC
0x00000002	DPVL_AUTO_HEADER
0x00000004	DPVL_PANNING
0x00000008	DPVL_NO_KERNEL_MODULE
0x00000010	DPVL_NO_USER_MODULE

Table 6-3: Flags returned by the DPVLGetDeviceAttr function call

6.1.2 DPVLGetCrtcMap

Using the function address for the DPVLGetCrtcMap function returned by the DPVLGetDriverInterfaceAddr function, the DPVL-SI Handler can obtain an array containing information of which devices are attached to which CRT controller. The length of the array in bytes returned is equal to the number of devices returned by the DPVLGetDriverInterfaceAddr function call. Each element of the array is an unsigned byte.

6.1.3 DPVLGetCrtcAttr

Using the function address for the DPVLGetCrtcAttr function returned by the DPVLGetDriverInterfaceAddr function, the DPVL-SI Handler can call once for each unique integer returned by the DPVLGetCrtcMap function, to retrieve parameters for the CRT controller. The DPVLGetCrtcAttr function is called as follows:

```
int DPVLGetCrtcAttr(DWORD iCrtcIndex,

DPVL_GET_CRTC_ATTR * DPVLGetCrtcAttr);
```

The DPVL_GET_CRTC_ATTR structure is defined in Table 6-4.

Storage Types	Parameter
DWORD	iCrtcIndex
DWORD	crtcMaxWidth
DWORD	crtcMaxHeight
DWORD	crtcMaxTotal
DWORD	crtcMaxSize
DWORD	crtcWidthGranularity
DWORD	crtcHeightGranularity
DWORD	crtcMaxPanningX
DWORD	crtcMaxPanningY

Table 6-4: Definition of the DPVL_GET_CRTC_ATTR data structure

6.1.4 DPVLEnable

Using the function address for the DPVLEnable function returned by the DPVLGetDriverInterfaceAddr function, the DPVL-SI Handler can call this function to transition between DPVL mode and raster mode. The DPVLEnable function is called as follows:

```
BOOLEAN DPVLEnable(DWORD index, BOOLEAN enableFlag);
```

When enableFlag is set to TRUE, the video card driver shall turn off raster mode and prepare to send DPVL packets. When enableFlag is set to FALSE, the video card driver shall turn off DPVL mode and transition to raster mode. The function returns TRUE if the transition was successful, otherwise the function returns FALSE.

6.1.5 DPVLSetCrt

Using the function address for the DPVLSetCrt function returned by the DPVLGetDriverInterfaceAddr function, the DPVL-SI Handler can call this function to submit a DPVL update rectangle request to the display driver. The DPVLSetCrt function is called as follows:

```
DWORD DPVLSetCrt(DWORD index, DWORD flags, DWORD priority, DWORD
rclWidth, DWORD rclHeight, DWORD rclTop, DWORD rclLeft,  DWORD
panRight, DWORD panDown, DWORD ulTimeout);
```

The flags parameter is defined as follows in Table 6-5.

Flag Value	Flag Name
0x00000000	DPVL_WAIT_FOR_UPDATE
0x00000001	DPVL_RETURN_IF_BUSY
0x00000002	DPVL_ASYNCHRONOUS
0x00000003	Reserved
0x00000004	DPVL_SIGNAL
0x00000008	DPVL_AUTO_HEADER
0x00000010	DPVL_PAN_ONLY

Table 6-5: Definition of the flags parameter in the DPVLSetCrt function call

The priority parameter is defined as follows in Table 6-6.

Priority Value	Priority Name
0x00000000	DPVL_PRIORITY_HIGH
0x00000001	DPVL_PRIORITY_NORMAL (Reserved)
0x00000002	DPVL_PRIORITY_LOW

Table 6-6: Definition of the priority parameter in the DPVLSetCrt function call

Finally, the DWORD return value for the DPVLSetCrt function is defined as follows in Table 6-7.

Return Value	Return Value Name
0x00000000	DPVL_SETCRT_SUCCESS
0x00000001	DPVL_SETCRT_INVALID_INDEX
0x00000002	DPVL_SETCRT_UNSUPPORTED_FLAG
0x00000003	DPVL_SETCRT_INVALID_RECTANGLE
0x00000004	DPVL_SETCRT_OUT_OF_BOUNDS
0x00000100	DPVL_SETCRT_BUSY
0x00000101	DPVL_SETCRT_AUTO_HEADER_FAILURE
0x00000102	DPVL_SETCRT_TIMED_OUT

Table 6-7: Return Values from the DPVLSetCrt function call

6.1.6 DPVLValidateVirtualResolution

Using the function address for the DPVLValidateVirtualResolution function returned by the DPVLGetDriverInterfaceAddr function, the DPVL-SI Handler can call this function to validate whether a given resolution can be supported for a given device.

The DPVLValidateVirtualResolution function is called as follows:

```
BOOLEAN DPVLValidateVirtualResolution(DWORD Index, DWORD width,
DWORD height);
```

The function returns TRUE if the resolution is supported and FALSE otherwise.

6.1.7 DPVLSetVirtualResolution

Using the function address for the DPVLSetVirtualResolution function returned by the DPVLGetDriverInterfaceAddr function, the DPVL-SI Handler can call this function to inform the display driver that the handler supports a given resolution.

The DPVLSetVirtualResolution function is called as follows:

```
BOOLEAN DPVLSetVirtualResolution(DWORD index, DWORD width, DWORD
height);
```

The function return parameter allows the video card driver to accept (TRUE) or refuse to set (FALSE) the requested virtual resolution.

6.1.8 DPVLSetHWCursorShape

Using the function address for the DPVLSetHWCursorShape function returned by the DPVLGetDriverInterfaceAddr function, the DPVL-SI Handler can call this function to enable and reshape the hardware cursor.

The DPVLSetHWCursorShape function is called as follows:

```
BOOLEAN DPVLSetHWCursorShape(DWORD index, xf86CursorInfo xci);
```

Where the xf86CursorInfo structure is defined in the XF86 sources. The function returns TRUE if successful, otherwise the function returns FALSE.

6.1.9 DPVLSetHWCursorPosition

Using the function address for the DPVLSetHWCursorPosition function returned by the DPVLGetDriverInterfaceAddr function, the DPVL-SI Handler can call this function to change the position of the hardware cursor.

The DPVLSetHWCursorPosition function is called as follows:

```
BOOLEAN DPVLSetHWCursorPosition(DWORD index, word x, word y);
```

The function returns TRUE if successful, otherwise the function returns FALSE.

6.1.10 DPVLSetSignaling

Using the function address for the DPVLSetSignaling function returned by the DPVLGetDriverInterfaceAddr function, the DPVL-SI Handler can call this function so that the handler will be notified when a given packet has been successfully sent by the video card driver to the attached device.

The DPVLSetSignaling function is called as follows:

```
BOOLEAN DPVLSetSignaling (DWORD index, DWORD flags);
```

The function returns TRUE if successful, otherwise the function returns FALSE. The DPVL-SI Handler must also register itself for SIGIO signal notification and a SIGIO signal is sent to notify the handler.

The flags parameter is defined as follows in Table 6-8.

Flag Value	Flag Name
0x00000000	DPVL_SIGNAL_NONE
0x00000001	DPVL_SIGNAL_PER_PACKET
0x00000002	DPVL_SIGNAL_QUEUE_EMPTY

Table 6-8: Definition of the flags parameter in the DPVLSetSignaling function Call

6.2 Application Interface

For Linux, as shown in Figure 2-3, the handler resides in user mode, and it is implemented as an application library and an extension to the X protocol. The DPVL X extension will be used to provide DPVL-specific communication between the application process and the DPVL-SI Handler residing in the X server. The application library will be used to provide a standard application interface to the DPVL X extension. The application library **shall** implement and export the function DPVLGetUserFeatureAddr. An application **shall** call this function and as parameters pass a pointer to a valid and active X server connection (represented by the Display structure) and a pointer to a DPVL_GET_USER_FEATURES structure as defined in Table 6-9.

```
int DPVLGetUserFeatureAddr(Display *dpy, DPVL_GET_USER_FEATURES
* DPVLGetUserFeatures);
```

Storage types	Description
word	Version number
void *	Address of Scene sync'ing function
void *	Address of Video Packet function
void *	Address of Genlock function
void *	Address of Scaled Video Stream Setup function
void *	Address of Scaled Video Stream function
void *	Address of Gamma Table function
void *	Address of Frame Buffer Control function
void *	Address of BitBlt function
void *	Address of Area Fill function
void *	Address of Pattern Fill function
void *	Address of Transparency Map Packet function

Table 6-9: Definition of the DPVL_GET_USER_FEATURES data structure

The application library **shall** fill in the data structure upon return. The library **shall** verify that the DPVL extension is supported by the passed X server connection and use that connection to communicate with the DPVL-SI Handler for additional information it may require to fill in the data structure appropriately. If a specific feature is unsupported a value of 0 **shall** be placed in the address of the function upon return. The function returns 1 upon successful completion and 0 otherwise. Since the application library needs the Display handle for all communication with the DPVL-SI Handler, it **should** choose to save it before returning to the application.

6.2.1 Feature Function Parameters

In the sections below the parameters needed for the feature functions described in Section 4-3 as they would appear in a Linux implementation are described.

6.2.1.1 Scene Sync'ing

The parameters used in the SceneSync function are defined in Table 6-10.

Storage Types	Parameter
byte	ImageBufferSelect '0' - pixel data is written to the image buffer refreshing the display module (= scene sync'ing for the packet is disabled) "1" - pixel data is written to an offline image buffer (= scene sync'ing for the packet is enabled)

Storage Types	Parameter
	"3": pixel data is written to all the image buffers
byte	LastBuffer "1" – set (last Buffer) "0" – not set (not last buffer)

Table 6-10 Parmeters used in the SceneSync function

The function is called as follows:

```
   int SceneSync(byte ImageBufferSelect,byte LastPacket)
```

The function returns 1 upon successful completion and 0 otherwise.

6.2.1.2 Video Packet

The parameters used in the VideoPacket function are defined in Table 6-11.

Storage Types	Parameter
word	Video Data Format
word	X Left Edge
word	Y Top Edge
word	X Right Edge
word	Y Bottom Edge
word	Pixel Data Attr.
word	Hoffset

Table 6-11: Parameters used in the VideoPacket function

The function is called as follows:

```
int VideoPacket(word VideoDataFormat, word XLeftEdge, word
YTopEdge, word XRightEdge, word YBottomEdge, word PixelDataAttr,
word HOffset);
```

The function returns 1 upon successful completion and 0 otherwise.

6.2.1.3 Genlock

The parameters used in the Genlock function are defined in Table 6-12

Storage Types	Parameter
DWORD	Clock Frequency
DWORD	Clocks per Frame
DWORD	Timestamp
word	Timestamp Accuracy

Table 6-12: Parmeters used in the Genlock function

The function is called as follows:

```
int Genlock(DWORD long ClockFrequency, DWORD ClockPerFrame,
DWORD Timestamp, word TimestampAccuracy)
```

The function returns 1 upon successful completion and 0 otherwise.

6.2.1.4 Scaled Video Stream Setup

The parameters used in the ScaledVideoStreamSetup function are defined in Table 6-13.

Storage Types	Parameter
word	Client ID
word	Stream ID
word	X Left Edge
word	Y Top Edge
word	X Right Edge
word	Y Bottom Edge
word	Image Size X
word	Image Size Y
word	Video Data Format

Table 6-13: Parameters used in the ScaledVideoStreamSetup function

The function is called as follows:

```
int ScaledVideoStreamSetup (word ClientID, word StreamID, word
XLeftEdge, word YTopEdge, word XRightEdge, word YBottomEdge,
word ImageSizeX, word ImageSizeY, word VideoDataFormat)
```

The function returns 1 upon successful completion and 0 otherwise.

6.2.1.5 *Scaled Video Stream*

The parameters used in the ScaledVideoStream function are defined in Table 6-14

Storage Types	Parameter
word	Stream ID
word	Pixel Data Attr.
word	Hoffset

Table 6-14: Parameters used in the ScaledVideoStream function

The function is called as follows:

```
int ScaledVideoStream (word StreamID, word PixelDataAttr, word
Hoffset)
```

The function returns 1 upon successful completion and 0 otherwise.

6.2.1.6 *Gamma Table*

The parameters used in the GammaTable function are defined in Table 6-15.

Storage Types	Parameter
word	Client ID
word	Video Data Format
DWORD	ColorMap Offset
DWORD	ColorMap Length
word	reserved

Table 6-15: Parameters used in the GammaTable function

The function is called as follows:

```
int GammaTable (word ClientID, word VideoDataFormat, DWORD
ColorMapOffset, DWORD ColorMapLength, word reserved)
```

The function returns 1 upon successful completion and 0 otherwise.

6.2.1.7 *Frame Buffer Control*

The parameters used in the Frame Buffer Control function are defined in Table 6-16.

Storage Types	Parameter
word	Client ID
word	Buffer Control
word	reserved

Table 6-16: Parameters used in the FrameBufferControl function

The function is called as follows:

```
int FrameBufferControl (word ClientID, word BufferControl, word
reserved)
```

The function returns 1 upon successful completion and 0 otherwise.

6.2.1.8 BitBlt

The parameters used in the BitBlt function are defined in Table 6-17.

Storage Types	Parameter
word	Client ID
word	Pixel Data Attr.
word	RasterOp
word	X Left Edge
word	Y Top Edge
word	X Right Edge
word	Y Bottom Edge
word	Src X Left Edge
word	Src Y Top Edge
word	Transparent Color Red
word	Transparent Color Green
word	Transparent Color Blue
word	Transparent Mask Red
word	Transparent Mask Green
word	Transparent Mask Blue

Table 6-17: Parameters used in the BitBlt function

The function is called as follows:

```
int BitBlt (word ClientID, word PixelDataAttr, word RasterOp,
word XLeftEdge, word YTopEdge, word XRightEdge, word
YBottomEdge, word SrcXLeftEdge, word SrcYTopEdge, word
TransparentColorRed, word TransparentColorGreen, word
TransparentColorBlue, word TransparentMaskRed, word
TransparentMaskGreen, word TransparentMaskBlue)
```

The function returns 1 upon successful completion and 0 otherwise.

6.2.1.9 Area Fill

The parameters used in the Area Fill function are defined in Table 6-18.

Storage Types	Parameter
word	Client ID
word	Pixel Data Attr.
word	RasterOp
word	X Left Edge
word	Y Top Edge
word	X Right Edge
word	Y Bottom Edge
word	Fill Color Red
word	Fill Color Green
word	Fill Color Blue
word	reserved

Table 6-18: Parameters used in the Area Fill function

The function is called as follows:

```
int AreaFill (word ClientID, word PixelDataAttr, word RasterOp,
word XLeftEdge, word YTopEdge, word XRightEdge, word
YBottomEdge, word FillColorRed, word FillColorGreen, word
FillColorBlue, reserved)
```

The function returns 1 upon successful completion and 0 otherwise.

6.2.1.10 Pattern Fill

The parameters used in the Area Fill function are defined in Table 6-19.

Storage Types	Parameter
word	Client ID
word	Pixel Data Attr.
word	RasterOp
word	X Left Edge
word	Y Top Edge
word	X Right Edge
word	Y Bottom Edge
word	Pattern Width
word	Pattern Height
word	Pattern Offset X
word	Pattern Offset Y
word	Video Data Format
word	reserved

Table 6-19: Parameters used in the Pattern Fill function

The function is called as follows:

```
int PatternFill (word ClientID, word PixelDataAttr, word
RasterOp, word XLeftEdge, word YTopEdge, word XRightEdge, word
YBottomEdge, word PatternWidth, word PatternHeight, word
PatternOffsetX, word PatternOffsetY, word VideoDataFormat, word
reserved)
```

The function returns 1 upon successful completion and 0 otherwise.

6.2.1.11 Transparency Map Packet

The parameters used in the Transparency Map function are defined in Table 6-20.

Storage Types	Parameter
word	Client ID
word	reserved
word	X Left Edge
word	Y Top Edge
word	X Right Edge

Storage Types	Parameter
word	Y Bottom Edge
word	Transp Map Depth

Table 6-20: Parameters used in the Transparency Map function

The function is called as follows:

```
int TransparencyMap (word ClientID, word reserved, word
XLeftEdge, word YTopEdge, word XRightEdge, word YBottomEdge,
word TranspMapDepth)
```

The function returns 1 upon successful completion and 0 otherwise.