VESA®

Video Electronics Standards Association

2150 North First Street, Suite 440
San Jose CA 95131-2029

Phone: (408) 435-0333
FAX: (408) 435-8225

VESA VM-Channel (VMC) Software Interface Standard Baseline Implementation

Version: 1.0
Ratification Date: September 23, 1994

Purpose

To standardize an open software interface for transferring pixel data between two or more devices on the VM-Channel. This document describes the baseline operation that allows interworking of VMChannel based products from different manufacturers.

Table of contents

Intellectual Property

Trademarks

All trademarks used in this document are property of their respective owners.
VESA, VMC Video Electronics Standards Association

Patents

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

Support for this Specification

If you have a product which incorporates VMChannel, you should ask the company that manufactured your product for assistance. If you are a display or controller manufacturer, VESA can assist you with any clarification you may require. All questions must be in writing to VESA, in order of preference

America On-line: KEYWORD: VESA. Post your message on the VESA Message Board.
FAX: 408-435-8225
US Mail

VMC Software Work group Members

Any industry standard requires input from many sources. The people listed below were members of the VMC Software Work group of the VESA Advanced Video Interface Committee which was responsible for combining all the industry input into this proposal:

Ed Callaway - ATI Technologies
Gabe Varga - ATI Technologies
Mike Eskin - Brooktree Corporation
Daniel Daum - Cirrus Logic
Rod Dewell - Excalibur Solutions
Andy Potemski - IBM Corporation
Larry Henson - IBM Corporation
Frank Schwartz - MainStream (Chairman)
Onis Cogburn - Motorola
Walter Washington - Motorola
Ivy Lui - National
Chai Vaidya - National
Rod Williams - National
Mike Yonker - Pixel Semiconductor
Kent Dahlgren - Sierra
Don Pannell - Sierra
Jim Jeffers - Tseng Labs.
Greg Moore - VESA
Richard Hudson - VideoLogic (Document Controller)
Jud Lehmann - VLSI
Scott Hung - Weitek

Related Documents

VESA Media Channel Standard 1.0
Microsoft Windows 3.1 SDK.
VESA Media Channel Hardware Design Requirements and Guidelines.
VESA Media Channel Software Application Note.
Display Control Interface (DCI) Specification.

Revision History

1.01 - original proposal.
1.03 - 'Statement of Direction' document merged back into Baseline proposal.
Added Device control functions for scaling, clipping of data across a VMC stream.
1.04 - Message manager definition added in Appendix D.
Added message system overview diagram in section 4.
Extended message flow diagrams to incorporate message manager definitions in section 4.
Highlighted OS-specific calls in message flow diagrams
Added OS interface diagram in section 5
Re-partitioned device registration to reside within message manager.
Added majority of message definitions to Stream Manager, device drivers and message manager
1.05 - Clarified device type definition
Clarified multiple device context support for device drivers.
Clarified synchronization/exception reporting
Clarified VMC controller initialization
Replaced Message Manager with resource manager
Clarified device types and their capabilities
clarified message packet definition
defined general purpose message interface to Device Type Drivers.
1.06 - section 3 - Updated property list to include type and value.
- Clarified GTR allocation
- Clarified device/stream control overview diagrams.
Section 4 - Numerous error corrections in VMC API
- added MSG_OBJ, DEVICE_OBJ and STREAM_OBJ definitions to API.
Section 5 - clarified OS specific message interface diagram
- added Windows utility functions
- clarified .INI file settings for Windows.
- Added Resource Manager configuration
Removed VMC_TYPES.H (was APPENDIX A)
Cleaned up Stream Manager message Description (NOW APPENDIX A)
Cleaned up VMC Type Driver Message Description (NOW APPENDIX B)
Cleaned up Resource Manager Interface Description (NOW APPENDIX C)
Numerous changes to VMC Structure Definitions
1.07 Scope of Proposal changed to reflect initial implementation requirements for multistream transactions between graphic subsystem and video subsystems.
added glossary of terms
Section 2 - updated scope added to introduction, restricting initial scope of VMC software proposal to Windows 3.1 environment
Section 3 - new System architecture added.
Section 4 - new Operational Overview to describe video stream configuration- replaces previous section 3.
Section 5 - example code, replacing previous section 4. VESA supplied APIs removed.
Appendix A - updated to include message flow.
Appendix C Resource Manger deleted - Now GTR allocation description
Appendix D - New- discussion on Scaling Negotiation over VMC.
Appendix E - VMC structure definitions reworked.

Appendix F - VMC message parameter definitions added

1.08 Section 3 - use of Scaling and clipping capabilities clarified.
- numbers removed from flow diagram boxes.
Section 4 - CODEC contexts removed.
Section 5 - Detail added on Installable driver interface.
Appendix F - dwSize member added to all structures., SET_MASK_DATA_PARAMS structure added, GET_BANDWITDHINFO_PARAMS removed.

1.09
DCI and clipping definitions added to glossary.
Terminology change - Masking now referred to as clipping
New 'How To Use' Section Added.
Old Section 3 - Clarification on use of System Loader with 3rd Party Shells.
- Initialization sequence now checks Version.
Old Section 4 - Dest Image rectangle definition corrected.
Old Section 5 - Section renamed VMC Client Operational Overview for clarity.
- corrected DRV_VSM_QUERY_STREAM_CONFIGURATION example call.
- Amended Clipping example to use DCI structure.
- Stream configuration now includes means to set scaler mode at source and destination.
- various typos corrected.
Appendix A - DRV_VSM_ATTACH_STREAM_DEVICE message flow diagram amended.
- DRV_VMC_QUERY_VERSION message amended to return predefined version.
- Stream Manager state diagram added for clarification.
- various typos corrected.
Appendix B - typos corrected.
Appendix C - Bus Bandwidth definition clarified.
Appendix E. - VMC_DEVICECAPS structure definition corrected.
- VMC_RECTINFO structure definition corrected.
- dwSourceMode and dwDestMode members added to VIDEO_STREAM_CONFIG. wScalePriority removed.
- re-worded Context scaling capabilities
- added RGNDATAHEADER to VMC_CLIPINFO structure.
Appendix F. - various typos corrected

1.10
Section 2 - Video subsystem responsibilities clarified.
section 4 - context, scaling and clipping capabilities clarified.
section 5 - Device capability hierarchy added and capabilities outlined.
section 6. - code example updated to reflected updated structures.
Appendix A - standalone device access messages added.
- VSM_CONFIGURE_STREAM updated to cater for new scaling validated process.
Appendix B - DRV_ENABLE added for driver initialisation.
- DRV_VMC QUERY_SCALER_CAPS added.
- DRV_VMC_QUERY_STREAM_CAPS added.
- DRV_VMC_QUERY_CONTEXT_BANDWIDTH added
- DRV_VMC_VALIDATE_VERTICAL_SCALING added.
-DRV_VMC_VALIDATE_HORIZONTAL_SCALING added.
- DRV_VMC_QUERY_NUM_SCALE_BANDS added.
Appendix D - replaced.
Appendix E - numerous updates
Appendix F - numerous structure updates.

1.11

Section 6 - "vesa.vmc" changed to "vmc.StreamManager" in code example

AppendixA	- various error return clarifications - DRV_VSM_CONFIGURE_STREAM - order of QUERY_CONTEXT bandwidth and STREAM_PREAPARE messages swapped.
Appendix B	- various error return clarifications.
Appendix D	- equation for number of vertical scale bands corrected.
Appendix E	- VIDEO_IN changed to VIDEO_SOURCE. VIDEO_OUT changed to VIDEO_SINK. VIDEO_SINK context types added. VIDEO_SOURCE contexts renamed. - some #defines declared LONG - error codes extended.

1. Glossary of Terms

Device Type Driver	- software modules developed by vendors which implement support for the VMC Stream management message set, device specific message sets and vendor specific messages for the VMC hardware devices it controls.
Capabilities	- allow VMC clients to determine the functionality provided by a device and decide how devices should be connected to either end of a VMC stream.
Clipping	- the hiding of regions of a video window, resulting from other windows partially or totally obscuring that video window.
DCI	- The Display Control Interface defined by Intel and Microsoft providing access to graphic display devices. As far as VMC is concerned, DCI provides a standard means for obtaining clipping information.
VMC Device Context	- represents a set of device control parameters used to manipulate and route data as it passes through a VMC device and onto VMC, or vice-versa. A VMC device may support more than one context (eg. Video-in and CODEC playback). Each device context is associated with a unique VMC stream. It is the responsibility of the device itself to multiplex multiple streams onto the VMC.
Grant Time Register (GTR)	- controls maximum length of time for which a specific device can burst a continuous stream of data onto VMC.
Host Addressable Devices	- VMC devices controlled via the system bus or some other similar mechanism not involving VMChannel based control.
MCI	- Microsoft's multimedia control interface for Windows. MCI Overlay or DigitalVideo drivers are examples of VideoSubsystem drivers requiring access to VMC.
Parameters	- define the variables associated with a device or stream.
Stand-alone Devices	- devices that can only be accessed over VMC. Device registers must be accessed through using the non-stream read and write commands provided by all VMC controllers. Software drivers use the stream manager to send read and write commands to these devices.
(VESA) Stream Manager	- a software module, supplied by a vendor implementing a VMC bus controller. All Stream Managers support the set of Stream Manager Control defined by VESA. Stream Managers are expected to know how to communicate with the VMC Controller for the purposes of system initialisation and stand-alone device access. The means of accessing VMC Controllers is considered proprietary and does not form part of this document.

VMC(or VM-Channel or VMChannel) - a dedicated multimedia multidrop highway, optimized to support input and output of digital video streams directly into and out of the graphics system frame buffer.

VMC Clients - are software modules implemented in video subsystems which allocate and configure VMC streams between a video subsystem and a graphics subsystem via messages supported by the Stream Manager. In addition, VMC Clients use generic message sets to control non-VMC aspects of the graphics subsystem.

VMC Controller - a device which exerts global control over the VMC itself. This proposal only considers a minimum specification controller to be present in the system.

2. Introduction and Scope

The VESA Media Channel is a dedicated multimedia channel, optimized to support digital video streams in a single frame buffer environment. The VMChannel is a multidrop highway, enabling multiple devices to be combined in a modular fashion, thus allowing flexibility in configuration of a VMChannel based system.

Given the initial requirements for VMC, this proposal outlines a software interface that enables the setup and control of multiple stream communication between video subsystems and a graphic subsystem, operating in the Microsoft Windows environment.

Today, the graphics subsystem is a well-defined and integral component of a computer system. The video subsystem is harder to quantify. It is usually an upgrade to an existing computer system and may take various shapes: compressors/decompressors, video teleconferencing subsystems, video capture or playback boards, video inlay, etc. With these differing subsystems come different types of software drivers and means of controlling them. In all cases, however, the graphics subsystem remains a constant and as such is considered to present a generic interface to those VMC video subsystems wishing to communicate with it.

Consequently, the fundamental assumption in the baseline implementation is that the responsibility for managing VMC stream transaction lies with the video subsystem, the variable system component.

Each Video Subsystem wishing to communicate with the Graphics Subsystem must do so via its Video Subsystem software driver . This software driver, known as the VMC Client is responsible for setting up and configuring VMC streams.

As a VMC Client is an integral part of the Video Subsystem it knows how to control the components that make up its subsystem and it may expect the capabilities of its Video Subsystem to remain constant. However, VMC Clients, cannot guarantee that the Graphic Subsystem they communicate with will remain constant. An end-user should be able to replace one VMC enabled graphics card with another and expect the VMC system to operate as expected. Likewise, an end-user should be able to insert a Video subsystem into a VMC enabled graphics system and expect the VMC system to operate.

Therefore, in order that one vendors graphic subsystem can work with another vendors video subsystem, a VMC Client must always interrogate the capabilities of the devices it is communicating with and then configure streams based on those capabilities.

In light of these requirements, this document defines those components that enable video subsystems to communicate with a graphic subsystem and the means of using these components.

3. How To Use This Document

This document is intended for developers of VMC Graphic and Video subsystems.

Graphic Subsystem developers are required to provide three main VMC software components, these being:-

- a Windows display driver, incorporating a DCI Display Module and a software Device Type Driver to control the transfer of video data between VMC and the graphics display memory.
- a Stream Manager, to co-ordinate use of VMC.
- a VMC System Loader, to load the Stream Manager on Windows initialization.

Video Subsystem developers are required to provide two software components, these being:-

- a software Device Type Driver, to control the transfers of video data between VMC and the Video Subsystem.
- a VMC Client, a software module that uses VMC Device Type Drivers and the Stream Manager to transfer data between a Video Subsystem and a Graphics Subsystem. The form of the VMC Client is decided by the Video Subsystem developer. As an example, the VMC Client for a Video-In subsystem could take the form of an MCI Overlay driver.

Details of these various VMC system components can be located as follows:-

- Details of the VMC software architecture are described in section 4. This section describes how the various VMC components outlined here interconnect. This section also addresses system issues such as VMC initialization and software installation.
- Section 5 describes how a VMC Client uses Device Type Drivers and the Stream Manager. In particular, it describes how a client establishes the capabilities of devices at either end of a VMC stream and uses these capabilities to configure the clipping, scaling and format characteristics of the stream.
- Section 6 provides code examples to illustrate how a VMC Client uses the VMC software architecture.
- Appendix A defines the message interface that must be supported by developers of Stream Managers.
- Appendix B defines the message interface that must be supported by developers of Device Type Drivers.
- Appendix C defines how developers of Stream Managers should calculate the bandwidth allocated to a device transmitting across VMC.
- Appendix D defines how developers of Stream Managers should verify and negotiate scaling across VMC in situations where bandwidth is limited. This section will also be of interest to VMC Client developers in helping them to understand bandwidth limitation issues.
- Appendix E defines the VMC data structures.
- Appendix F defines the format of the messages used by the Stream Manager and Device Type Drivers.

4. VM-Channel System Software Architecture

The initial goals of the proposed architecture encompass the following:

- integration into Windows 3.1 and higher (using the installable driver message interface).
- multiple video stream transactions between Video subsystems and the graphics subsystem.
- interworking of (non-generic) vendor Video Subsystems with a generic graphic subsystem.
- agreed means of obtaining video clipping data from graphics subsystem.
- support for host addressable and Stand-alone devices.
- hide, as far as possible, access to VM-Channel specific control registers (through the use of a Stream Manager and Device Type Drivers).
- flexibility in the configuration of devices on the VM-Channel through support of user definable configuration messages.
- integration of new devices and stream types as they become available

The proposed architecture which addresses these goals is illustrated in figure one.

The operation of the VMChannel is controlled by VMC clients. These clients are responsible for two aspects of the VMChannel operation:

1. Allocation and configuration of a VMChannel stream between a graphics subsystem and a video subsystem. VMChannel stream control is achieved through the use of a Stream Manager

2. Control of non-VMChannel related aspects of the graphics and video subsystems. Device control is achieved through Device Type Drivers. Control of the Graphics subsystem must use generic video device type messages, although extended message interfaces can be used if known about.

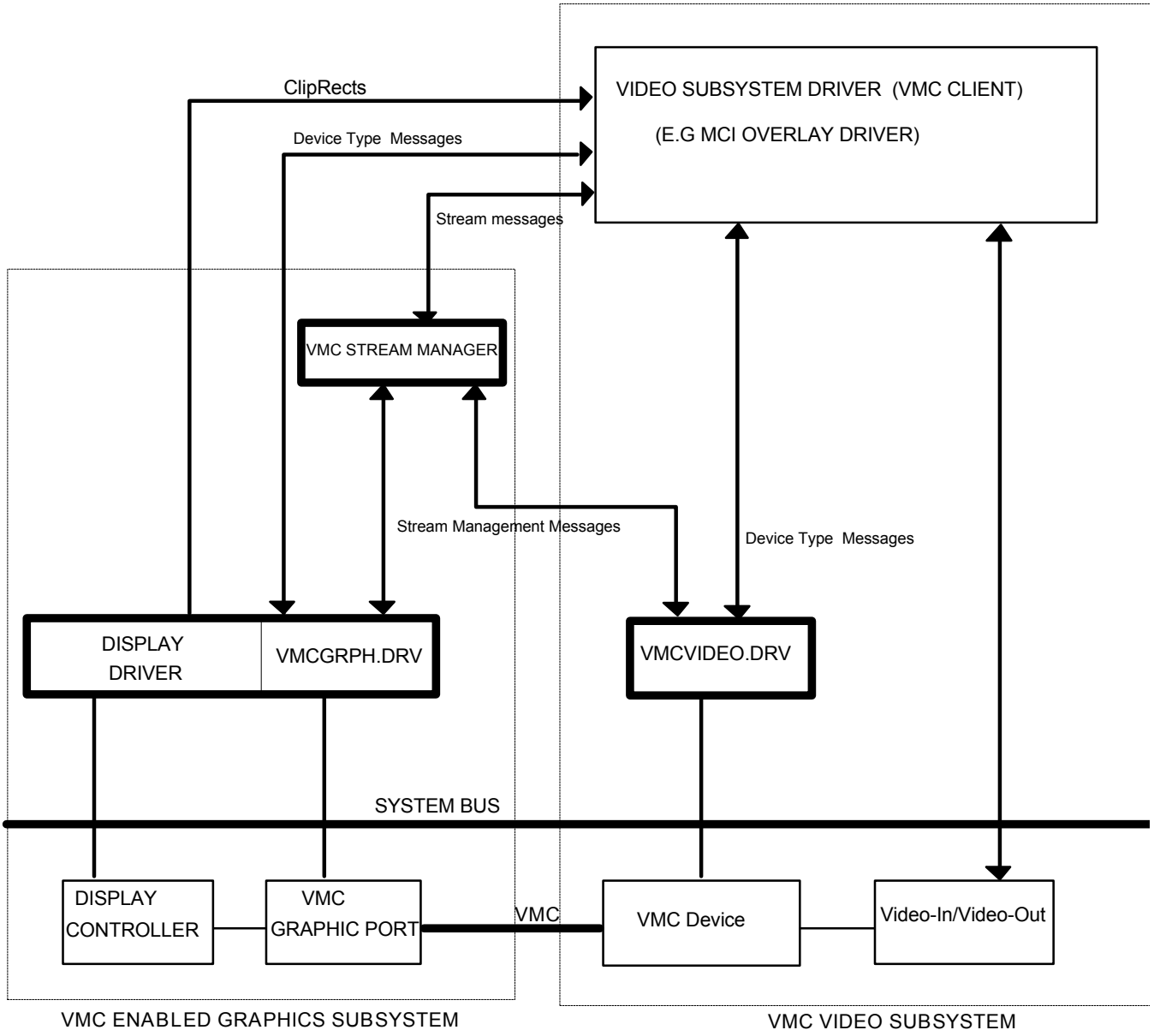


Figure 1 - VMC Software Architecture

4.1. VMC Software Components

4.1.1. VMC Device Type Drivers

Device control is modelled using the notion of VMC Device Type Drivers. VMC Device Type drivers are defined as those software drivers used to control VMC devices that source data onto or sink data from VMC.

Device Type Drivers respond to two separate message sets; those related to VMC stream control and those relating to device type specific control.

All Device Type drivers must implement a VMC stream management interface, used solely by the Stream Manager to manage VMC stream ID and bandwidth allocation.

In addition, Device Type Drivers belong to a given device type and may respond to a common set of VESA defined messages appropriate to that type and in accordance with its own capabilities. This provides an open interface that enables the integration of differing vendor products. In providing a usable and open interface, Device Type Drivers are a key concept in the VMChannel architecture.

In addition to implementing a generic set of messages, each device may also respond to vendor specific messages.

The architecture allows for different device types to be installed as and when they become available.

Device type drivers are developed by Vendors and/or their partners implementing the appropriate interface defined by VESA.

4.1.2. Stream Managers

VM-Channel Stream control is achieved through the notion of streams. VM-Channel streams are managed by a Stream Manager under control of clients.

Stream Managers are supplied and installed by vendors implementing VM-Channel controllers as an integral part of their VMC product. Initial VMC implementations will be in the form of VMC enabled graphic subsystems.

The interface to and the operation of these Stream Managers are defined by VESA.

The architecture allows for different Vendor Stream Managers to be installed in a given system. However, only one instance of this type of driver can ever be active at a given time.

The Stream Manager is the central co-ordinator for devices using streams on the Channel and is responsible for:

- maintaining a list of devices on the bus.
- allocating unique stream ID's
- determining and monitoring Channel bandwidth usage.
- stand-alone device register access

The Stream Manager uses the inherent round robin scheduling mechanism of the VMChannel hardware. The manager also allocates VMC bandwidth based on a fixed token rotation time and a fixed device time slices (i.e once allocated, a devices' bandwidth remains fixed for the duration of its processing activity or until the stream characteristics need to be changed eg. Window size changes).

4.1.3. System Loader

System Initialization is handled through use of a loader, ideally in the form of a window-less applet. This component loads the selected Stream Manager (see System installation) on Windows initialization and issues an enable command to the Stream Manager. The Loader remains resident until Windows shuts-down, where-upon it issues a Disable message to the Stream Manager, prior to exiting.

Only the Stream Manager is permanently resident. Device Type Drivers are loaded and unloaded on demand.

4.1.4. Software Driver Impleme ntations

Stream Managers and Device Type Drivers are implemented as Windows 3.1 installable drivers. By convention, drivers are assigned a .DRV extension and respond to the standard installable messages as defined by Microsoft.

In addition to the standard Windows installable driver messages, drivers must also respond to a number of VESA defined messages. These are detailed in the appendices.

A VMC Client communicates with the Stream manager using the standard Windows OpenDriver(), SendDriverMessage() and CloseDriver() calls.

All messages defined in this specification are synchronous and relate to the applications thread of execution.

Device Type Drivers and Stream Managers are required to implement version numbering, which can be queried by VMC Clients. Version Numbers returned correspond the VESA Software Specification Version and Revision number for which they have been implemented.

4.2. Baseline System Configurations

Two component types constitute the baseline VMC system, these being a Graphics Subsystem and one or more video subsystems.

A Graphics Subsystem provides the following VMC features:

- a minimum specification VMC hardware controller implemented as part of its VMC interface with support for at least one input stream.
- a VMC Stream Manager, supporting the message set defined in Appendix A. The Stream Manager, in implementing the message set in Appendix A, will need to communicate with the VMC hardware controller for the purposes of VMC initialization and stand-alone device access.
- a loader, to startup the Stream Manager on Windows initialization.
- a Device Type Driver that can be called upon by VMC clients to control VMC video stream access to and from the frame buffer. Device Type drivers for the graphic subsystem must implement the VMC Stream management messages, as defined in Appendix B.
- a Display driver (possibly supporting an extension to the current DCI specification) that enables VMC Clients to obtain clipping information.
- an installation procedure for installing and configuring the above components.

A VMC video subsystem provides the following features:

- a VMC device (host addressable or Stand-alone).
- a VMC Device Type Driver to enable control of VMC streams passing into or out of the video subsystem. A Video subsystem Device Type Driver must support the VMC Stream Management messages as defined in Appendix B. Video Device control is proprietary.
- a VMC video subsystem driver (VMC Client) to set-up and control video streams passing between the Graphics subsystem and the video subsystem.
- an installation procedure for installing and configuring the above components.

For a given set of product mixes, the system level functionality that can be supported by a given VMC configuration is governed by the following capability matrices.

VMC Clients are required to query the capabilities of the VMC devices attached to a stream to determine the functionality that can be offered to the application user.

Video Subsystem Context Capabilities

Context Type	Comments
VIDEO_OUT	If this type is defined, the Video subsystem supports at least one context that will take data off VMC for processing by the subsystem. Video subsystems may further define the context through a sub-type. The sub-type may be used, for example, for routing the VMC Stream to a CODEC, video-out device or FX processor.
VIDEO_IN	If this type is defined, the Video subsystem supports at least one context that will source data onto VMC from the subsystem. Video subsystems may further define this context through a sub-type. The sub-type may be used, for example, for routing data from a CODEC, video-decoder or FX processor onto VMC.

It is possible for Video Subsystems to support multiple context types, and thus allow multistream transactions. The number of streams supported by a subsystem is vendor specific.

Graphic Subsystem Context Capabilities

Context Type	Comments
GRAPHIC_SINK	Data sourced from VMC into graphics memory. The actual number of active contexts is Graphic Subsystem specific. The only context subtype currently defined is for routing data to on screen memory. Graphics subsystem must support at least one video window.

GRAPHIC_SOURCE	Data sourced from graphics memory for output to an external video device. This is an optional context that can be supported by the graphics subsystem. The only context subtype currently defined is for routing data from on-screen memory to VMC.

It is possible for Graphic Subsystems to support multiple context types, and thus allow multistream transactions. The number of streams supported by a subsystem is vendor specific.

System Scaling Capabilities

System scaling capabilities depend on the stream type and the scaler modes supported by the source and destination devices. Two stream types are defined, interlaced and non interlaced. For interlaced streams, two possible scaler modes are defined, interlaced and progressive scan. For non-interlaced stream only one scaler mode is defined, non-interlaced scaling.

Each scaling mode defines independent horizontal and vertical scaling quality, accuracy, and scale factor capabilities.

The following illustrate how capabilities should be used in either stream mode.

- Interlaced Video streams

With interlaced video streams, the source device passes both odd and even fields across VMC. The Graphics Subsystem is responsible for de-interlacing these fields onto the display.

Video Subsystem	Graphics Subsystem	Comment
cannot scale	cannot scale	Video Window size fixed to source size. The video subsystem must generate pixels of the correct aspect ratio. Graphics Subsystem de-interlaces the fields onto the display. In VMC bandwidth limitations the stream is disabled.

cannot scale	interlaced scaling (horizontal only)	Source device passes both fields across VMC. Video window height is fixed to the source image height. (i.e the Graphics subsystem does not scale vertically in this mode). Video window width is determined by the horizontal scaling capabilities of the Graphics Subsystem. Graphics Subsystem de-interlaces both fields onto the display. In VMC bandwidth limited situations, the stream is disabled.
Interlaced scaling (horizontal only)	Progressive scan scaling (vertical only)	Source device passes both fields across VMC and does not scale vertically. Video Window height determined by the vertical scale capabilities of the Graphics Subsystem. Video Window width determined by the horizontal capabilities of the Video Subsystem. Graphics Subsystem de-interlaces both fields onto the display. In VMC bandwidth limited situations, a size smaller than the requested window size is produced.
interlaced scaling (horizontally & vertically)	cannot scale	Video Window height determined by the vertical scale capabilities of the Video Subsystem. Video Window width determined by the horizontal capabilities of the Video Subsystem. Graphics Subsystem de-interlaces both fields onto the display. In VMC bandwidth limited situations, the source scales down to a size smaller than the requested window size.
interlaced scaling (horizontal & vertical)	interlaced scaling (Horizontal only)	Video window height determined by the vertical scale capabilities of the Video Subsystem. Video window width determined by the horizontal capabilities of both the Video Subsystem and the Graphics subsystem. Graphics Subsystem de-interlaces both fields onto the display. In VMC bandwidth limited situations, the source device may scale down horizontally and the destination scale up horizontally. Alternatively, a smaller image size may result.

cannot scale	progressive scan scaling (vertical & horizontal)	Video window height determined by the vertical scale capabilities of the Graphics Subsystem. Video window width determined by the horizontal capabilities of both the Graphics subsystem. Graphics Subsystem de-interlaces both fields onto the display. In VMC bandwidth limited situations, the stream is disabled

Subsystems must implement scalers that either scale arbitrarily or perform integer scaling.

- NON-Interlaced Video Streams

With non-interlaced video streams the source device is responsible for transmitting an odd or an even field across VMC. There are no de-interlacing issues here as far as the Graphics Subsystem is concerned so scaling is less complex.

Video Subsystem	Graphics Subsystem	Comment
cannot scale	cannot scale	video window size fixed to source size. In the case of VMC bandwidth limitations, it will not be possible to transmit across VMC.
cannot scale	non-interlaced scaling (vertical & horizontal)	Video window height limited by the vertical scaling capabilities of the Graphics Subsystem. Video window width determined by the horizontal capabilities of the Graphics Subsystem. In VMC bandwidth limited situations, the stream is disabled.
non-interlaced scaling (vertical & horizontal)	cannot scale	Video window height determined by the vertical scale capabilities of the Video Subsystem. Video window width determined by the horizontal capabilities of the Video Subsystem. In VMC bandwidth limited situations, the source device will have to scale down to a size smaller than the requested destination size.

non-interlaced scaling (vertical & horizontal)	non-interlaced scaling (vertical & horizontal)	Video window height determined by the vertical scale capabilities of the Video and Graphics Subsystems. Video window width determined by the horizontal scale capabilities of the Video and Graphics subsystem. In VMC bandwidth limited situations, the source device will have to scale down to a size smaller than the requested destination size and the destination attempt to scale the image up the required size.

Subsystems must implement scalers that either scale arbitrarily or perform integer scaling.

Clipping Capabilities

VMC Clients obtain clipping information from the DCI driver provided as part of the graphics subsystem. The exact means of achieving this are defined in the Application Note.

Video Subsystem	Graphics Subsystem	Comment
cannot clip	cannot clip	Video cannot be obscured. For video display, overlaying graphics data will get corrupted. Therefore, VMC Clients must detect that their Video Window is about to be obscured and stop video.
cannot clip	can clip	clip data can only be generated at receiver.
can clip	cannot clip	video clipping performed by transmitter. Transmitter uses VMC hardware mask signals to inform receiver of data to be masked out. Clipping needs to take into account that scaling may be active in receiver. However, clipping at source will not be pixel accurate if the the destination device is scaling and the overlapping window is not aligned on a scale factor boundary.
can clip	can clip	clipping probably determined by clipping granularity of devices. No effect on VMC bandwidth.

4.3. System Installation

Stream Manager

A Stream manager is installed when the user installs the graphics drivers for his board or when the computer manufacturer pre-loads the system software. This driver is copied to

the fixed disk drive (probably WINDOWS\SYSTEM) and SYSTEM.INI and CONTROL.INI are updated.

The selected stream Managers is defined through a setting in the drivers section of SYSTEM.INI, the entry format being:

[drivers]
vmc.StreamManager="drive:\fullpath\mydriver.drv"

The stream Managers description text, which appears in the list box in the Drivers Control Panel is defined through a setting in the drivers.desc section of CONTROL.INI, the entry format being:

[drivers.desc]
"drive:\fullpath\mydriver.drv"= MY Stream Manager Description Text

VMC System Loader

A system loader is installed along with the Stream Manager, and is copied to the fixed disk drive (probably WINDOWS\SYSTEM) and WIN.INI is updated.

The Loader is activated on Windows startup and must be defined through a setting in the windows section of WIN.INI, the entry format being:

[windows]
Load="drive:\fullpath\myloader.exe"

Note:- this definition is not guaranteed to work with third party shells. Details for installing the system loader with third party shells are detailed in the Software Application Note.

Device Type Drivers

The baseline implementation only supports VMC devices directly addressable over the system bus. A vendor specific installation procedure identifies which subsystems the Device Type Driver is expected to support and the I/O address and IRQ level for that product. Device Type Drivers are installed when the user installs the graphics drivers/Video drivers for his board or when the computer manufacturer pre-loads the system software. This driver is copied to the fixed disk drive (probably WINDOWS\SYSTEM) and SYSTEM.INI is updated.

All Device Type Drivers are required to make an entry in the [VMC.TypeDrivers] section of SYSTEM.INI.

There is only one selected Graphic Subsystem Device Type driver in a system and is assigned the key vmc.GraphicPort.

Video Subsystem Drivers are assigned the key: vmc.VideoSubsystem#n, where n is from 1 to 16.

As there may well be a number of Video Subsystem Drivers, care must be taken in not duplicating the key names for the drivers and ensuring consecutive numbering. The first available device number is 1 and the last device number 16.

Any additional VXD's or dll's required by a driver must be specified as required by windows 3.1.

Example System.ini entry:-

```
[vmc.TypeDrivers]
vmc.GraphicPort="drive:\fullpath\mydriver.drv"
vmc.VideoSubsystem1="drive:\fullpath\mydriver.drv"
```

4.4. System Initialization

VMC system initialization is defined in figure 2:-

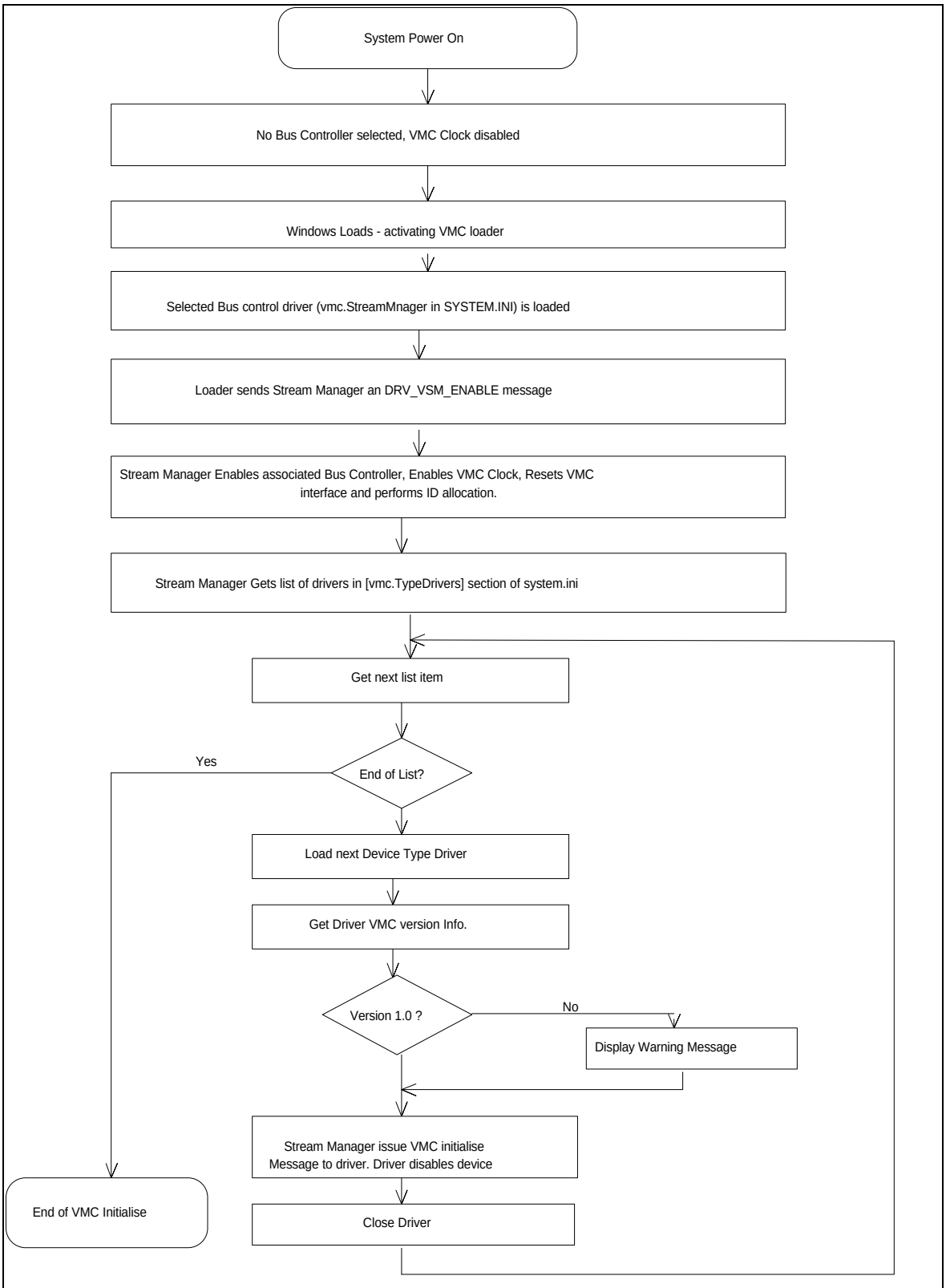


Figure 2. VMC System Initialization

5. About VMC Clients

This section outlines how VMC Clients use the Stream Manager and the video and graphic subsystem device type drivers to control VMC streams.

To use VMC, a Client must create device contexts in the video and graphics subsystem, allocate a VMC stream and associate these device contexts with the allocated stream. For a given application, VMC clients know in advance the specific types of device contexts they require to communicate across a VMC stream.

VMC Clients cannot guarantee that the Graphic Subsystem they communicate with will remain constant. An end-user should be able to replace one VMC enabled graphics card with another and expect the VMC system to operate as expected. Likewise, an end-user should be able to insert a Video subsystem into a VMC enabled graphics system and expect the VMC system to operate.

A VMC Client must always obtain the capabilities of the devices it is controlling to ensure that the required functionality can be provided, although in the baseline implementation, only those of the Graphic Subsystem can be expected to change without notice.

The hierarchy for determining VMC capabilities is illustrated in figure 3. This illustrates that a given device type driver can support many VMC device instances. In turn, each VMC device instance may support many context types (e.g CT_VIDEO_SOURCE and CT_VIDEO_SINK) and each context type may support many different video stream modes (eg VSM_INTERLACED_VIDEO and VSM_NON_INTERLACED_VIDEO_EVEN_FIELD). Further, a device may have different scaling algorithms that can be associated with a stream mode (e.g SM_INTERLACED).

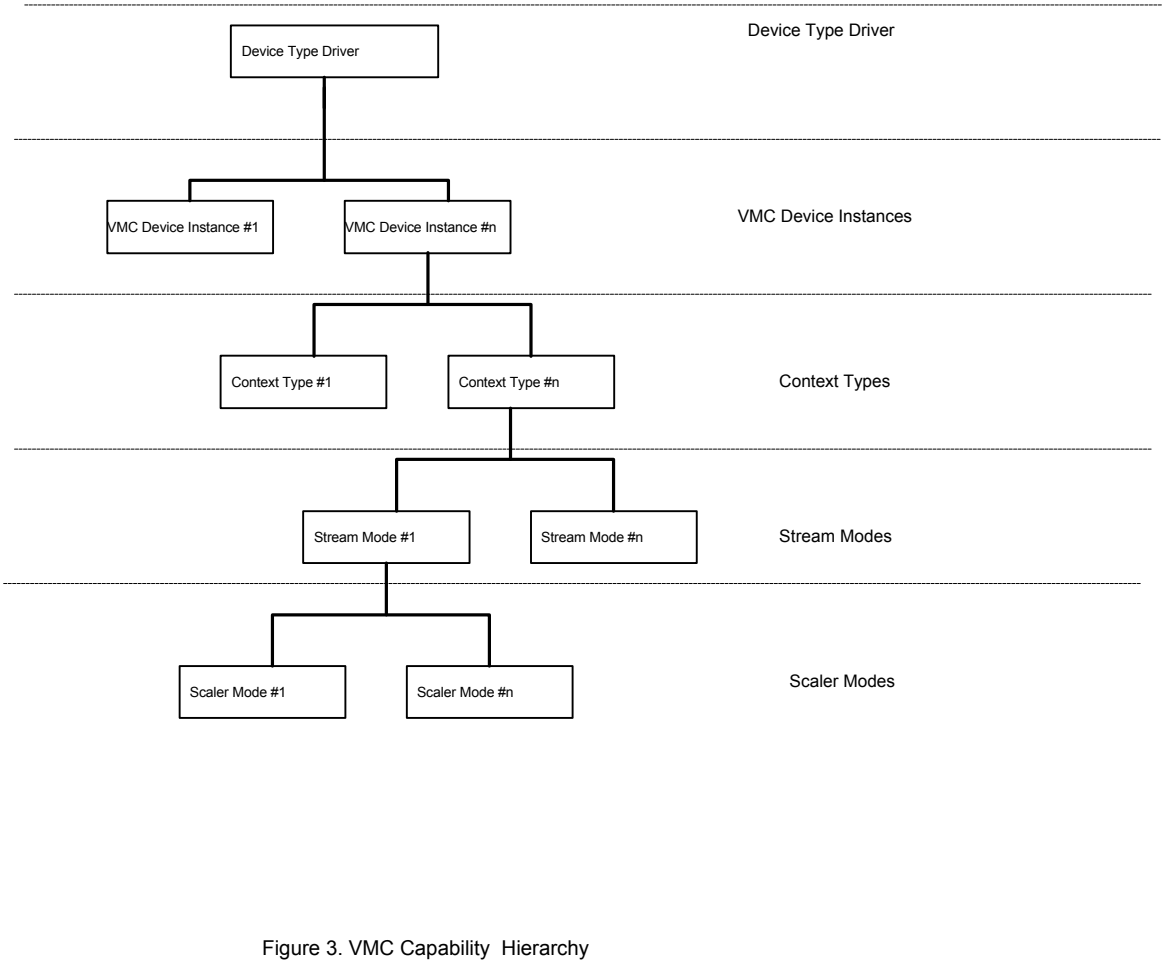


Figure 3. VMC Capability Hierarchy

A VMC Client needs to descend this hierarchy in order to obtain the required capabilities. The following section describes how this is achieved.

Once device contexts have been allocated and associated with a stream, the VMC Client must use proprietary messages to control the non-VMC related aspects of the Video Subsystem.

The Graphic Subsystem is controlled via the Stream Manager, through issuing stream configuration and clipping messages.

5.1. Using VMC Video Device Type Drivers

The stages of using a Device Type driver are as follows:

- Open the device type drivers, using the .INI entry keys in the [vmc.TypeDrivers] section of SYSTEM.INI.
- Query the driver to see how many VMC devices are controlled by the driver and get the Vendor information block for each device. The Vendor Information Block provides the following information for each VMC device instance it controls:
    - Vendor Name

- Product Name
- Board Information String (contains Vendor Specific Information)
- Board Information DWORD (Contains Vendor Specific Information)
- Board Instance ID (board ID associated with the device)
- Board revision (board revision ID associated with the device)
- VMC device ID
- VMC Device Type
- Supported Context Types (CT_VIDEO_SOURCE & CT_VIDEO_SINK for Video Subsystems. CT_GRAPHIC_SOURCE, CT_GRAPHIC_SINK for Graphic Subsystems).
- supported sub types (for the baseline implementation, these are only used by the Video Subsystem for routing VMC streams through a device where a VMC device can interface to more than one data source or sink).

- Examine the capabilities of the Vendor Information Block. The information retrieved enables a client to uniquely identify all VMC devices on all Video Subsystems controlled by the Type Driver.

  In order to allocate a device context, a VMC client first needs to identify the contexts and subtypes available for a given device.

  Not all devices will have the same contexts available.

  The following contexts are defined for a Graphic Subsystem:-
  - can sink stream from VMC into the frame buffer
  - can source Streams onto VMC from the frame buffer

  currently, Graphic Subsystems only support one subtype, writing to or from on-screen memory.

  The following context types are defines for a Video Subsystem:-
  - can source data from a video Source device onto VMC
  - can sink data from VMC to a video out device

- create device contexts for use with the required Video and Graphics subsystems. The VMC Client requests the Device Type Driver to open a context of the specified type and subtype for a specific VMC device instance. It is possible for a Device Type Driver to support many instances of the same context type. The driver will reject a request to create a context if there are no more available.

- Obtain the Context Specific Capabilities to decide which stream modes can be supported. The following context capabilities can be obtained for all context types:
  - supported stream modes (interlaced or non-interlaced)

- Obtain the Stream specific capabilities to decide where in a stream the desired functions should be performed. The following stream parameters can be obtained:
  - supported stream formats
  - clipping capabilities (none or rectangular clipping)
  - maximum number of clipping regions
  - Scaler mode capabilities - a scaler may support several scaling algorithms for a given stream type. A VMC client should determine the supported types and then get the specific scaler mode capabilities to determine the appropriate scaling options. The defined scaler modes are SM_NON_INTERLACED, SM_PROGRESSIVE_SCAN and SM_INTERLACED.

- Obtain the Scaler mode specific capabilities to decide on maximum image sizes: The following scaler mode parameters can be obtained:-
    - scaler quality(none, interpolation/filtering and replication/decimation)
    - stream type scaler accuracy (arbitrary or integer)
    - minimum and maximum scale factors
    - peak input pixel rate
    - peak output pixel rate
- associate the device contexts with a stream. When associating contexts, the Stream Manager needs to be informed which device is the stream transmitter and which, if any, of the devices are to perform clipping.
- verify stream configuration and scaling parameters with the Stream Manager. This must be performed as at this stage it has not been verified that there is sufficient bandwidth or that the video scalers can achieve the required scale factors or stream formats.
- configure the Video and Graphic Subsystems. Video Subsystem contexts are configured using proprietary messages. Graphic subsystem contexts are configured using the Stream Configuration Commands in order to configure video window size, position and clip regions.
- configure and enable the stream, using Stream Manager messages
- a device is enabled when VMC transfers are about to commence and disabled when they are about to stop. A device must be disabled every time the stream configuration is changed (i.e image size and format).

5.2. Using the Stream Manager

In order to use the Stream Manager, it must be opened using the vmc.StreamManager key in the [drivers] section of SYSTEM.INI. Once opened the process of allocating and configuring VMC streams can begin.

5.2.1. Stream ID Allocation

Devices connected to the VMChannel may support one or more transmission streams. Each stream on the VMChannel must have a unique ID that provides an identifiable transmission path between a transmitter and a receiver.

Streams are characterized by a specific type. The type of stream affects those characteristics that can be configured. For example, a video stream has format, type, size, data rate and scaling/clipping capabilities . Currently, the defined VMC stream types are:

- VST_VIDEO_STREAM

Stream ID's are allocated by the Stream manager when a client requests a new stream of a given type to be reserved. Clients must release the stream when no longer required. Failure to release a stream may prevent other clients using the VMC.

5.2.2. Stream Control

In order to configure a stream, a client must associate the required device contexts with the stream and then query the Stream Manager to obtain stream configuration options.

The model used for defining the control functionality between a video subsystem and a graphics subsystem is illustrated in figure 4.

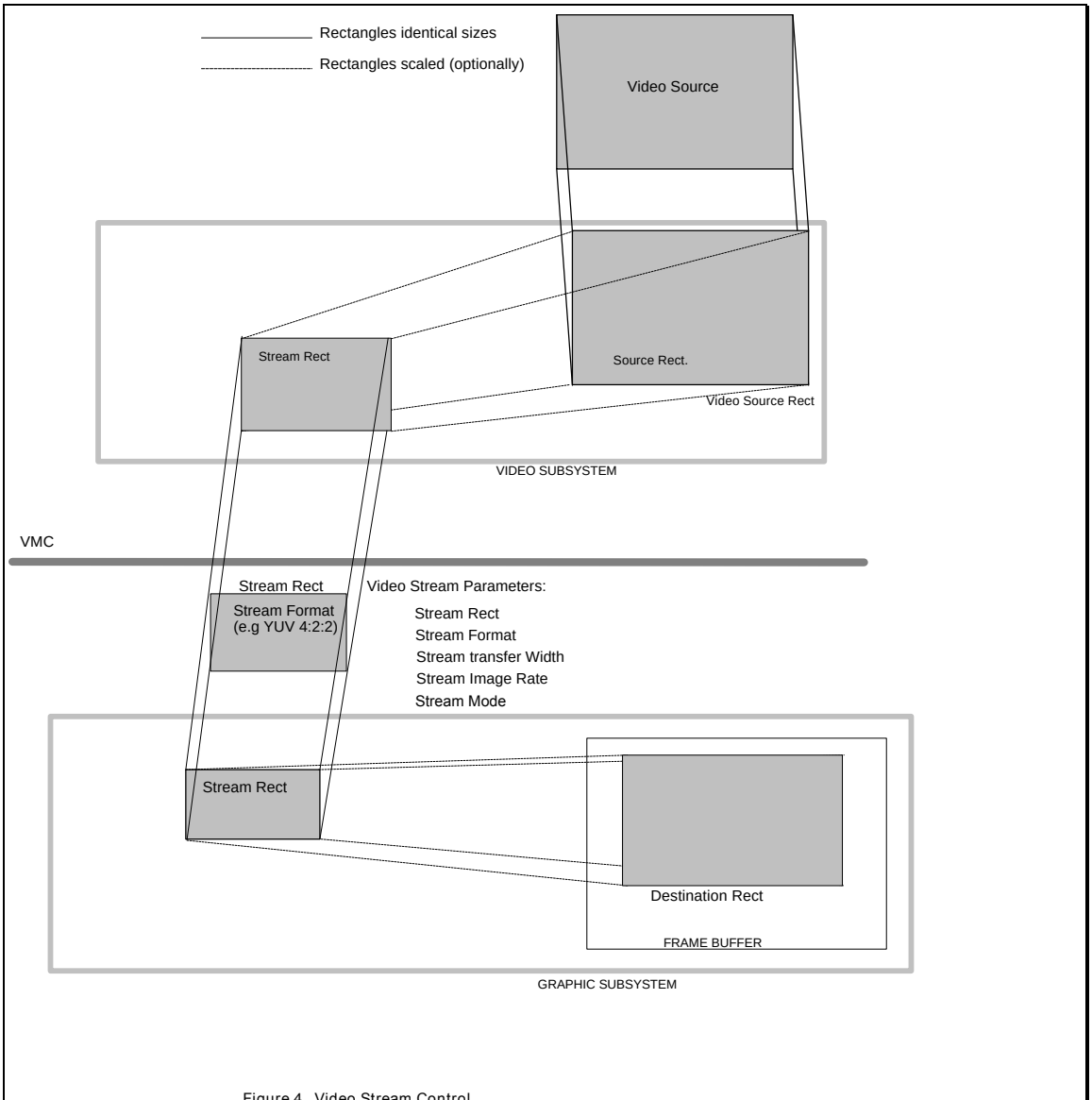


Figure 4. Video Stream Control

The model enables a VMC client to decide where in the transmission path scaling and clipping should be performed, depending upon the capabilities of the video subsystem and the graphics subsystem. However, the required scaling that can be achieved may be limited by bandwidth restrictions in the system. Appendix D, although intended for Stream Manager Vendors, discusses bandwidth restrictions for Video Streams.

Not all devices will have the same capabilities in terms of scaling and clipping. Therefore, source and destination capabilities of a device can be ascertained by VMC

Clients through querying the device context capabilities. VMC Clients use these capabilities to determine where in the stream certain functions should be performed.

Once the stream options have been determined, the VMC Client can begin the process of configuring a stream.

5.2.3. Stream Scaling

Stream scaling is defined by three rectangles, source Rect, Stream Rect and Dest Rect. The ratios of Stream Rect to Source Rect define the scaling performed in the transmitting device. The ratio of Dest Rect to Stream Rect defines the scaling performed in the receiver. The following defines how image rectangles are calculated:-

SourceWidth = SourceRect.iRight - SourceRect.iLeft.
SourceHeight = SourceRect.iBottom - SourceRect.iTop
StreamWidth = StreamRect.iRight - StreamRect.iLeft.
StreamHeight = StreamRect.iBottom - StreamRect.iTop
DestWidth = DestRect.iRight - DestRect.iLeft.
DestHeight = DestRect.iBottom - DestRect.iTop

Note:- The Dest Rect defines both position and size. The position information is in co-ordinates relevant to the destination output device. The Source Rect can imply both size and position. Specifying Non-zero position information in the Source Rect implies that the device can clip the incoming Video. Stream Rect does not contain position information.

Note:- Some scalers, due to rounding errors, may produce an extra pixel/line than that requested. Where possible, these rounding errors should be clipped at the ouput of the scaler. However, as a safeguard, destination devices should always try and clip a scaled stream to ensure that it is never greater than the requested destination rectangle.

The scaling that can be achieved by a device is calculated from the following scaler mode parameters:-

dwScaleCaps - defines horizontal and vertical qualities of the scaler mode.
wXScaleGranularity - (defines horizontal scaling accuracy)
wYScaleGranularity - (defines vertical scaling accuracy)
wXScaleFactorMax - defines maximum horizontal scale factor relative to the incoming image width.
wXScaleFactorMin - defines minimum horizontal scale factor relative to the incoming image width.
wYScaleFactorMax - defines maximum vertical scale factor relative to the incoming image height.
wYScaleFactorMin - defines minimum vertical scale factor relative to the incoming image height.

For a transmitting device capable of horizontal scaling the following must hold true:
(StreamWidth/SourceWidth) * 100 >= wXScaleFactorMin

and

(StreamWidth/SourceWidth) * 100 <= wXScaleFactorMax

For a transmitting device capable of vertical scaling, the following must hold true

(StreamHeight/SourceHeight) * 100 >= wYScaleFactorMin

and

(StreamHeight/SourceHeight) * 100 <= wYScaleFactorMax

For a receiving device capable of horizontal scaling the following must hold true:
(DestWidth/StreamWidth) * 100 >= wXScaleFactorMin

and

(DestWidth/StreamWidth) * 100 <= wXScaleFactorMax

For a receiving device capable of vertical scaling, the following must hold true
(DestHeight/StreamHeight) * 100 >= wYScaleFactorMin

and

(DestHeight/StreamHeight) * 100 <= wYScaleFactorMax

It is the responsibility of the VMC Client to ensure that the scale factors are supported by devices within the stream.

For a given stream colour format and stream mode, the VMC Client should request the Stream Manager to determine if a specified Destination size can be achieved in light of current VMC bandwidth availability, image source size, source and destination scaling capabilities. The Stream Manager will ascertain if scaling can be achieved as requested. If not, it will return to the VMC Client details describing the maximum size of the destination window that can be achieved, given current stream and device capability restrictions. The Stream Manager will use scaling capabilities at either end of the stream in order to achieve the required destination size.

It may be that due to limitations of a device or insufficient bandwidth that the required destination size cannot be realized. In this case, a VMC Client is responsible for changing stream parameters in order to try and achieve the required Destination size or accept a reduced destination size.

5.2.4. Stream Clipping

Clipping is used to define the area of active video within a display region. Some devices may only perform rectangular clipping while others may be capable of more complex shape clipping. Clipping can be applied either by transmitting or receiving devices although its is most commonly used by receivers in the graphics domain.

When clipping is performed at source, then the clipping applied must be related to any scaling being performed in the graphics domain. For example, if the destination is scaling up twice vertically, the source must halve the vertical extents of the clip information.

Note:- The drawback to clipping at source is that if the destination is being used to scale, then non-pixel accurate clipping will result if the position of the overlapping window is not on a multiple of the destination scale factor.

Clip data is obtained by a VMC Client through defining a call-back to the Display Driver Module. In order to do this, the VMC Client must have first registered a video surface with the Display Driver Module. Once setup, The VMC client gets informed every time the clipping rectangles for the Video Window change. The VMC Client needs to apply

these rectangles to the stream. This is achieved through the Stream Manager, which sends the supplied clip data to the device assigned to handle clipping for the specified stream. The stream Manager sends to the clip device the current scaling factors at source and destination.

In order to simplify clipping implementation, data is always supplied in the form of clipping rectangles. More complex clipping must be implemented as a proprietary mechanism.

VMC Clients should check the maximum number of clipping rectangles supported by the clipping device to ensure that graphics data does not become over-written with video.

5.2.5. Stream Configuration

When the best configuration has been determined, a VMC client configures its Video Subsystem and then requests the Stream Manager to set the configuration for the specified stream (either using the scaling factors suggested by the Stream Manager or using its own calculations). Once devices and streams have been configured, the bandwidth requirements for that stream are known and fixed and the VMC bandwidth can be allocated and the stream enabled.

Enabling a stream allows data to pass from the source device, over VMC and through the destination device.

Once a stream has been enabled, a stream is considered free running. If changes to the stream characteristics are required then the stream must be disabled so that bandwidth can be reassessed. Once disabled, device and stream characteristics can be re-configured prior to the stream being re-enabled.

6. VMC Client Operational Overview

The operation of the VMC by a client is summarized as illustrated in figure 4:

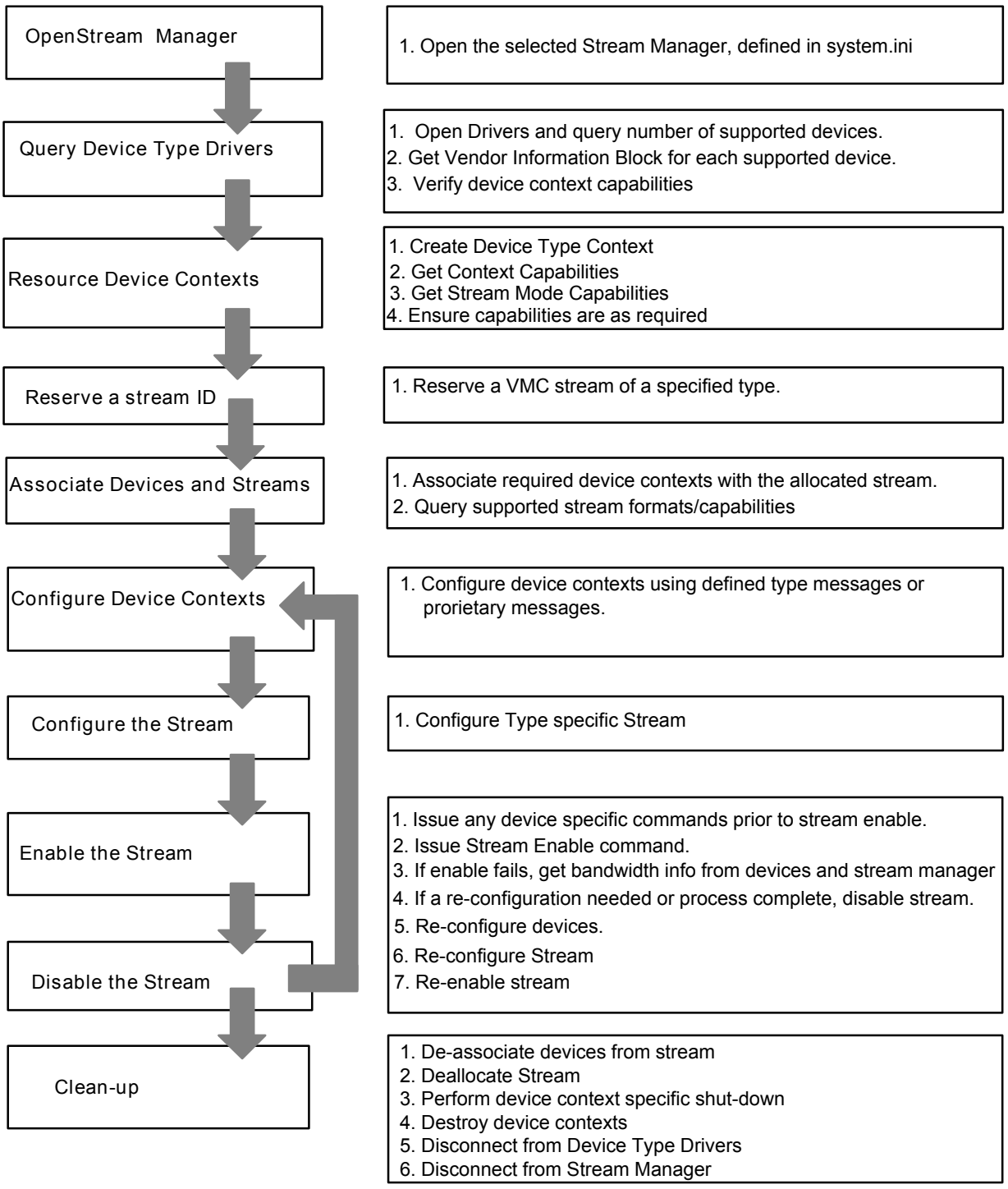


Figure 4: VMC Client Control Overview

VMC Clients communicate with the Stream Manager and Device Type Drivers through the Windows installable driver interface. The installable driver interface is message based. A range of messages from DRV_RESERVED (defined in WINDOWS.H) to DRV_USER (also defined in WINDOWS.H) are available for usage by an installable driver. Each message has specific input and output parameters outlined in Appendix A, Appendix B, Append F and VESA_VMC.H (supplied with the VMC development kit).

Video and Graphic Subsystem Vendors can extended the range of messages supported by their Device Type Drivers, but these will be proprietary and should be defined in vendor specific header files.

The VMC client uses the Windows SendDriverMessage() function to send messages to the Stream Manager and Video and Graphic subsystems and is in the form:-

```
(vmcBOOLEAN)SendDriverMessage( (HDRVR)hDriver,
                               (UINT)DRV_MESSAGE_NAME,
                               (LPARAM)lpMessageParams,
                               (LPARAM)lpVMCErrorParams);
```

This function will only return non-zero (TRUE) and zero (FALSE). These returns should only be used to validate Driver messages defined by Microsoft.

Stream Managers and Device Type drivers return TRUE for the SendDriverMessage function, even if they detect an error. VMC defined messages should be validated through checking the error return structure (VMC_ERROR_PARAMS), returned for each message.

When configuring message buffers, a VMC Client must ensure that the dwSize member is set. Calls may fail if the size of a structure passed into a device type driver or the Stream Manager does not match that for the implementation version.

As an illustration of VMC Client control, the following example demonstrates how to setup and configure a VIdeo-In device and Graphics subsystem.

All the listed code is within the VMC Client. While calls to Device Type Drivers and the Stream Manager are made through the SendDriverMessage() function, no code examples relating to the processing of Stream Manager or Device Type Driver messages are given. These are supplied with the VMC developers kit.

- Open the Stream Manager
  The Stream Manager must be opened to gain access to information about the VM-Channel. The ID returned as result of the OPEN message is used as a parameter to all future communication with the Stream Manager to ensure that messages are routed correctly.

```
/* Definitions for the VMC Client Example */
VMC_ERROR_PARAMS                VMCErrorParams;
DEVICE_OBJ                      GraphicObj;
CREATE_CONTEXT_PARAMS           VideoContextParams;
CREATE_CONTEXT_PARAMS           GraphicContextParams;
DEVICE_OBJ                      DeviceObj;
QUERY_NUM_VMC_DEVICES_PARAMS    QueryNumDevicesParams;
QUERY_VENDOR_INFO_PARAMS        QueryVendorParams;
QUERY_CONTEXT_CAPS_PARAMS       QueryTypeParams;
QUERY_STREAM_CAPS_PARAMS        QueryStreamParams;
QUERY_SCALER_CAPS_PARAMS        QueryScalerParams;
QUERY_STREAM_CONFIG_PARAMS      QueryConfigParams;
```

```
CREATE_CONTEXT_PARAMS           GraphicsContextParams;
ALLOCATE_STREAM_PARAMS          AllocateStreamParams;
STREAM_OBJ                      StreamObj;
ATTACH_STREAM_DEVICE_PARAMS     AttachParams;
CONFIGURE_STREAM_PARAMS         ConfigStreamParams;
GET_VMC_BANDWIDTH_PARAMS        QueryVMCBandwidthParams;
ENABLE_STREAM_PARAMS            EnableStreamParams;
DISABLE_STREAM_PARAMS           DisableStreamParams;
DETACH_STREAM_DEVICE_PARAMS     DetachParams;
vmcDWORD                        dwSourcePixelRate;
vmcDWORD                        dwStreamPixelRate;
vmcDWORD                        dwDestPixelRate;
VMC_RECTINFO                    rSourceRect, rStreamRect, rDestRect;
LPVMC_RECTINFO                  lprRect;
vmcDWORD                        dwSize;
vmcHANDLE                       hMem;
vmcHANDLE                       hVSM;
LPSET_STREAM_CLIP_DATA_PARAMS   lpStreamClipParams;
```

```
/* Attempt Driver Open                  */
/* Driver details defined in [drivers] section of system.ini   */
if ((hVSM = (vmcHANDLE)OpenDriver((LPCSTR)"vmc.StreamManager",
            (LPCSTR)NULL,
            (LPARAM)NULL) ) == NULL)
{
   /* Stream Manager could not load */
}
```

- Querying Device Type Drivers and Capabilities
  Before a client can access a device a VMC Client must identify that the required device contexts are available. As the VMC Client is a component of the Video Subsystem, it knows its appropriate type driver and the capabilities.

```
/* Request Access to Video Subsystem Driver */
if ( (DeviceObj.hDriver = (vmcHANDLE)OpenDriver((LPCSTR)"vmc.videosubsystem1",
                  (LPCSTR)"vmc.TypeDrivers",
                  (LPARAM)NULL) ) == NULL)
{
   /* Type Driver could not load */
}
```

Once suitable drivers have been identified, a Driver Context can be opened for the Video subsystem. A VMC Client can use its own proprietary mechanism or standard messages to resource its Video subsystem contexts. Example here is based on generic context messages but assumes that the logical device context supports the required context type. It is recommended, for future compatibility, that VMC clients use the capability messages to determine the functionality offered by their own subsystems. Accessing the Graphic Port driver gives a more complete example on using device type drivers generically.

```
   /* define context type and board instance */
   VideoContextParams.dwSize = sizeof(CREATE_CONTEXT_PARAMS);
   VideoContextParams.wDeviceInstance = 1;
   VideoContextParams.dwContextType= CT_VIDEO_SOURCE;
   VideoContextParams.dwContextSubType= ST_VIDEO_DECODER;

   /* Request Driver to Create VIDEO_IN COntext          */
   SendDriverMessage((HDRVR)DeviceObj.hDriver,
```

```
            (UINT)DRV_VMC_CREATE_CONTEXT,
            (LPARAM)&VideoContextParams,
            (LPARAM)&VMCErrorParams);

   /* retrieve context ID and define Video device object for use with stream manager*/
   DeviceObj.dwSize = sizeof(DEVICE_OBJ);
   DeviceObj.hContext = VideoContextParams.hContext;
   DeviceObj.dwContextType = CT_VIDEO_SOURCE;
   DeviceObj.dwContextSubType = ST_VIDEO_DECODER;
```

A VMC Client uses the SYSTEM.INI file settings to load the selected Graphic Subsystem Type Driver to determine if it has the required capabilities.

```
/* Attempt Driver Open on Graphic Port             */
/* Driver defined in [vmc.TypeDrivers] section of system.ini   */
if ((GraphicObj.hDriver = (vmcHANDLE)OpenDriver((LPCSTR)"vmc.GraphicPort",
         (LPCSTR)"vmc.TypeDrivers",
         (LPARAM)NULL) == NULL) )
{
   /* Graphic SubSystem Driver will not load */
}
```

Must use Generic messages to obtain Graphic Port Capabilities to ensure Data can be inlayed in frame buffer. Request the Graphic Port driver to provide information on all supported contexts. A given Subsystem Instance may support multiple context types.

```
   /* query number of Subsystems supported */
   /* returns number of logical devices supported by driver */
   QueryNumDevicesParams.dwSize = sizeof(QUERY_NUM_VMC_DEVICES_PARAMS);
   SendDriverMessage((HDRVR)GraphicObj.hDriver,
            (UINT)DRV_VMC_QUERY_NUM_VMC_DEVICES,
            (LPARAM)&QueryNumDevicesParams,
            (LPARAM)&VMCErrorParams);

   /* Graphic Subsystem - should only be one per system */
   if (QueryNumDevicesParams.wNumVMCDevices != 1)
   {
      /* not what we expected */
   }

   /* request Graphic Subsystem  Vendor  details for  */
   /* for specified logical device instance  */
   QueryVendorParams.dwSize = sizeof(QUERY_VENDOR_INFO_PARAMS);
   QueryVendorParams.wDeviceInstance = 1;     /* logical device Instance */
   SendDriverMessage((HDRVR)GraphicObj.hDriver,
            (UINT)DRV_VMC_QUERY_VENDOR_INFO,
            (LPARAM)&QueryVendorParams,
            (LPARAM)&VMCErrorParams);


   /* ensure that Graphic Port has input capability */
   if ( (QueryVendorParams.VESADevCaps.dwVMCContextCaps & CT_GRAPHIC_SINK) )
   {
      /* Graphics system supports inlay */
   }
   else
   {
      /* Should not occur, as all Graphic subsystems support */
      /*  this context type */
   }
```

Once it has been verified that the required context is supported, the VMC Client requests the Device Type Driver to create a Context.

```
   GraphicContextParams.dwSize = sizeof(CREATE_CONTEXT_PARAMS);
   GraphicContextParams.dwContextType= CT_GRAPHIC_SINK;
   GraphicContextParams.dwContextSubType = ST_ON_SCREEN;
   GraphicContextParams.wDeviceInstance = 1;
```

```
SendDriverMessage((HDRVR)GraphicObj.hDriver,
         (UINT)DRV_VMC_CREATE_CONTEXT,
         (LPARAM)&GraphicContextParams,
         (LPARAM)&VMCErrorParams);

/* Ensure we have a valid context */
if (VMCErrorParams.wErrorCode != VMCERR_NONE)
{
   /* Problem allocating context .. maybe all in use */
   if (VMCErrorParams.wErrorCode == VMCERR_CANNOT_ALLOCATE_DEVICE_CONTEXT)
   {
      /* No free contexts */
   }
   else
   {
      /* some other problem */
   }
}
```

Once created, a VMC Client obtains from the device type driver context specific capabilities.

```
/* retrieve context ID and define graphic context object*/
GraphicObj.hContext = GraphicContextParams.hContext;
GraphicObj.dwContextType = CT_GRAPHIC_SINK;
GraphicObj.dwContextSubType = ST_ON_SCREEN;
GraphicObj.dwSize = sizeof(DEVICE_OBJ);

/* request Supported stream Modes for specified context*/
QueryTypeParams.dwSize = sizeof(QUERY_CONTEXT_CAPS_PARAMS);
QueryTypeParams.oDevice = GraphicObj;
SendDriverMessage((HDRVR)GraphicObj.hDriver,
         (UINT)DRV_VMC_QUERY_CONTEXT_CAPS,
         (LPARAM)&QueryTypeParams,
         (LPARAM)&VMCErrorParams);

if ( (QueryTypeParams.ContextCaps.wStreamModes & VSM_INTERLACED_VIDEO) )
{
   /* device supports Interlaced Video Streams */
}
else
{
   /* Problem, non supported stream type */
   /* Return */
}
```

Now get the Video Stream Capabilities

```
/* Get the Video Stream Capabilities */
QueryStreamParams.dwSize = sizeof(QUERY_STREAM_CAPS_PARAMS);
QueryStreamParams.oDevice = GraphicObj;
QueryStreamParams.wStreamMode = VSM_INTERLACED_VIDEO;
SendDriverMessage((HDRVR)GraphicObj.hDriver,
         (UINT)DRV_VMC_QUERY_STREAM_CAPS,
         (LPARAM)&QueryStreamParams,
         (LPARAM)&VMCErrorParams);


/* Check to see if we have clipping capability at destination */
if ( (QueryStreamParams.StreamCaps.dwClipCaps & CM_CLIPRECTS) )
{
   /* destination supports clipping */
}

/* Check Supported Stream Formats */
if ( (QueryStreamParams.StreamCaps.dwStreamFormats & VMCRGB_565) )
{
   /* destination supports 565 */
}

/* Check to see if standard interlaced scaling mode supported */
if ( (QueryStreamParams.StreamCaps.dwScalerModes & SM_INTERLACED) )
```

```
{
   /* destination supports standard Interlaced scaling */
}
```

Get the Scaler mode capabilities for the specified stream mode and context.

```
/* Now get scale mode parameters */
/* Get the Video Stream Capabilities */
QueryScalerParams.dwSize = sizeof(QUERY_SCALER_CAPS_PARAMS);
QueryScalerParams.oDevice = GraphicObj;
QueryScalerParams.wStreamMode = VSM_INTERLACED_VIDEO;
QueryScalerParams.dwScalerMode = SM_INTERLACED;
SendDriverMessage((HDRVR)GraphicObj.hDriver,
         (UINT)DRV_VMC_QUERY_SCALER_CAPS,
         (LPARAM)&QueryScalerParams,
         (LPARAM)&VMCErrorParams);

/* determine scaler quality */
if ((QueryScalerParams.ScalerCaps.dwScaleCaps & HSC_CAN_REPLICATE_DECIMATE_X))
{
   /* device supports horizontal scaling using replication/decimation */
}
else if ((QueryScalerParams.ScalerCaps.dwScaleCaps & HSC_CAN_INTERPOLATE_FILTER_X) )
{
   /* device supports horizontal scaling using interpolation/filtering */
}
else if ((QueryScalerParams.ScalerCaps.dwScaleCaps & VSC_CAN_REPLICATE_DECIMATE_Y) )
{
   /* device supports vertical scaling using replcation/decimation */
}
else if ((QueryScalerParams.ScalerCaps.dwScaleCaps& VSC_CAN_INTERPOLATE_FILTER_Y) )
{
   /* device supports vertical scaling using replication/filtering */
}
else
{
   /* No scaling in graphic subsystem */
}
```

- Reserving VMC Streams
  In order to use the VMC, a stream must be allocated and devices attached to that stream. The VMC Client sends a DRV_VSM_ALLOCATE_STREAM message to the Stream Manager requesting it to reserve a stream ID for the specified stream type. The Stream Manager returns to the Client a unique stream ID.

```
/*  Reserve an ID for a Video Stream  */

   AllocateStreamParams.dwSize = sizeof(ALLOCATE_STREAM_PARAMS);
   AllocateStreamParams.wStreamType = VST_VIDEO_STREAM;
   SendDriverMessage((HDRVR)hVSM,
         (UINT)DRV_VSM_ALLOCATE_STREAM,
         (LPARAM)&AllocateStreamParams,
         (LPARAM)&VMCErrorParams);

   if (VMCErrorParams.wErrorCode != VMCERR_NONE)
   {
      /* process error */
   }

   /* extract stream ID allocated */
   StreamObj.dwSize = sizeof(STREAM_OBJ);
   StreamObj.hStream = AllocateStreamParams.hStream;
   StreamObj.dwStreamType = VST_VIDEO_STREAM;
   StreamObj.hDriver = hVSM;
```

Deallocate a VMC Stream when it is no longer required.

- Attaching stream devices

  Once a stream has been reserved, the VMC Client must associate devices with the stream. This is achieved by the VMC Client sending the DRV_VSM_ATTACH_STREAM_DEVICE message to Stream Manager, requesting it to associate the specified device context with the specified stream.

  One device attached to the stream must be marked as being the transmitter. Devices can also be requested to handle stream clipping. The VMC Client is responsible for stating which device is to be the transmitter and which (if any) is to handle clipping.

```
/* Attach Video in device to stream as a transmitter */
/* and to handle clipping */

AttachParams.dwSize = sizeof(ATTACH_STREAM_DEVICE_PARAMS);
AttachParams.oStream = StreamObj;
AttachParams.oDevice = DeviceObj;
AttachParams.bIsTransmitter=vmcTRUE;
AttachParams.bIsClipDevice = vmcTRUE;

SendDriverMessage((HDRVR)hVSM,
      (UINT)DRV_VSM_ATTACH_STREAM_DEVICE,
      (LPARAM)&AttachParams,
      (LPARAM)&VMCErrorParams);
if (VMCErrorParams.wErrorCode != VMCERR_NONE)
{
   /* process errors */
}

/* Attach Graphic Port device to stream as a receiver */
AttachParams.dwSize = sizeof(ATTACH_STREAM_DEVICE_PARAMS);
AttachParams.oStream = StreamObj;
AttachParams.oDevice = GraphicObj;
AttachParams.bIsTransmitter=vmcFALSE;
AttachParams.bIsClipDevice = vmcFALSE;

SendDriverMessage((HDRVR)hVSM,
      (UINT)DRV_VSM_ATTACH_STREAM_DEVICE,
      (LPARAM)&AttachParams,
      (LPARAM)&VMCErrorParams);
if (VMCErrorParams.wErrorCode != VMCERR_NONE)
{
   /* process errors */
}
```

- Query Stream Configurations

  Once all required devices have been associated with a stream, the VMC Client needs to ascertain scaling options for a given stream configuration and available bandwidth. The VMC Client can query the Stream Manager specifying a stream format, stream mode, required destination size, source size and scaling modes. The Stream Manager returns the achievable source, stream and destination sizes and the peak source, stream and destination pixel rates.

```
   /* Determine if required scale factors can be achieved */

   /* Define Source Image Size */
   rSourceRect.iLeft = 0;
   rSourceRect.iTop = 0;
   rSourceRect.iRight = 766;
   rSourceRect.iBottom = 288;
   /* Define Required Dest Size and position*/
```

```
rDestRect.iLeft = 100;
rDestRect.iTop = 100;
rDestRect.iRight = (rDestRect.iLeft) + (vmcINT)383;
rDestRect.iBottom = (rDestRect.iTop) + (vmcINT)144;

QueryConfigParams.dwSize = sizeof(QUERY_STREAM_CONFIG_PARAMS);
QueryConfigParams.oStream = StreamObj;
QueryConfigParams.VideoStreamParams.dwStreamFormat = VMCRGB_565;
QueryConfigParams.VideoStreamParams.wBitsPerPixel = 16;
QueryConfigParams.VideoStreamParams.wStreamMode = VSM_INTERLACED_VIDEO;
QueryConfigParams.VideoStreamParams.rSourceRect = rSourceRect;
QueryConfigParams.VideoStreamParams.rDestRect = rDestRect;
QueryConfigParams.VideoStreamParams.dwSourceScaleMode = SM_INTERLACED;
QueryConfigParams.VideoStreamParams.dwDestScaleMode = SM_INTERLACED;
QueryConfigParams.bScaleAtSource = vmcTRUE;
QueryConfigParams.bMaintainAspect= vmcTRUE;

SendDriverMessage((HDRVR)hVSM,
         (UINT)DRV_VSM_QUERY_STREAM_CONFIGURATION,
         (LPARAM)&QueryConfigParams,
         (LPARAM)&VMCErrorParams);
if (VMCErrorParams.wErrorCode != VMCERR_NONE)
{
   /* process errors */
}


/* Check to see if we achieved required scale factor and get*/
/* Stream rates                        */
rDestRect = QueryConfigParams.SuggestedStreamParams.rDestRect;
dwSourcePixelRate = QueryConfigParams.StreamBandwidthParams.dwSourcePixelRate;
dwStreamPixelRate = QueryConfigParams.StreamBandwidthParams.dwVMCPixelRate;
dwDestPixelRate = QueryConfigParams.StreamBandwidthParams.dwDestPixelRate;
```

- Configuring a Stream

Once devices and contexts have been configured and the image scaling parameters determined, the stream must be configured. The VMC Client issues a DRV_VSM_CONFIGURE_STREAM message to the Stream Manager.

```
ConfigStreamParams.dwSize = sizeof(CONFIGURE_STREAM_PARAMS);
ConfigStreamParams.oStream = StreamObj;
ConfigStreamParams.VideoStreamParams.dwStreamFormat = VMCRGB_565;
ConfigStreamParams.VideoStreamParams.wBitsPerPixel = 16;
ConfigStreamParams.VideoStreamParams.wStreamMode= VSM_INTERLACED_VIDEO;
ConfigStreamParams.VideoStreamParams.rSourceRect = rSourceRect;
ConfigStreamParams.VideoStreamParams.rStreamRect = rStreamRect;
ConfigStreamParams.VideoStreamParams.rDestRect = rDestRect;

SendDriverMessage((HDRVR)hVSM,
         (UINT)DRV_VSM_CONFIGURE_STREAM,
         (LPARAM)&ConfigStreamParams,
         (LPARAM)&VMCErrorParams);
if (VMCErrorParams.wErrorCode != VMCERR_NONE)
{
   /* process errors */
}
```

- Setting Clipping Data

The VMC Client must also set any clipping data for the stream.

```
/* Register Video Window with DCI */
/* exact mechanism described in Application Note */
.
.
```

```
/*  Set default Mask to full Destination Rectangle */
/* construct Clip command data buffer */
dwSize = (DWORD) (sizeof(VMC_RECTINFO) + sizeof(SET_STREAM_CLIP_DATA_PARAMS) );
hMem = (vmcHANDLE)GlobalAlloc( (UINT)(GMEM_MOVEABLE | GMEM_SHARE), (DWORD)dwSize);
lpStreamClipParams = (LPSET_STREAM_CLIP_DATA_PARAMS) GlobalLock((HANDLE) hMem);

/*  Set default Clip region to full Destination Rectangle */
lpStreamClipParams->dwSize = sizeof(SET_STREAM_CLIP_DATA_PARAMS);
lpStreamClipParams->oStream = StreamObj;
lpStreamClipParams->ClipData.rdh.nCount = 1;
lpStreamClipParams->ClipData.rdh.iType = RDH_RECTANGLES;
lpStreamClipParams->ClipData.rdh.nRgnSize= (DWORD)sizeof(VMC_RECTINFO);
lprRect = (LPVMC_RECTINFO)&(lpStreamClipParams->ClipData.buffer[0]);
lprRect->iLeft = rDestRect.iLeft;
lprRect->iRight = rDestRect.iRight;
lprRect->iTop = rDestRect.iTop;
lprRect->iBottom = rDestRect.iBottom;

SendDriverMessage((HDRVR)hVSM,
                (UINT)DRV_VSM_SET_STREAM_CLIP_DATA,
                (LPARAM)lpStreamClipParams,
                (LPARAM)&VMCErrorParams);
if (VMCErrorParams.wErrorCode != VMCERR_NONE)
{
   /* process errors */
}
```

- Enabling a Stream
  Once devices and streams have been configured, the stream can be enabled. This has the effect of allocating bandwidth to the device, and informing devices that VMC transfers are about to commence. VMC Clients are responsible for sending a DRV_VSM_ENABLE_STREAM message to the Stream Manager, requesting that a stream be enabled.

```
/* All set up to go, enable the stream */
EnableStreamParams = sizeof(ENABLE_STREAM_PARAMS);
EnableStreamParams.oStream = StreamObj;
SendDriverMessage((HDRVR)hVSM,
      (UINT)DRV_VSM_ENABLE_STREAM,
      (LPARAM)&EnableStreamParams,
      (LPARAM)&VMCErrorParams);
if (VMCErrorParams.wErrorCode != VMCERR_NONE)
{
   /* Possibly still out of bandwidth */
}
```

- Re-Configuring an Active Stream
  If the parameters of a stream configuration need to be changed (Windows size change, for example), then the stream must be disabled, and device and stream configurations re-negotiated.

```
DisableStreamParams.dwSize = sizeof(DISABLE_STREAM_PARAMS);
DisableStreamParams.oStream = StreamObj;
SendDriverMessage((HDRVR)hVSM,
      (UINT)DRV_VSM_DISABLE_STREAM,
      (LPARAM)&DisableStreamParams,
      (LPARAM)&VMCErrorParams);
if (VMCErrorParams.wErrorCode != VMCERR_NONE)
{
}

/* Renegotiate stream configurations */
```

```
.
.
.

/* and re-enable the stream */

EnableStreamParams.dwSize = sizeof(ENABLE_STREAM_PARAMS);
EnableStreamParams.oStream = StreamObj;
SendDriverMessage((HDRVR)hVSM,
      (UINT)DRV_VSM_ENABLE_STREAM,
      (LPARAM)&EnableStreamParams,
      (LPARAM)&VMCErrorParams);
if (VMCErrorParams.wErrorCode != VMCERR_NONE)
{
   /* Possibly still out of bandwidth */
}
```

- Shutting down the Stream

  When the stream is no longer required, the stream should be closed down by first disabling the stream.

```
DisableStreamParams.oStream = StreamObj;
SendDriverMessage((HDRVR)hVSM,
      (UINT)DRV_VSM_DISABLE_STREAM,
      (LPARAM)&EnableStreamParams,
      (LPARAM)&VMCErrorParams);
if (VMCErrorParams.wErrorCode != VMCERR_NONE)
{
}

/* Detach devices from streams */

DetachParams.dwSize = sizeof(DETACH_STREAM_PARAMS);
DetachParams.oStream = StreamObj;
DetachParams.oDevice = DeviceObj;

SendDriverMessage((HDRVR)hVSM,
      (UINT)DRV_VSM_DETACH_STREAM_DEVICE,
      (LPARAM)&DetachParams,
      (LPARAM)&VMCErrorParams);

DetachParams.oDevice = GraphicObj;
SendDriverMessage((HDRVR)hVSM,
      (UINT)DRV_VSM_DETACH_STREAM_DEVICE,
      (LPARAM)&DetachParams,
      (LPARAM)&VMCErrorParams);

/* Destroy device contexts */

/* Close down access to stream Manager */
```

7. VMC Development Kit (VMCDK)

A VMC development kit is provided by VESA to assist Graphic and Video Subsystem developers. The VMCDK consists of the following:

- header file, VESA_VMC.H
- sample driver code
- sample Stream Manager code
- sample client code.

Appendix A. VESA Stream Manager (VSM) Message Set Description

This section describes the messages that must be supported by the Stream Manager along with the parameter blocks and possible error returns associated with each message. All messages received for the Stream manager arrive through the DriverProc interface. The format of this interface is:-

```
(LRESULT) CALLBACK DriverProc( (DWORD)dwDriverIndentifier,
                               (HDRVR) hDriver,
                               (UINT)wMesage,
                               (LPARAM)lpMessageParameterBlock,
                               (LPARAM)lpErrorParameterBlock)
```

Each message has its own parameter block associated with it. These are defined in Appendix F. For VESA defined driver messages, the driver should always return TRUE and report any errors in the Error Parameter Block.

Note:- All parameter blocks have a dwSize member. If the size is not correct for the the version implemented or the structure packing is incorrect, then a VMCERR_BAD_PARAMETERS error code is returned.

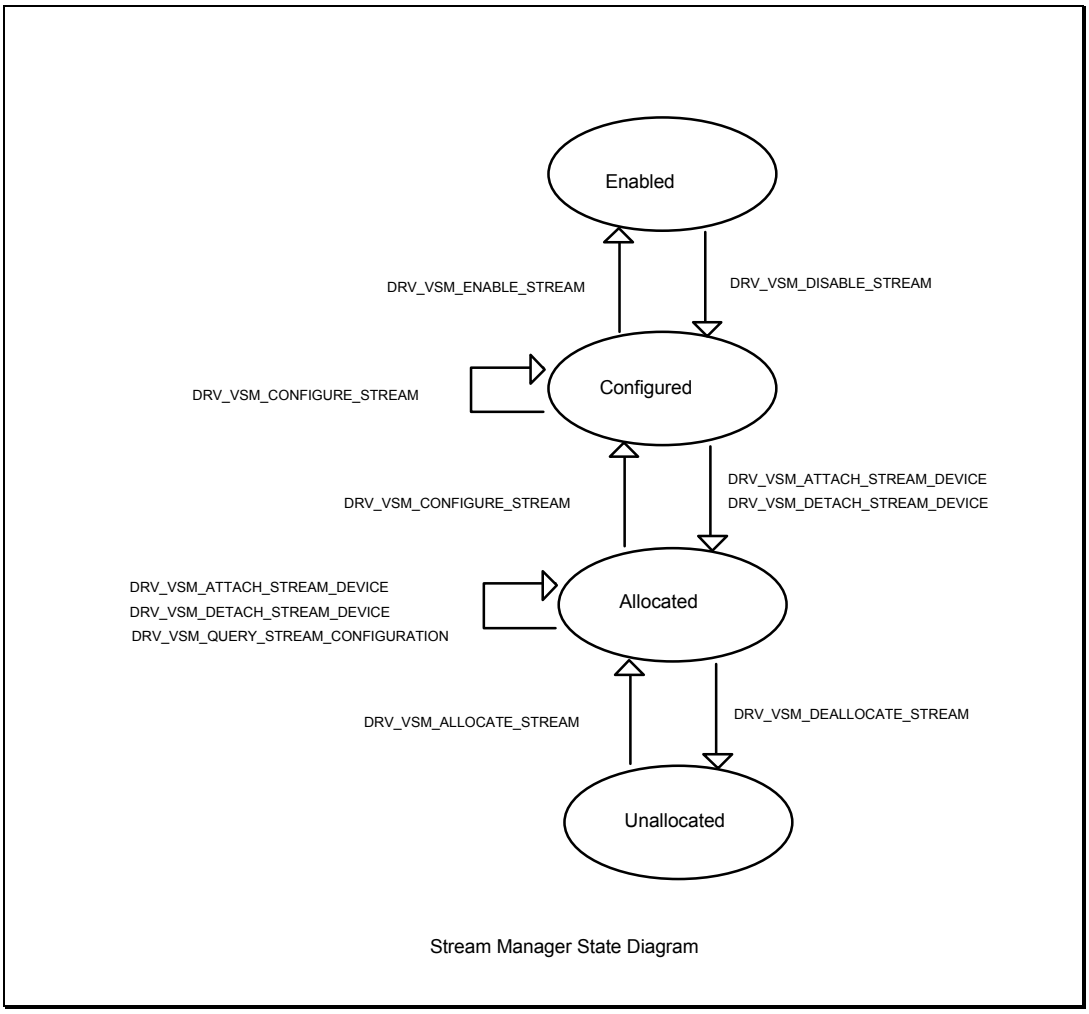
Error codes are returned in:- lpErrorParameterBlock->wErrorCode.

The value of hDriver should be returned in:- lpErrorParameterBlock->hMessageOriginator.

The set of messages that must be supported by the Stream Manager are listed below.

Message	Description
DRV_OPEN	Windows Driver Open Message
DRV_CLOSE	Windows Driver Close
DRV_CONFIGURE	Windows Query driver Configuration message
DRV_VSM_ENABLE	Enable the manager - perform any initialization
DRV_VSM_DISABLE	Disable the manager - perform any shutdown management
DRV_VSM_GET_VMC_BANDWIDTHINFO	Get available bandwidth info
DRV_VSM_QUERY_STREAM_CONFIGURATIO N	query if stream can support the required configuration.
DRV_VSM_CONFIGURE_STREAM	configure stream
DRV_VSM_ALLOCATE_STREAM	VSM allocates new stream ID.
DRV_VSM_DEALLOCATE_STREAM	VSM de-allocates stream ID.
DRV_VSM_ATTACH_STREAM_DEVICE	Associates a device with the specified stream.
DRV_VSM_DETACH_STREAM_DEVICE	Detaches a device with the specified stream.
DRV_VSM_ENABLE_STREAM	Marks the specified stream as active and allocates a GTR value to the transmitting device.
DRV_VSM_DISABLE_STREAM	Marks stream as inactive.
DRV_VSM_SET_STREAM_CLIP_DATA	Sends clip data to the device allocated to handle clipping for the specified stream
DRV_VSM_NON_STREAM_WRITE	Write a data value to a stand-alone device
DRV_VSM_NON_STREAM_READ	Read a data value from a stand-alone device
DRV_QUERY_VMC_VERSION	return version info.
DRV_GET_ERROR_TEXT	return error string for supplied error text.

The ordering of these messages from the VMC Client is important to the use of the Stream Manager. The diagram below defines the ordering of Stream Manager messages.



Stream Manager State Diagram

A.1 DRV_OPEN (standard Windows Installable Driver Message)

Message Description:

Opens access to the Stream Manager for a new Client. The Stream Manager should return a unique driver handle in response to this message. This handle can be used to detect ownership of streams in subsequent calls.

For Further information, see Windows 3.1 SDK.

Parameter Buffer Name: none

Error returns:

None

A.2 DRV_CLOSE (standard Installable driver message)

Message Description:
Closes access to the Stream Manager for the specified caller.

For Further information, see Windows 3.1 SDK.

Parameter Buffer Name: none

Error returns are:
None.

A.3 DRV_VSM_ENABLE

Message Description:
Sent by System Loader to inform the Stream Manager that it has been selected to perform the necessary VMC baseline controller functions.

It causes the VSM to Initialize the VMC controller specific registers, issues a VMChannel reset and initialize a Stream Manager instance. NOTE: this specification does not define how Stream Managers locate or communicate with their associated VMC Controller. This is considered a proprietary installation issue.

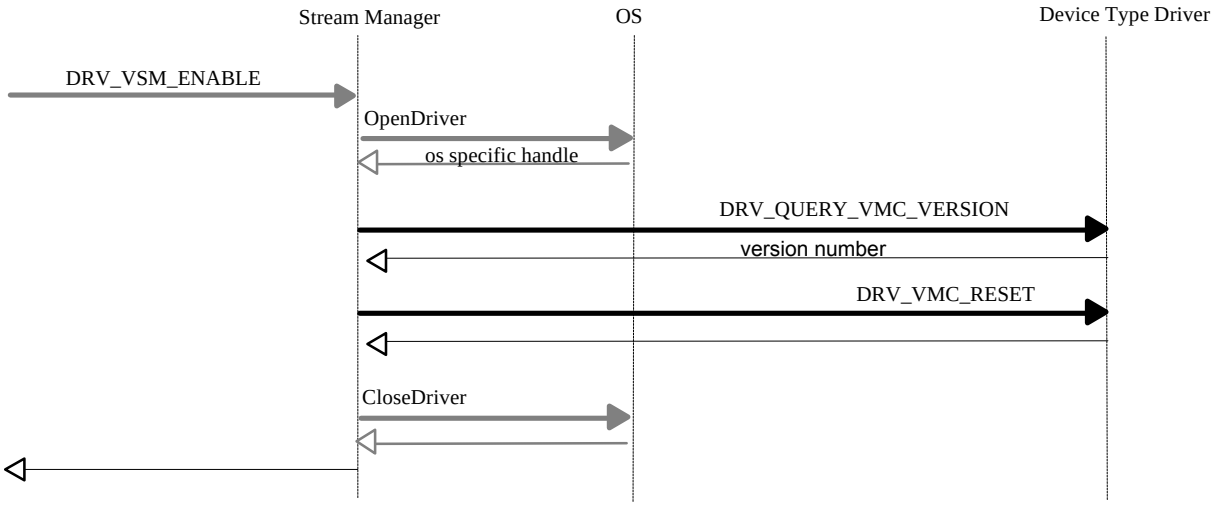
As part of its initialization process, the Stream Manager resets its bandwidth allocation tables. As the speed of the VMC bus cannot be determined from the VMC Controller, this bus speed parameter must again be considered a proprietary installation procedure.

The Stream Manager is also required to interrogate all available Device Type Drivers defined in the [vmc.TypeDrivers] section of SYSTEM.INI and issue a DRV_QUERY_VMC_VERSION and a DRV_VMC_RESET message to each one in turn.

If a driver fails to return the expected version number (VMC_VERSION_NUMBER) then a warning message is displayed.

Parameter Buffer Name: none

Message Flow:



Defined return values are:
none.

A.4 DRV_VSM_DISABLE

Message Description:
Sent by System Loader to inform the driver that it should disable the VMC controller and perform and driver specific clean-up, such as deallocation of data structures.

The Stream Manager should load all Device Type Drivers in turn, issue a DRV_VMC_RESET message to each one and then unload the driver.

Parameter Buffer Name:
No parameters associated with this message.

Defined return values are:
None.

A.5 DRV_VSM_ALLOCATE_STREAM

Message Description:
Sent by a VMC Client to request a VMC stream ID for the specified stream type.

The Stream Manager must provide for 16 VMChannel Stream ID's taking the value range 00h and 0Fh inclusive.

Associated with each stream is a list of up to 16 VMChannel device that can be associated with that stream.

When the Stream Manager receives this message it looks though the stream ID list looking for a spare stream ID.

If one is found, it is marked as being for use with the specified type. The Stream ID is returned to the caller. The returned Stream ID value provided is that which will actually be programmed into the device stream ID register by the caller.

If an invalid stream type is specified, VMCERR_INVALID_STREAM_TYPE is returned.

If no spare stream ID's are available, VMCERR_NO_STREAMS_AVAILABLE is returned.

Parameter Buffer Name: LPALLOCATE_STREAM_PARAMS

Error returns:
VMCERR_NO_STREAMS_AVAILABLE
VMCERR_INVALID_STREAM_TYPE

A.6 DRV_VSM_DEALLOCATE_STREAM

Message Description:
Sent by a VMC Client to free up the specified stream.

Before processing this message, the Stream Manager performs the following checks:-

- The stream ID is valid, otherwise the VMCERR_STREAM_ALREADY_DEALLOCATED error code is returned.
- A stream cannot be deallocated while its is still enabled or has devices associated with it. If this message is received while the stream still has devices associated with it then the VMCERR_STREAM_STILL_ASSOCIATED error code is returned.

Assuming the stream ID is a valid one and that the stream is disabled and has no associated devices, the stream is freed and marked unused.

Parameter Buffer Name: LPDEALLOCATE_STREAM_PARAMS

Error returns:
VMCERR_STREAM_ALREADY_DEALLOCATED
VMCERR_STREAM_STILL_ASSOCIATED

A.7 DRV_VSM_ATTACH_STREAM_DEVICE

Message Description:
Sent by a VMC Client to associate the specified VMC device context with the specified stream and to define whether the device is a transmitting or receiving device and whether the device will be handling video clipping.

Before processing this message, the Stream Manager performs the following checks:

- The stream is disabled, otherwise the VMCERR_STREAM_ENABLED error code is returned.
- The stream ID must be an already allocated Stream ID, otherwise the VMCERR_INVALID_STREAM_OBJ error code is returned.
- The Device Context must be one that has already been obtained through a device open. The call fails if the Stream Manager cannot access the specified driver (the VMCERR_INVALID_DEVICE_OBJ error code is returned).
- Only one transmitter allowed per stream. The message returns the VMCERR_TRANSMITTER_ALREADY_ALLOCATED error code if a transmitter has already been allocated.

On receiving this message, the Stream Manager first request that the specified device returns a VMC_DEVICECAPS information block. This is because the Stream Manager needs to know device bus width and device buffering capabilities for subsequent calculations of the devices GTR.

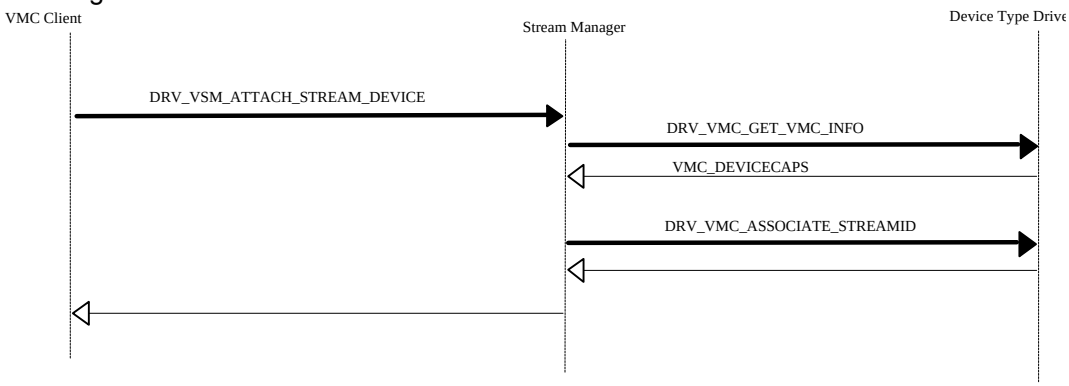
Having obtained the VMC capabilities, the Stream Manager checks that the specified device can operate in the specified mode (by checking the wVMCModes member of the VMC_DEVICECAPS structure). If not, an error is returned. Otherwise, a DRV_VMC_ASSOCIATE_STREAMID message is sent to the associated device context, informing it of the stream ID to associate with the context. If the DRV_VMC_ASSOCIATE_STREAMID message returns with an error, this error code is then returned to the caller.

Specifying an ID of a device that has already been associated with the stream is ignored.

Applying the message to an already configured stream will have the effect of De-configuring that stream. The caller will be required to re-issue a stream configuration command. If this message is received for an already configured stream, a DRV_VMC_STREAM_UNPREPARE message is sent to all currently associated devices.

Parameter Buffer Name: LPATTACH_STREAM_DEVICE_PARAMS

Message Flow:



Error returns:

VMCERR_STREAM_ENABLED
VMCERR_INVALID_STREAM_OBJ
VMCERR_INVALID_DEVICE_OBJ
VMCERR_TRANSMITTER_ALREADY_ALLOCATED

A.8 DRV_VSM_DETACH_STREAM_DEVICE

Message Description:

Sent by a VMC Client to de-associate the specified VMC device context from the specified stream. The stream must be disabled in order for this message to be valid.

Before processing the message, the Stream Manager performs the following checks:

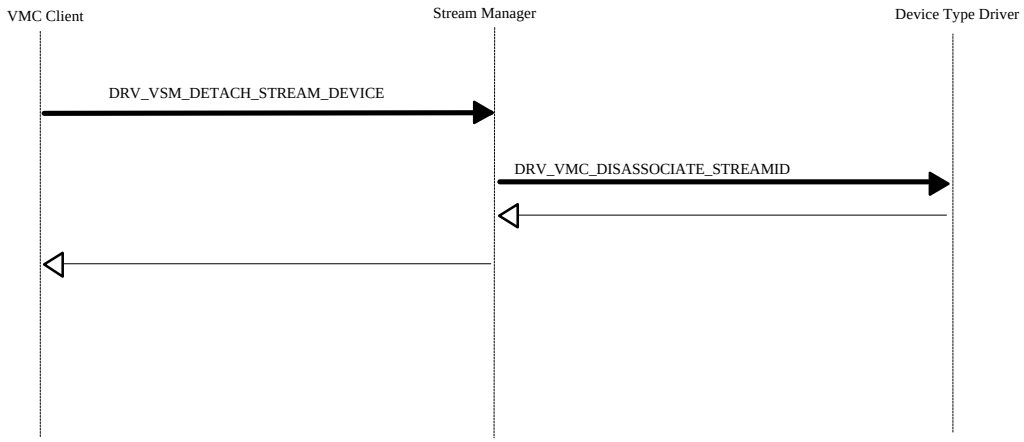
- ensures that the stream is currently disabled. If not, the call fails with the VMCERR_STREAM_ENABLED error code.
- ensures that the specified device ID is already associated. If not, the call fails with the VMCERR_INVALID_STREAM_OBJ error code.
- the specified device context handle is valid. If the Stream Manager cannot communicate with the driver the call fails with the VMCERR_INVALID_DEVICE_OBJ error code.

On receiving this message, the Stream manager sends a DRV_VMC_DISASSOCIATE_STREAMID message to the specified device context, informing the device context that it no longer has a stream ID associated with it.

Applying the message to an already configured stream will have the effect of De-configuring that stream. The caller will be required to re-issue a stream configuration command. If this message is received for an already configured stream, a DRV_VMC_STREAM_UNPREPARE message is sent to all currently associated devices.

Parameter Buffer Name: LPDETACH_STREAM_DEVICE_PARAMS

Message Flow:



Error returns:
VMCERR_STREAM_ENABLED
VMCERR_INVALID_STREAM_OBJ
VMCERR_INVALID_DEVICE_OBJ

A.9 DRV_VSM_ENABLE_STREAM

Message Description:
Sent by a VMC Client to instructs the Stream Manager to enable VMC transfers on the specified stream.

Before the message is processed, the following additional checks must be made:

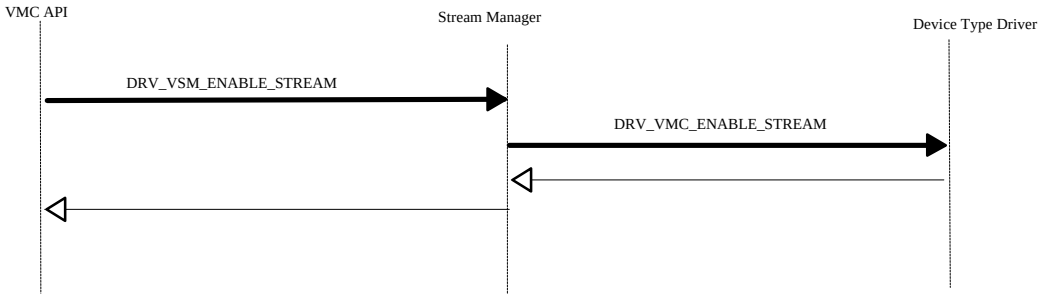
- the stream ID must be valid. If not, the call fails with the VMCERR_INVALID_STREAM_OBJ error code.
- the stream must have been configured (see DRV_VSM_CONFIGURE_STREAM). If not the call fails with the VMCERR_STREAM_NOT_CONFIGURED error code.

On the receipt of this message a DRV_VMC_ENABLE_STREAM message is sent to all receivers. A receiving device can reject an enable command if it does not have sufficient bandwidth If one device on the stream fails then the call fails with VMCERR_INSUFFICIENT_BANDWIDTH. The caller must request bandwidth information from the appropriate parties to establish a new configuration

Finally, the Stream Manager send a DRV_VMC_ENABLE_STREAM message, informing the transmitter of its GTR value and that data transfers can commence on the specified stream. See Appendix C on allocating GTR values on Stream enable.

Parameter Buffer Name: LPENABLE_STREAM_PARAMS

Message Flow:



Error returns:
VMCERR_INVALID_STREAM_OBJ
VMCERR_INSUFFICIENT_BANDWIDTH
VMCERR_STREAM_NOT_CONFIGURED

A.10 DRV_VSM_DISABLE_STREAM

Message Description:
Sent by a VMC Client to disable the specified stream.

This has the effect of freeing any bandwidth allocated to the specified stream.

The following checks are performed:

- The stream ID must be an already allocated Stream ID. If not, the VMCERR_INVALID_STREAM_OBJ error code is returned.
- A stream must be enabled. If not, the VMCERR_STREAM_NOT_ENABLED error code is returned.

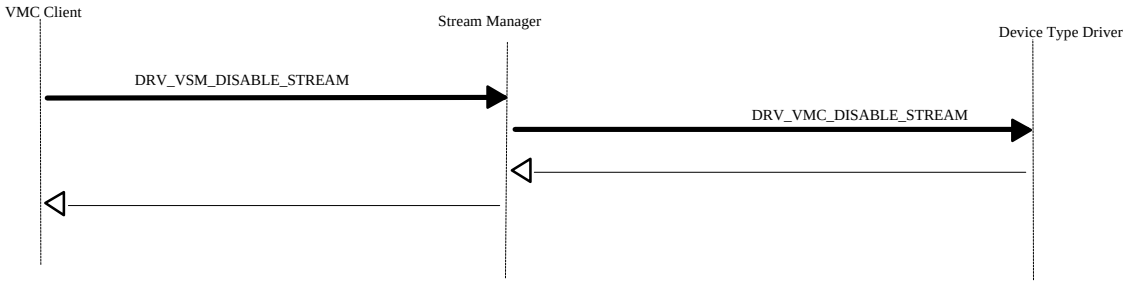
Assuming these checks are passed, the Stream Manager needs to re-assess the bandwidth requirements for the device. See Appendix C, Allocating Device GTRs on Stream Disable.

Once a new GTR value has been calculated, the Stream Manager sends a DRV_VMC_DISABLE_STREAM message to the transmitter, informing the transmitter of its new GTR value and that data transfers must cease on the specified stream.

The Stream Manager then sends a DRV_VMC_DISABLE_STREAM message to all receiving devices associated with the stream, informing them that data transmissions have stopped for the specified stream.

Parameter Buffer Name: LPDISABLE_STREAM_PARAMS

Message Flow:



Error returns:

VMCERR_INVALID_STREAM_OBJ
VMCERR_STREAM_NOT_ENABLED

A.11 DRV_VSM_CONFIGURE_STREAM

Message Description:

Sent by the VMC Client to configure the specified stream. A stream can only be configured when a stream ID has been allocated and all the required devices have been attached to that stream. The type of configuration information passed through depends on the Stream type. Currently only VIDEO configurations are supported.

It is intended that a VMC Client would have first validated the stream configuration using the DRV_VSM_QUERY_STREAM_CONFIGURATION. Calling this message without having validated the stream configuration may result in unpredictable behavior.

The Stream Manager performs the following checks:

- must be a valid stream ID. If not, the VMCERR_INVALID_STREAM_OBJ error code is returned.
- the stream must have been allocated a transmitter and one or more receivers. If not, the VMCERR_INVALID_CONFIGURATION error code is returned.
- the stream must not be enabled. If not, the VMCERR_STREAM_ENABLED error code is returned.

When the Stream Manager receives the stream configuration details it must perform the following:

- calculates a GTR value for the stream.
- sets the source and destination device stream configuration.

To calculate a GTR value for the stream, the Stream Manager needs to determine the Stream bandwidth requirement (DB_{stream}). Before the bandwidth can be determined devices must be informed of their VMC configuration.

To configure devices, the Stream Manager sends a DRV_VMC_STREAM_PREPARE message to all stream devices, informing them of the stream configuration. A device may reject this message if the specified configuration cannot be supported, and the Stream Manager returns an VMCERR_INVALID_CONFIGURATION error code.

Once the stream has been configured, the DB_{stream} value is obtained from the transmitting device through sending it a DRV_VMC_QUERY_CONTEXT_BANDWIDTH message and is specified in Pixels Per Second. The Stream Manager then uses the procedure Allocating Stream GTRs on Stream Configuration , as described in Appendix C (Assigning GTR Values), to calculate the GTR value for the stream.

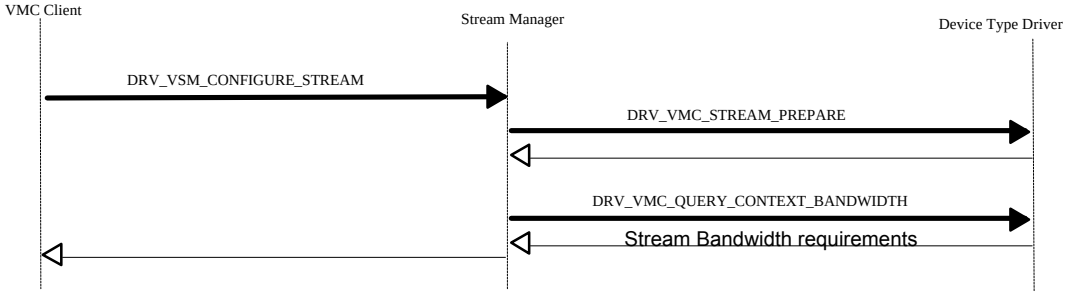
If the required DB_{stream} value is greater than the available bandwidth, then the VMCERR_INSUFFICIENT_BANDWIDTH message is returned.

Although a GTR value for the stream has been allocated at this stage, it is not sent to the transmitting device until the stream is enabled.

If a client tries to associate/deassociate another device with an already configured stream then that stream configuration becomes invalidated, and the stream must be re-configured.

Parameter Buffer Name: LPCONFIGURE_STREAM_PARAMS

Message Flow:



Error returns:
VMCERR_INVALID_STREAM_OBJ
VMCERR_STREAM_ENABLED
VMCERR_INSUFFICIENT_BANDWIDTH
VMCERR_INVALID_CONFIGURATION

A.12 DRV_VSM_QUERY_STREAM_CONFIGURATION

Message Description:
Sent by a VMC Client to request the Stream Manager to verify the configuration for the specified stream. VMC Client supplies Stream Format, and pixel depth, Stream Mode, source and destination scaler modes, Source Size, Stream Size and Destination Size parameters.

The Stream Manager performs the following:

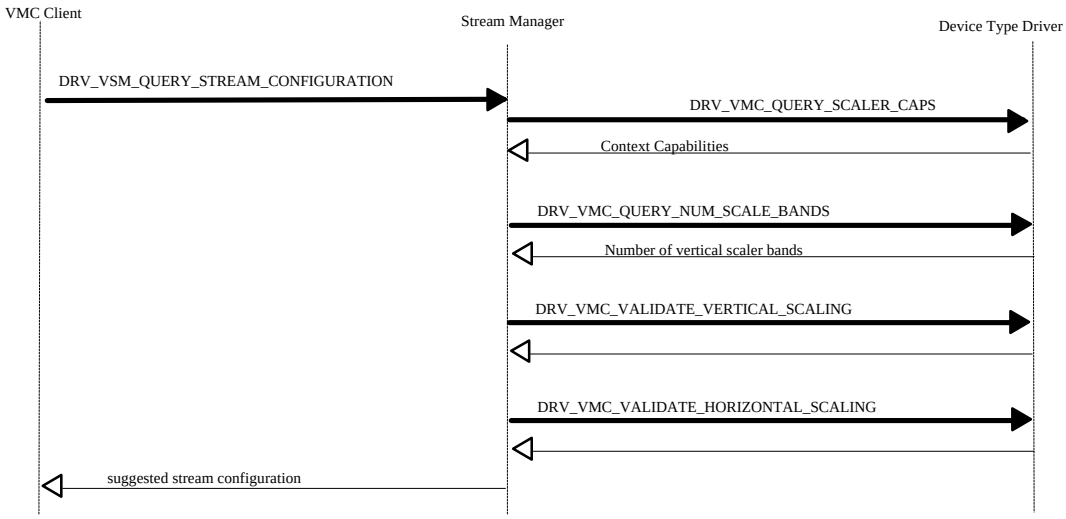
- Checks to see if specified scaling can be achieved given device and VMC constraints. To do this the Stream Manager uses the mechanism outlined in Appendix D.

If the specified colour format, stream mode, scaling modes or scaling factors are not supported, either the VMCERR_INVALID_STREAM_FORMAT or the VMCERR_INVALID_SCALE_FACTORS error code is returned.

If the required scaling factors are supported by source and destination devices but cannot be achieved due to bandwidth limitations, success is returned but the actual scaling values that can be achieved across the source and destination returned to the caller will be different to those requested.

Parameter Buffer Name: LPQUERY_STREAM_CONFIG_PARAMS

Message Flow:



Error returns:
VMCERR_INVALID_STREAM_OBJ
VMCERR_INVALID_STREAM_FORMAT
VMCERR_INVALID_SCALE_FACTORS

A.13 DRV_VSM_GET_VMC_BANDWIDTHINFO

Message Description:
Sent by VMC Client to obtain current VMC bandwidth usage.

Parameter Buffer Name: LPGET_VMC_BANDWIDTH_PARAMS

Defined error returns are:
none

A.14 DRV_VSM_SET_STREAM_CLIP_DATA

Message Description:
Sent by VMC Client requesting the Stream Manager to forward the supplied clip data to the device allocated to handle clipping for the specified stream.

The following checks are performed:-

- that a clipping device has been allocated. The VMCERR_INVALID_DEVICE_OBJ error code is generated if no clipping device has been allocated.

In response, the stream Manager prepares a DRV_VMC_SET_CLIP_DATA message to send to the allocated clipping device.

Parameter Buffer Name: LPSET_STREAM_CLIP_DATA_PARAMS

Defined error returns are:
VMCERR_INVALID_DEVICE_OBJ.

A.15 DRV_VSM_NON_STREAM_WRITE

Message Description:

Sends the specified number of 32-bit words to the specified VMC device ID starting at the specified address. Only the first 24 bits of the address are significant.

The stream Manager is responsible for setting up its bus controller to initiate the transfer to the stand-alone device and for communicating with its Bus Controller to ascertain whether or not the transfer completed. The completion status must be reported back to the caller.

If the Bus Controller cannot handle multiple writes, the supplied buffer must be broken down into individual write commands.

A non-stream write can fail because there is no device at the specified VMC Device ID or that the specified device does not support non-stream writes.

Parameter Buffer Name: LPNON_STREAM_WRITE_PARAMS

Error returns:
VMCERR_NO_SUCH_DEVICE
VMCERR_NON_STREAM_WRITE_NOT_SUPPORTED.

A.16 DRV_VSM_NON_STREAM_READ

Message Description:
Reads the specified number of 32-bit words from the specified VMC device ID starting at the specified address into the specified buffer. Only the first 24 bits of the address are significant.

The Stream Manager is responsible for communicating with its Bus Controller to first issue a non-stream write to the specified device and then awaiting the reply from the specified device.

A non-stream read can fail because there is no device at the specified VMC Device ID or that the specified device does not support non-stream reads.

Parameter Buffer Name: LPNON_STREAM_READ_PARAMS

Error returns:
VMCERR_NO_SUCH_DEVICE
VMCERR_NON_STREAM_READ_NOT_SUPPORTED.

A.17 DRV_GET_ERROR_TEXT

Message Description:
Sent by VMC Client to return a null terminated error string for the supplied error code.

Parameter Buffer Name: LPERROR_TEXT_PARAMS

Error returns :
none.

A.18 DRV_QUERY_VMC_VERSION

Message Description:
Sent by a VMC Client to request the version number of this driver.

The Stream Manager always returns VMC_VERSION_NUMBER, as defined in appendix E.

Parameter Buffer Name: LPQUERY_VERSION_PARAMS

Error returns :
none.

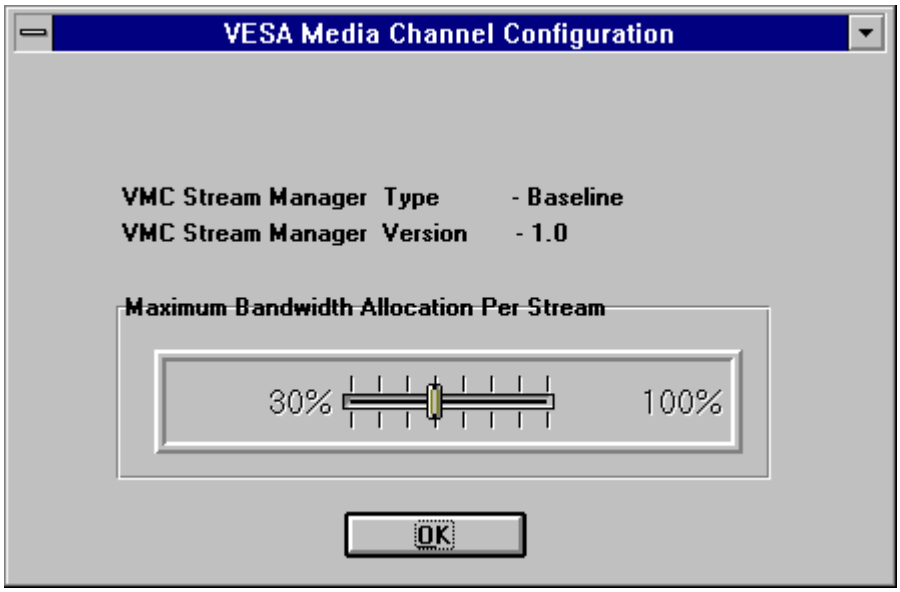
A.19 DRV_CONFIGURE (Standard Windows Driver Message)

Message Description:
Stream Managers should return TRUE to the DRV_QUERYCONFIGURE message. The Windows Control Panel configuration mechanism is used to setup the selected Stream Manager. To date the only configurable parameters are:

- Maximum bandwidth allocation per device. This provides a coarse tuning mechanism to enable the user to decide how much bandwidth to allocate to each transmitting device on the VMC. It assumes that if applications will have to adapt themselves to the available bandwidth. For example, reduce the frame rate or window size.

An example configuration panel is shown below.

In addition, the revision of the Stream Manager should also be indicated in the configuration panel, as shown below:



Parameter Buffer Name: None

Error returns :
none.

Appendix B. VMC Video Type Driver Message Set

This section describes the messages that must be supported by Device Type Drivers along with the parameter blocks and possible error returns associated with the message. All messages received for the Type Driver arrive through the DriverProc interface. The format of this interface is:-

```
(LRESULT) CALLBACK DriverProc( (DWORD)dwDriverIndentifier,
                               (HDRVR) hDriver,
                               (UINT)wMesage,
                               (LPARAM)lpMessageParameterBlock,
                               (LPARAM)lpErrorParameterBlock)
```

Each message has its own parameter block associated with it. These are defined in Appendix F. For VESA defined driver messages, the driver should always return TRUE and report any errors in the Error Parameter Block.

Note:- All parameter blocks have a dwSize member. If the size is not correct for the the version implemented or the structure packing is incorrect, then a VMCERR_BAD_PARAMETERS error code is returned.

Error codes are returned in:- lpErrorParameterBlock->wErrorCode.

wErrorCode must be set to VMCERR_NONE if no error has occurred.

The value of hDriver should be returned in:- lpErrorParameterBlock->hMessageOriginator.

Type Drivers for host addressable devices communicate with their device directly using a proprietary mecanism.

Type Drivers for stand-alone devices communicate with their device using the non-stream read and write message interface of the Stream Manager. To this end, Type Drivers for standalone devices must set up a communication path to the Stream Manager through the Windows OpenDriver() call. The key to the Stream Manager can be obtained from SYSTEM.INI.

The defined messages for Device Type Drivers are listed below.

Message	Description
DRV_OPEN	Open driver
DRV_CLOSE	Close Driver
DRV_ENABLE	Enable Driver
DRV_VMC_CREATE_CONTEXT	Create a video device context
DRV_VMC_DESTROY_CONTEXT	Destroy a device context
DRV_VMC_QUERY_VENDOR_INFO	Return vendor VESA device information block
DRV_VMC_QUERY_NUM_VMC_DEVICES	Return number of VMC device instances supported by driver.
DRV_VMC_RESET	Reset all supported devices
DRV_VMC_QUERY_CONTEXT_CAPS	Request the driver to return context capability details.
DRV_VMC_QUERY_STREAM_CAPS	Request driver to return details on stream capabilities for the specified stream mode.
DRV_VMC_QUERY_SCALER_CAPS	Request driver to return details on scaler capabilities for the specified scaler mode.

DRV_VMC_STREAM_PREPARE	Prepare stream i/f using supplied configuration details
DRV_VMC_STREAM_UNPREPARE	Stream configuration now invalid.
DRV_VMC_ENABLE_STREAM	VMC stream enable
DRV_VMC_DISABLE_STREAM	VMC stream disable
DRV_VMC_QUERY_VMC_INFO	Return vmc specific device information block
DRV_VMC_ASSOCIATE_STREAMID	Sets stream ID and device type
DRV_VMC_DISASSOCIATE_STREAMID	Device no longer associated with a stream
DRV_VMC_SET_CLIP_DATA	Set clip data
DRV_VMC_QUERY_NUM_SCALE_BANDS	Get number of vertical scale bands generated by transmitting device.
DRV_VMC_QUERY_CONTEXT_BANDWIDTH	Get the Peak bandwidth for the context configuration.
DRV_VMC_VALIDATE_VERTICAL_SCALING	Request driver to validate the vertical scale factors for the current device configuration
DRV_VMC_VALIDATE_HORIZONTAL_SCALING	Request driver to validate the horizontal scale factors for the current device configuration
DRV_GET_ERROR_TEXT	Returns error text associated with error code
DRV_QUERY_VMC_VERSION	Returns version info.

B.1 DRV_OPEN (standard Windows Installable Driver Message)

Message Description:
Sent by a VMC Client or the Stream Manager to open access to the Device Type Driver. The Driver should return a unique handle in response to this message. This handle can be used to detect ownership of contexts in subsequent calls.

For Further information, see Windows 3.1 SDK.

Parameter Buffer Name: none

Error returns:
None

B.2 DRV_CLOSE (standard Windows Installable Driver Message)

Message Description:
Sent by a VMC Client or the Stream Manager to close access to the Device Driver.

For Further information, see Windows 3.1 SDK.

Parameter Buffer Name: none

Error returns are:
None.

B.3 DRV_ENABLE (standard Windows Installable Driver Message)

Message Description:
Sent by a Windows when the driver is to be enabled.

The Device Type Driver uses this message to Initialize the VESA_DEVICECAPS, VMC_DEVICECAPS and type specific capability structures. In addition, the driver must interrogate each supported device to obtain its VMC device ID and product information. If

a device cannot be located then this should be recorded and any request to create a context for that device instance should be rejected.

For Further information, see Windows 3.1 SDK.

Type Drivers for Standalone devices must poll (via the Stream Manager) all possible VMC device ID's (0-15) for the Vendor ID and Device Type register information appropriate to its device. A VMCERR_NO_SUCH_DEVICE or VMCERR_NON_STREAM_READ_NOT_SUPPORTED error message for a given VMC device ID indicates that the device does not support standalone access and the returned data for that device should be ignored. A VMCERR_NONE return indicates a stand-alone device and the returned information can be queried.

Parameter Buffer Name: none

Error returns are:
None.

B.4 DRV_VMC_RESET

Message Description:
Sent by the Stream Manager, requesting the Device Type Driver to reset all associated devices to their default state (i.e disabled from transmitting over VMC).

Parameter Buffer Name: none.

Error returns:
VMCERR_NO_SUCH_DEVICE

B.5 DRV_VMC_CREATE_CONTEXT

Message Description:
Sent by a VMC Client, requesting the driver to open the specified context type and subtype for the specified Logical VMC device instance in support of a VMC stream and returns to the caller the ID of the created context. In addition, for video subsystems, a subtype can be specified to define the routing of VMC stream data.

The driver performs the following checks:-

- That the specifed device instance is in the range 1 to <max supported instances>. If not the VMCERR_INVALID_DEVICE_INSTANCE error code is generated.
- The device instance should be validated to ensure that the device is present, If not, a VMCERR_NO_SUCH_DEVICE error is generated.
- The context type and subtype are supported by the driver. If not, the VMCERR_INVALID_CONTEXT_TYPE error code is returned.

The number of supported contexts is device dependent but one device context is associated with one VMC stream. The driver must decide whether the specified context type can be supported for the specified device. If so, it allocates a unique ID that is used in subsequent calls for sending message to the context. Otherwise, a VMCERR_CANNOT_ALLOCATE_DEVICE_CONTEXT message is returned.

A device must never allocate a device context ID of zero, as this may be interpreted by VMC clients as an invalid context ID.

A device context block is returned to the caller, informing it of its context ID and context type.

Parameter Buffer Name: LPCREATE_CONTEXT_PARAMS

Error returns:
VMCERR_CANNOT_ALLOCATE_DEVICE_CONTEXT
VMCERR_INVALID_CONTEXT_TYPE
VMCERR_NO_SUCH_DEVICE
VMCERR_INVALID_DEVICE_INSTANCE

B.6 DRV_VMC_DESTROY_CONTEXT

Message Description:
Sent by VMC Clients to request that the specified device context be destroyed. The driver must free up any data structures associated with the context and mark the context as invalid.

Before destroying a context the driver must check the following:-

- That the specified device context is valid. If not, the VMCERR_INVALID_DEVICE_OBJ error code is returned.
- That the specified context is not associated with a stream. If this message is received while the device is still associated with a stream then the VMCERR_CONTEXT_STILL_ASSOCIATED error is reported and the context is not destroyed.

Parameter Buffer Name: LPDESTROY_CONTEXT_PARAMS

Error returns:
VMCERR_INVALID_DEVICE_OBJ
VMCERR_CONTEXT_STILL_ASSOCIATED.

B.7 DRV_VMC_QUERY_NUM_VMC_DEVICES

Message Description:
Sent by VMC Clients to request Device Type Driver to return the number of supported VMC device instances.

The number of supported VMC device instances is returned in the supplied buffer. This value returned represents the maximum logical device number supported by the Device Type driver. Logical device numbers always start at 1. Therefore the valid range of logical VMC device numbers supported by the driver is in the range 1 to <max number of supported devices>

If, for some reason, a device type driver does not support any devices (i.e the board has been removed) then a VMCERR_DEVICES_NOT_AVAILABLE message is returned.

Parameter Buffer Name: LPQUERY_NUM_VMC_DEVICES_PARAMS

Error returns:
VMCERR_DEVICES_NOT_AVAILABLE

B.8 DRV_VMC_QUERY_VENDOR_INFO

Message Description:
Sent by VMC Clients to request device information block for the specified Logical device instance. The logical device instance relates to the number returned from the DRV_VMC_QUERY_NUM_VMC_DEVICES message.

The information is returned in the supplied buffer.

If the specified device instance is less than one or greater than the number of supported devices then the VMCERR_INVALID_DEVICE_INSTANCE error is returned.

Parameter Buffer Name: LPQUERY_VENDOR_INFO_PARAMS

Error returns:
VMCERR_INVALID_DEVICE_INSTANCE

B.9 DRV_VMC_QUERY_CONTEXT_CAPS

Message Description:
Sent by the Stream Manager and VMC Clients to request driver to return its capabilities for the specified context.

The driver ensures that the specified context is valid before filling in the structure defined by QUERY_CONTEXT_CAPS_PARAMS, which is then returned to the caller.

Parameter Buffer Name: LPQUERY_CONTEXT_CAPS_PARAMS

Error returns:
VMCERR_INVALID_DEVICE_OBJ

B.10 DRV_VMC_QUERY_STREAM_CAPS

Message Description:
Sent by the Stream Manager and VMC Clients to request driver to return the stream capabilities for the specified context and stream mode.

Before returning the stream capabilities, the driver checks for the following:-

- a valid device context. If not valid, the VMCERR_INVALID_DEVICE_OBJ error code is returned.
- the specified stream mode is valid. If not, the VMCERR_INVALID_STREAM_MODE error code is returned.

Assuming the specified context and stream mode are valid, the driver fills in the QUERY_STREAM_CAPS_PARAMS structure, which is then returned to the caller.

Parameter Buffer Name: LPQUERY_STREAM_CAPS_PARAMS

Error returns:
VMCERR_INVALID_DEVICE_OBJ
VMCERR_INVALID_STREAM_MODE

B.11 DRV_VMC_QUERY_SCALER_CAPS

Message Description:

Sent by a VMC client or the Stream Manager to obtain the scaler capabilities for the specified device context, stream mode and scaler mode.

Before returning the scaler capabilities, the driver checks for the following:-

- a valid device context. If not valid the VMCERR_INVALID_DEVICE_OBJ error code is returned.
- the specified stream mode is supported by the context. If not, the VMCERR_INVALID_STREAM_MODE error code is returned.
- the specified scaler mode is supported by the stream mode. If not, the VMCERR_INVALID_SCALER_MODE error code is returned.

Assuming the specified context, stream mode and scaler mode are valid, the driver fills in the QUERY_SCALER_CAPS_PARAMS structure, which is then returned to the caller.

Parameter Buffer Name: LPQUERY_SCALER_CAPS_PARAMS

Error returns:
VMCERR_INVALID_DEVICE_OBJ
VMCERR_INVALID_SCALER_MODE
VMCERR_INVALID_STREAM_MODE

B.12 DRV_VMC_STREAM_PREPARE

Message Description:
Sent by the Stream Manager to a device context informing it of the configuration for the specified stream. The device is required to configure its context accordingly.

The driver performs the following sanity checks:

- ensure the specified stream ID is associated with the context. If not, VMCERR_INVALID_STREAM_OBJ is returned.
- ensure the stream type and configuration is supported. If not, VMCERR_INVALID_STREAM_FORMAT is returned.
- ensure the stream configuration does not violate device context capabilities. If not, VMCERR_INVALID_STREAM_FORMAT is returned.

If any of these checks fails, an error is reported and the message processing is abandoned.

Otherwise, a device context is configured according to the defined parameters.

The configuration parameters are stream dependent. The currently supported types are: VIDEO_STREAM.

The configuration parameters are defined by VIDEO_STREAM_CONFIG member of the parameter buffer.

The driver configures its VMC interface, scaler and, in cases where the scaler generates additional pixels due to rounding errors, setup up its output clipper to restrict the stream to the required output size.

Parameter Buffer Name: LPSTREAM_PREPARE_PARAMS

Error returns :

VMCERR_INVALID_STREAM_FORMAT
VMCERR_INVALID_STREAM_OBJ

B.13 DRV_VMC_STREAM_UNPREPARE

Message Description:
A message sent by the Stream Manager to a device context informing it that the current stream configuration is no longer valid.

The driver performs the following sanity checks:

- ensure the specified stream ID is associated with the context. If not, the VMCERR_INVALID_STREAM_OBJ error code is returned.
- ensure the stream is disabled. If not, the VMCERR_STREAM_ENABLED error code is returned.

If any of these checks fails, an error is reported.

Parameter Buffer name: LPSTREAM_UNPREPARE_PARAMS

Error returns :
VMCERR_INVALID_STREAM_OBJ
VMCERR_STREAM_ENABLED

B.14 DRV_VMC_ENABLE_STREAM

Message Description:
Message sent by the Stream Manager to the device context informing it to enable transfers to (transmitter) or from (receivers) the VMC for the specified stream.

Driver performs the following sanity checks:

- the stream ID is valid. If not, the VMCERR_INVALID_STREAM_OBJ error code is returned.
- the device has been prepared for use on the VMC. If not, the VMCERR_DEVICE_NOT_PREPARED error code is returned.

3. the device has sufficient h/w contexts to support the device. If not, the VMCERR_CANNOT_ALLOCATE_DEVICE_CONTEXT error code is returned.

Included in the message is a GTR value that should be programmed into the device when the stream is enabled. Only necessary if different from the currently defined GTR.

A driver should not return from this call until GTR allocation is guaranteed for the specified stream.

Parameter Buffer name: LPENABLE_CONTEXT_PARAMS;

Error returns:
VMCERR_INVALID_STREAM_OBJ

B.15 DRV_VMC_DISABLE_STREAM

Message Description:
Message sent by the Stream Manager to the device context informing it to stop transfers to/from the channel.

Driver performs the following sanity checks:

- the stream ID is valid. If not, the VMCERR_INVALID_STREAM_OBJ error code is returned.
- the stream is currently enabled. If not, th VMCERR_STREAM_NOT_ENABLED error code is returned.

Included in the message is a GTR value that should be programmed into the device when the specified stream is disabled. Only necessary if different from the currently defined GTR.

Parameter Buffer Name: LPDISABLE_CONTEXT_PARAMS;

Error returns :
VMCERR_INVALID_STREAM_OBJ
VMCERR_STREAM_NOT_ENABLED

B.16 DRV_VMC_QUERY_VMC_INFO

Message Description:
Message sent by the Stream Manager or a VMC Client to request the VMC device information block for the specified device context.

The information is returned in the supplied buffer.

Parameter Buffer Name: LPQUERY_VMC_DEVINFO_PARAMS;

Error returns are:
VMCERR_INVALID_DEVICE_OBJ

B.17 DRV_VMC_ASSOCIATE_STREAMID

Message Description:
Message sent by the Stream Manager to associate a stream ID with the context and informs device as to whether it is to be a transmitter or receiver. Can only be applied while the context has a null stream ID (i.e is currently a unallocated context).

If the stream ID for the context is not null, then an already allocated message is returned to the Stream Manager.

The driver should also ensure that it can act in the required VMC mode (transmit or receive). If not, an invalid mode error is returned.

Parameter Buffer Name: LPASSOCIATE_STREAMID_PARAMS

Error returns:
VMCERR_STREAMID_ALREADY_ALLOCATED
VMCERR_INVALID_DEVICE_MODE

B.18 DRV_VMC_DISASSOCIATE_STREAMID

Message Description:

Sent by the Stream Manager to Inform the device context that there is no longer a stream ID associated with the device context.

Cannot be applied while stream is still enabled.

Parameter Buffer Name: LPDISASSOCIATE_STREAMID_PARAMS;

Error returns :
VMCERR_STREAM_ENABLED

B.19 DRV_VMC_SET_CLIP_DATA

Message Description:
Message sent by the Stream Manager to inform the device context that there is new clipping data to be set for the stream.

Parameter Buffer Name: LPSET_CLIP_DATA_PARAMS

Error returns :
VMCERR_CLIPPING_RECTANGLES_USED

B.20 DRV_VMC_QUERY_CONTEXT_BANDWIDTH

Message Description:
Message sent by the stream manager to obtain the bandwidth requirements for the specified context, stream mode, stream format, scaler mode and scale factors.

This message only applies to transmitting devices.

The device driver first checks it supports the specified stream mode and format and scaler mode. If not, an error is returned.

The bandwidth requirements (which must represent the peak throughput) are calculated as follows:

PeakBandwith = LineRate * StreamHorizontalScaleFactor * StreamVerticalScaleFactor
where:-

LineRate = Rate (SourceWidth / LineRate) at which pixels enter Source device - StreamHorizontalScaleFactor = actual horizontal scale factor applied by device. - defined by StreamWidth/SourceWidth.
StreamVerticalScaleFactor = actual vertical scale factor applied but rounded up to next integer. eg 1-100% = 1, 101%-200% = 2, 201%-300% = 3, etc. - defined by StreamHeight/SourceHeight.

eg given a line Rate of 768 Pixels in 64uS, a horizontal scale factor of 150% and vertical scale factor of 150%, then:-
PeakStreamRate = 12000000 * 1.5 * 2 = 36MPixels/Sec.

The PeakBandwidth value is then returned to the caller and is expressed in pixels per second.

Parameter Buffer Name: LPQUERY_CONTEXT_BANDWIDTH_PARAMS

Error returns :
VMCERR_INVALID_DEVICE_OBJ

VMCERR_INVALID_STREAM_FORMAT
VMCERR_INVALID_SCALE_FACTORS

B.21 DRV_VMC_QUERY_NUM_SCALE_BANDS

Message Description:
Message sent by the Stream Manager requesting it to determine the number of vertical scale bands for the specified context, stream mode, scaler mode and scale factors.

A suggested method for calculating the number of vertical scale bands is illustrated in 'Determining Vertical Scale Bands' in Appendix D.

Parameter Buffer Name: LPQUERY_NUM_SCALE_BAND_PARAMS

Error returns :

B.22 DRV_VMC_VALIDATE_VERTICAL_SCALING

Message Description:
Message sent by the Stream Manager requesting the driver to validate the supplied vertical scaling factors.

The means for validating vertical scale factors is illustrated in 'Source and Destination Scale Factor Validation' in Appendix D.

Parameter Buffer Name: LPVALIDATE_SCALING_PARAMS

Error returns :
VMCERR_BANDWIDTH_LIMITED

B.23 DRV_VMC_VALIDATE_HORIZONTAL_SCALING

Message Description:
Message sent by the Stream Manager requesting the driver to validate the supplied horizontal scaling factors.

The means for validating horizontal scale factors is illustrated in 'Source and Destination Scale Factor Validation' in Appendix D.

Parameter Buffer Name: LPVALIDATE_SCALING_PARAMS

Error returns :
VMCERR_BANDWIDTH_LIMITED

B.24 DRV_GET_ERROR_TEXT

Message Description:
Return a null terminated error string for the supplied error code.

Parameter Buffer Name: LPERROR_TEXT_PARAMS

Error returns :
none.

B.25 DRV_QUERY_VMC_VERSION

Message Description:
Return the version number for this driver.

The driver always returns VMC_VERSION_NUMBER in response to this call, as defined in Appendix E.

Parameter Buffer Name: LPQUERY_VERSION_PARAMS

Error returns :
none.

Appendix C. Assigning Grant Time Register Values

The Stream Manager allocates bandwidth through defining a Grant Time Register for a transmitting device. Transmitting devices are time multiplexed onto the VMChannel, for a duration defined by its Grant Time Register (GTR). i.e GTR allocation must be on a per transmitting device basis.

The Stream Manager does not perform dynamic channel bandwidth allocation and uses the notion of a maximum ring time for the token to be passed around transmitting devices using the inherent round robin scheduling mechanism built into the VMChannel hardware interface. This is fixed at 4uS, in line with the minimum VMChannel buffering requirements.

A GTR value is assigned to all streams. The one actually programmed into a transmitting device is based on the maximum GTR allocated to each stream originating from that device. GTR values are assigned to devices when streams are enabled and disabled.

Once a stream has been enabled, a stream is considered free running. If changes to the stream characteristics are required then the stream must be disabled so that GTR's can be reassessed. Once disabled, device and stream characteristics can be re-configured prior to the stream being re-enabled.

VMC stream control is illustrated below.

Bus Clock Rate	- [25MHz or 33MHz]
NumVMCDevices	- number of devices attached to VMC.
MTRT	- maximum token rotation time = 4uS.
MTRTCLKS	- maximum rotation time in Clocks = MTRT x Bus Clock Rate
TransitionOverhead	- 2clocks x NumVMCDevices
MBDLY	- Maximum Bus Delay = ∑Allocated GTR's + TransitionOverhead
BufferSize	- Buffer size of transmitting device
DB_{stream}	- Transmitting Stream Peak Data Rate = peak rate at which a stream is filling a device FIFO.
DB_{device}	- Transmitting Device Data Rate = maximum DB_{stream} for the device.
BB	- Bus Bandwidth = Bus Clock rate x Bytes transferred per clock (based on narrowest bus width used by the stream).
GTR_{max}	- Maximum GTR that can be allocated to a device, calculated as a percentage of MTRCLKS defined by the user.
GTR_{stream}	- Bandwidth required by a given stream
GTR_{device}	- Maximum $GTR_{stream\ value}$ associated with a device.
$GTR_{current}$	- GTR value currently assigned to a device

Querying Available VMC bandwidth

Spare VMC bandwidth available to a given stream must also take into account any bandwidth already consumed by another stream originating from the same device.

Spare capacity(bytes per second) = ((MTRTCLKS - (MBDLY - GTR_{device})) / MTRTCLKS) * BB

Allocating Stream GTRs on Stream Configuration

When a stream is configured, bandwidth should be reserved for the stream, but not programmed into the device until stream enable.

The procedure for allocating a Stream GTR is as follows:-

Allocate a GTR (in VMChannel clock cycles) to a stream as a proportion of MTRTCLKS. The Stream Manager calculates DB_{stream} from the stream configuration and is type dependent.

The stream GTR is calculated as follows:

$$GTR_{Stream} = (DB_{stream} / BB_{stream}) * MTRTCLKS$$

The GTR_{device} value for a transmitting device is defined as the maximum GTR_{stream} for that device (this includes all currently enabled streams plus that just configured).

The GTR_{device} can only be the same as or an increment of a GTR already allocated to the device. An existing GTR value for a device is remembered by the Stream Manager and is represented here as GTR_{device}.

Additional VMC bandwidth only needs to be reserved if the new GTR is greater than the existing GTR.

```
if (GTR_stream > GTR_device)
{
        /*
         * Ensure adding the extra GTR dose not exceed the MTRT or the maximum
        allowed per  * device (as configured from the Windows Control Panel).
        */

        if ( (MBDLY + (GTR_stream - GTR_device)) > MTRTCLKS)
        {
            Error. - insufficient bandwidth
        }

        if (GTR_stream > GTR_max )
        {
            Error. - insufficient bandwidth
        }

        /* Ensure buffering restrictions are not broken */
        if (MBDLY >  (BufferSize_xmit * BB / (DB * (BB-DB)))
        {
            Error. - insufficient device buffering
        }

        /* Update MBDLY  */
        MBDLY = MBDLY + (GTR_stream - GTR_device)

        /* Remember the GTR allocated to the specified device */
        GTR_device =  GTR_stream
    }
    else
    {
     sufficient bandwidth
    }
```

Allocating Device GTR's on Stream Enable

Device GTR's are allocated when a stream is enabled.
Enable the stream transmitter, supplying it with GTR_{device}.

Allocating Device GTR's on Stream Disable

A new Device GTR's needs to be allocated when a stream is disabled.

The GTR_{device} value for a transmitting device is defined as the maximum GTR_{stream} for the device and includes all currently enabled streams but excludes the Stream being disabled.

The GTR_{device} value must be same as or a decrement of the GTR already allocated to the device.

Upon disabling the stream, first the transmitting device is disabled (supplying it with a new GTR_{device} value for the remaining streams) followed by the receiving devices associated with the stream.

Appendix D. Video Stream Scaling Validation

D.1 Introduction

For a given Source and Destination image size, a VMC Client defines where in the stream the scaling is carried out (at source or destination). The VMC Client defines the members of the VideoStreamParams member of the QUERY_STREAM_CONFIG_PARAMS structure associated with the DRV_VSM_QUERY_STREAM_CONFIGURATION message sent by the VMC Client to the Stream Manager. The VMC Client specifies a source and a destination size, source and destination scaler modes, Stream data format and stream mode. In addition, the VMC Client also specifies if scaling is to be carried out at the transmitter or receiver.

The scaling that can actually be achieved will be limited by the various bandwidth bottlenecks present in a stream and the characteristics of the scaling devices.

Characteristics affecting stream scaling and bandwidth usage have been identified as:

Bandwidth Availability

The bandwidth available to a given video stream is affected by the following criteria:

- Maximum rate at which transmitter can source data.
- Available VMC bandwidth. The available bandwidth is governed by the number of bytes per pixel and the number of free clock cycles
- Rate at which Destination device can take data off VMC.
- Rate at which Destination device can output data (eg into VRAM).

The bandwidth available to a stream is defined as the minimum value of these criteria.

Bandwidth limitations will affect video scaling across a VMC stream as follows:

If output from destination device is the limiting factor, then the image size is dictated by the destination output pixel rate.

If bandwidth limitations are at source, over VMC or at the destination input, the required image size may still be achieved through scaling down at source and up at the destination (depending on source and destination scaling capabilities).

Scaler Capabilities

Devices at either end of a stream may well have differing scaling capabilities; in how much they can scale, how well they scale and how accurately they can scale. The VMC architecture allows a device to have separate capabilities for horizontal and vertical scaling. Thus when validating scaling across a stream, the Stream Manager should handle horizontal and vertical scaling separately.

One of the biggest problems affecting scaling validation is one of alignment, especially when devices have different scaler accuracy's. The possible scaler granularity's defined are PRECISE, INTEGER or NONE. The process of validating scaling parameters where devices have non-precise scalers is iterative, especially in the light of bandwidth limitations.

An integer scaler should, where possible, always attempt to over-scale. The Stream Manager will pull scale factors in if possible. On a first pass, given a target destination size, an image should (if exact scale factors are not attainable) be over-scaled. In this case the Stream Manager needs to adjust the stream size such that the best fit is realised.

Vertical Scale Bands

Horizontal scaling produces a linear effect on bandwidth while vertical scaling produces a step effect on peak bandwidth. Identifying the number of Vertical scale bands generated by a transmitter helps identification of bandwidth consumption and enables segmentation of vertical scaling between a transmitter and receiver.

For each vertical band generated by the transmitter, the available stream bandwidth is known and allows the receiver to attempt to scale the image to the required destination size. To achieve this, the number of vertical scale bands generated by the source device is controlled by the stream height. For the first scale band the source and stream height are equal and the receiver is given the opportunity to scale the image completely in the vertical direction. For each subsequent scale band, the stream height is incremented by the source height and means the source is required to scale more and the destination device less. If 'Scale at destination' has been selected, then stream height is set to the source height.

As horizontal scaling produces a linear effect on bandwidth, for a given scaler band, if bandwidth is limited, then the image width is reduced sufficiently to fall below the bandwidth threshold. Height cannot be reduced as it would take a move into a lower scaler band to have any effect on bandwidth. This will have occurred as a matter of course.

The number of scaler bands depends on the vertical scale capability of the transmitting device and where the scaling has been requested.

If the VMC Client requests scaling at source then the number of scale bands is defined as:-

((VerticalScalefactor-1)/100) + 1

where 1:1 scale factor = 100%

If scaling is requested at destination, then the number of scale bands is 1.

The Stream Manager must take into these factors into account when validating sacle factors.

D.2 Scale Validation Process

For a given stream configuration specified by the VMC Client (source and destination image size, stream type and format and scaling preference), the stream manager builds a table defining the scaling options and bandwidth usage which most closely matches the requested stream configuration.

The Stream Manager must also obtain, from the source and destination device their scaler capabilities for the specified scaler mode and stream mode. This provides the Stream Manager with details about device bandwidth limitations and scaler granularity's. This is achieved through sending the source and destination devices associated with the stream a DRV_VMC_QUERY_SCALER_CAPS message.

The table generated by the Stream Manager is segmented by the number of vertical bands that will be generated by the transmitter in trying to achieve the required destination height. For example, if the required vertical scale factor is 250% and scaling has been requested at source, then 3 vertical scale bands will be required.

The number of vertical scale bands are obtained from the transmitting device through sending it a DRV_VMC_QUERY_NUM_SCALE_BANDS message.

For each vertical scale band the stream manager must validate the vertical and horizontal scale requirements in light of device capabilities and bandwidth limitations. The horizontal and vertical scale factors are defined by the source, stream and destination heights.

The source Vertical Scale factor is defined by (StreamHeight / SourceHeight).
The source horizontal scale factor is defined by (StreamWidth / SourceWidth).

The destination vertical scale factor is defined by (DestinationHeight / StreamHeight).
The destination horizontal scale factor is defined by (DestinationWidth / StreamWidth).

Each vertical scale band has a VIDEO_STREAM_CONFIG block that holds the following information:-,

- source scaler mode
- destination scaler mode
- stream mode
- stream format
- stream pixel depth
- source image size
- stream image size
- destination image size
- Peak Stream Bandwidth available to transmitting device

Each VIDEO_STREAM_CONFIG structure in the table must be initialised as follows:

The stream mode, source and destination scaler modes, stream format, stream pixel depth and source and destination image size are set as defined by the VMC Client.

If the VMC Client has requested scaling at source, then the stream size is set to the destination size otherwise it is set to the source size. The stream height has then to be set so that it forces the transmitter to generate the correct vertical scale factor. So, for example, if the required number of vertical scale factor was 3, the for the first vertical scale band the source height and the stream height would be equal. For the second band, the stream height would be twice the source height, and for the third band the stream height would be 3 times the source height.

If the VMC Client requests scaling at destination, the stream size is set to the source size and there is only one vertical scale band.

The Stream Manager is responsible for setting the Bandwidth available to the stream transmitter. This will be the minimum of the source peak output Rate, Available VMC bandwidth, destination peak input rate and destination peak output rate. The Stream Manager obtains device bandwidth characteristics from the devices scaler mode capabilities. The Bandwidth limit for a receiver is defined by its Peak Output Pixel Rate (as defined in its Scaler mode Capabilities). This limit is expected to be known by the receiver and does not form part of the Stream Configuration Block.

For each configuration block, the Stream Manager requests the source and destination devices to validate the scale factors. This is done in two stages. First the vertical scale factors and then the horizontal scale factors.

To validate the vertical scale factors, the Stream Manager sets up a VALIDATE_SCALING_PARAMS structure and sends a DRV_VMC_VALIDATE_VERTICAL_SCALING message first to the transmitter and then to the receiver.

The transmitter returns the stream height it can actually achieve and should be the same as the stream height. The stream manager then requests the receiver to validate the destination vertical scale factor as defined by the stream height (that achieved by the transmitter) and the Destination height. If the destination device employs an integer scaler, it is recommended that the device over-scales rather than under-scale if it cannot exactly scale to the requested size

If the required destination height is not met exactly, the Stream Manager needs to re-adjust the Stream Height, such that it is aligned on the vertical scaling accuracy's of the source and destination devices. Once adjusted the Stream Manager requests the source and destination device to re-validate the adjusted parameters. Vertical scale validation does not take into account bandwidth limitations as these are inherent in the scale band.

Once the vertical scale factors for the current band have been validated, the Stream Manager can validate the horizontal factors.

As for vertical validation, the Stream Manager must validate the horizontal scale factors of the source and destination devices and re-adjust the stream width if the desired destination width was not achieved. The Stream Manager sends a DRV_VMC_VALIDATE_HORIZONTAL_SCALING message first to the transmitter and then to the receiver. When validating horizontal scale factors, source and destination devices must take into account the available bandwidth as these may limit the scale factors that can be applied. A transmitting device must always produce a stream size that is a achievable by its scaler capabilities. (i.e if the source has an integer scaler, then the stream size must always be an integer multiple of the source size). If the source device is bandwidth limited, then the stream size will be less than the requested stream size. If not bandwidth limited and the requested width cannot be achieved, then a transmitting device may over-scale. A receiving device will also try and generate a destination size that is achievable by its scaler and may over-scale if necessary. The exception is when the destination device is bandwidth limited. Here, the destination device returns and image width that exactly consumes the available bandwidth. The destination device must flag that it is bandwidth limited, otherwise the Stream Manager will believe that the required scaling factors have been achieved.

If the required image width was not achieved or the stream was bandwidth limited, the stream manager must re-adjust the stream and destination widths so that they are aligned on the horizontal scaling accuracy's of the source and destination devices and/or do not exceed the bandwidth limited image width. Having adjusted the stream and destination widths, the Stream Manager then requests the source and destination devices to re-validate the horizontal scaling factors.

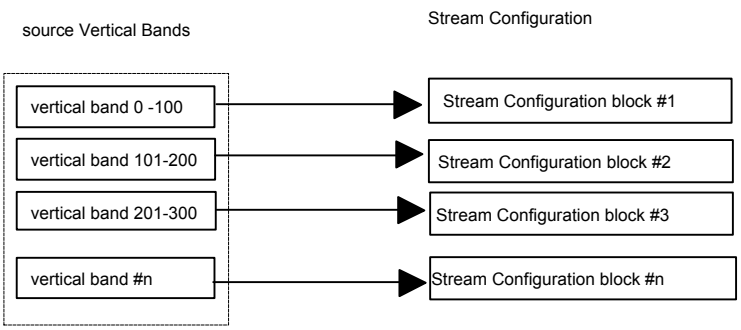
Having validated the vertical and horizontal scale factors, the achieved image size may be of an incorrect aspect ratio. The Stream Manager then needs to further adjust the stream height or stream width so that correct aspect can be maintained.

The working copy of the validated stream configuration for the current scale band is then copied into the stream configuration table and the next scale band is processed.

Having repeated the above process for each scale band, the stream Manager has built a picture, of how an image can be scaled over VMC given source and destination scaling capabilities and bandwidth requirements for the specified stream configuration.

The Stream Manager returns to the VMC Client the Stream Configuration that best matches the requested Stream Configuration in terms of size and aspect.

The format of the table is shown below.



Below, a number of code extracts are given to further describe the Validation procedure.

- Main Validation Loop

The following code example demonstrates the form of the main validation loop

```
/*
 * Function:   QueryStreamConfig
 *
 */
VMC_ERR FAR PASCAL DoQueryStreamConfig( LPQUERY_STREAM_CONFIG_PARAMS lpQueryConfigParams)
{
VMC_ERR                 Ret;
LPVMC_STREAM_MODE       lpStreamMode;
LPVIDEO_STREAM_CONFIG      lpStreamFormat;
LPVMC_RECTINFO          lprSrcRect, lprDestRect;
QUERY_SCALER_CAPS_PARAMS      SourceScalerParams;
QUERY_SCALER_CAPS_PARAMS      DestScalerParams;
vmcHANDLE               hMem;
vmcDWORD             dwStreamFormat;
vmcWORD                 wNumVerticalBands;
vmcINT                  iBand, iScaleBand;
vmcINT                  iSourceHeight, iDestHeight, iStreamHeight;
vmcINT                  wBitsPerPixel;
vmcLONG                 lDestWidth, lDestHeight;
vmcLONG                 lRequiredAspectRatio;
vmcBOOLEAN              bAdjusted;


lprSrcRect  = &(lpQueryConfigParams->VideoStreamParams.rSourceRect);
lprDestRect = &(lpQueryConfigParams->VideoStreamParams.rDestRect);


/* Extract target destination size and aspect ratio */
GetParamsFromRect(lprDestRect, &lDestWidth, &lDestHeight, &lRequiredAspectRatio);

/* Get source capabilities for specified stream configuration */
SourceScalerParams.wStreamMode = lpQueryConfigParams->VideoStreamParams.wStreamMode;
SourceScalerParams.dwStreamFormat=lpQueryConfigParams->VideoStreamParams.dwStreamFormat;
SourceScalerParams.wBitsPerPixel=lpQueryConfigParams->VideoStreamParams.wBitsPerPixel;
SourceScalerParams.dwScalerMode=lpQueryConfigParams->VideoStreamParams.dwSourceScaleMode;
if ( (Ret = GetDeviceScalerCaps(hTransmitter, &SourceScalerParams)) != VMCERR_NONE)
{
      return Ret;
}

/* Get destination scaler parameters for specified stream configuration */
DestScalerParams.wStreamMode = lpQueryConfigParams->VideoStreamParams.wStreamMode;
DestScalerParams.dwStreamFormat = lpQueryConfigParams->VideoStreamParams.dwStreamFormat;
DestScalerParams.wBitsPerPixel = lpQueryConfigParams->VideoStreamParams.wBitsPerPixel;
DestScalerParams.dwScalerMode = lpQueryConfigParams->VideoStreamParams.dwDestScaleMode;
if ( (Ret = GetDeviceScalerCaps(hReceiver, &DestScalerParams)) != VMCERR_NONE)
{
   return Ret;
}
```

```
InitIntegerScaleTables((vmcWORD) DestScalerParams.ScalerCaps.wXScaleFactorMax,
                    (vmcWORD) SourceScalerParams.ScalerCaps.wXScaleFactorMin );

/* Get number of vertical scale bands supported by transmitter for the target stream
configuration */
if ( (Ret = GetNumVerticalBands(lpQueryConfigParams,
                                  (vmcIPTR) &wNumVerticalBands)) != VMCERR_NONE)
{
   return Ret;
}

/* Extract Stream format and  bytes per pixel */
if ( (Ret = GetStreamFormatData(lpQueryConfigParams,
                    (vmcDPTR)&dwStreamFormat, (vmcWPTR)&wBitsPerPixel)) != VMCERR_NONE)
{
   return Ret;
}

/* Create Stream Mode Characteristics Table */
lpStreamMode = (LPVMC_STREAM_MODE) (NULL);
Ret =    CreateVMCScalingTable( (LPHANDLE) &hMem,
                                (vmcPTRPTR) &lpStreamMode, (vmcWORD) wNumVerticalBands);
if (Ret != VMCERR_NONE)
{
   /* failed to allocate table */
   return Ret;
}

/* initialise Table header */
lpStreamMode->wStreamMode = (vmcWORD)lpQueryConfigParams->VideoStreamParams.wStreamMode;
lpStreamMode->wNumVerticalBands = (vmcWORD)wNumVerticalBands;

/* initialise table entries --Source and destination sizes defined by Client */
/* Stream size set according to whether scaling is to be performed at   */
/* source or destination */
   InitScalingTableEntries(lpStreamMode,
                    lprSrcRect,
                    lprDestRect,
                    dwStreamFormat,
                    wBitsPerPixel,
                    lpQueryConfigParams->bScaleAtSource);

/* Build the stream Characteristics table for specified source and destination sizes */
/* and specified stream mode and data format                                  */
for (iBand = 0, iScaleBand = 1; iBand < (vmcINT)lpStreamMode->wNumVerticalBands; iBand++,
iScaleBand++)
{
/* Initialise Table for required scaling characteristics */
/* handle to this format entry for current vertical scale band */
lpStreamFormat    = &(lpStreamMode->StreamEntry[iBand]);

/*  adjust stream height for this scale band */
iSourceHeight  = lprSrcRect->iBottom - lprSrcRect->iTop;
iDestHeight    = lprDestRect->iBottom - lprDestRect->iTop;
iStreamHeight  = iScaleBand * iSourceHeight;
if (iStreamHeight > iDestHeight)
{
   iStreamHeight = iDestHeight;
}
lpStreamFormat->rStreamRect.iBottom =  lpStreamFormat->rStreamRect.iTop +
                                    iStreamHeight;

/* FIRST, Verify the vertical scaling across source and destination devices */
if (VerifyVerticalScaleFactors( (LPVIDEO_STREAM_CONFIG)lpStreamFormat,
                              (vmcINT) iBand,
                              (LPVIDEO_SCALER_CAPS) &(SourceScalerParams.ScalerCaps),
                              (LPVIDEO_SCALER_CAPS) &(DestScalerParams.ScalerCaps),
                              (vmcWORD) wBitsPerPixel) != vmcFALSE)
{
      /* SECOND, Verify the horizontal scaling across source and destination devices */
       if (VerifyHorizontalScaleFactors( (LPVIDEO_STREAM_CONFIG)lpStreamFormat,
                                    (vmcINT) iBand,
```

```
                                (LPVIDEO_SCALER_CAPS)
                               &(SourceScalerParams.ScalerCaps),
                                (LPVIDEO_SCALER_CAPS)
                               &(DestScalerParams.ScalerCaps),
                                (vmcWORD) wBitsPerPixel) != vmcFALSE)
      {
         if ( lpQueryConfigParams->bMaintainAspect == vmcTRUE )
         {
            /* see if any adjustment is required to maintain aspect ratio */
            bAdjusted = VerifyAspectRatio(   (LPVIDEO_STREAM_CONFIG)lpStreamFormat,
                           (LPQUERY_STREAM_CONFIG_PARAMS) lpQueryConfigParams,
                           (vmcINT) iBand,
                           (LPVIDEO_SCALER_CAPS) &(SourceScalerParams.ScalerCaps),
                           (LPVIDEO_SCALER_CAPS) &(DestScalerParams.ScalerCaps),
                           (vmcWORD) wBitsPerPixel);
            if (bAdjusted == vmcTRUE)
            {
               /*  Re-Verify the scaling with source and destination devices */
               VerifyVerticalScaleFactors( (LPVIDEO_STREAM_CONFIG)lpStreamFormat,
                           (vmcINT) iBand,
                           (LPVIDEO_SCALER_CAPS) &(SourceScalerParams.ScalerCaps),
                           (LPVIDEO_SCALER_CAPS) &(DestScalerParams.ScalerCaps),
                           (vmcWORD) wBitsPerPixel);

               VerifyHorizontalScaleFactors( (LPVIDEO_STREAM_CONFIG)lpStreamFormat,
                           (vmcINT) iBand,
                           (LPVIDEO_SCALER_CAPS) &(SourceScalerParams.ScalerCaps),
                           (LPVIDEO_SCALER_CAPS) &(DestScalerParams.ScalerCaps),
                           (vmcWORD) wBitsPerPixel);
            }
         }
      }
   }
   }

   /* Check to see if we got anywhere near the required destination size */
   Ret = DoQueryTable( (LPVMC_STREAM_MODE) lpStreamMode, (LPQUERY_STREAM_CONFIG_PARAMS)
lpQueryConfigParams);

if (lpStreamMode != (LPVMC_STREAM_MODE) (NULL) )
{
      VMCMemUnlock(hMem, (vmcPTRPTR)&lpStreamMode);
      VMCMemFree(hMem);
   }

   return Ret;
}
```

- Validating Scale Factors

An example of validating source and destination horizontal scale factors is illustrated below. A very similar mechanism would be used for the validation of vertical scale factors with the exception that bandwidth limitation calculations would not be necessay.

```
vmcBOOLEAN VerifyHorizontalScaleFactors(LPVIDEO_STREAM_CONFIG lpStreamConfig,
                           vmcINT                  iScaleBand,
                           LPVIDEO_SCALER_CAPS    lpSourceScalerCaps,
                           LPVIDEO_SCALER_CAPS  lpDestScalerCaps,
                           vmcWORD                 wBitsPerPixel)
{

    VMC_ERR             Ret;
   VALIDATE_SCALING_PARAMS ValidateFormatParams;
   vmcLONG             lStreamPixelRateMax, lSourcePixelRateMax, lDestPixelRateMax;
    LPVMC_RECTINFO         lprSrcRect, lprDestRect, lprStreamRect;
   vmcINT              iRequestedWidth;
   vmcLONG             lDestWidth;
```

```
/* lpStreamConfig holds the rectangles we are trying to achieve */
/* ValidateFormatParams holds a working state of the stream and destination rectangles */
/* as scaling negotiation between source and destination progresses towards the required
scaling rectangles */

ValidateFormatParams.StreamConfig      = *lpStreamConfig;
ValidateFormatParams.SourceScalerMode  = *lpSourceScalerCaps;
ValidateFormatParams.DestScalerMode    = *lpDestScalerCaps;

/* remeber requested destiantion width we are trying to achieve */
iRequestedWidth  = lpStreamConfig->rDestRect.iRight - lpStreamConfig->rDestRect.iLeft;

/* set up local references to working structure */
lprSrcRect    = &(ValidateFormatParams.StreamConfig.rSourceRect);
lprStreamRect = &(ValidateFormatParams.StreamConfig.rStreamRect);
lprDestRect   = &(ValidateFormatParams.StreamConfig.rDestRect);


/* Calculate Maximum bandwidth available to source device */
/* pixel rates specified as max mumber of pixels per second */
lSourcePixelRateMax = lpSourceScalerCaps->dwPeakOutputPixelRate;
lDestPixelRateMax = min( (vmcLONG)lpDestScalerCaps->dwPeakInputPixelRate,
                         (vmcLONG)lpDestScalerCaps->dwPeakOutputPixelRate );

/* calculate avialable PIXEL bandwidth */
/* First, get available clock cycles */
dwAvailableVMCBandwidth = GetAvailableBandwidth();

switch (wBitsPerPixel)
{
      case 4:
         lStreamPixelRateMax = (vmcLONG)dwAvailableVMCBandwidth * 8;
         break;
      case 8:
         lStreamPixelRateMax = (vmcLONG)dwAvailableVMCBandwidth * 4;
         break;

      case 16:
         lStreamPixelRateMax = (vmcLONG)dwAvailableVMCBandwidth * 2;
         break;

      case 24:
      case 32:
         lStreamPixelRateMax = (vmcLONG)dwAvailableVMCBandwidth;
         break;

      default:
         return VMCERR_INVALID_STREAM_FORMAT;
   }

if ((lSourcePixelRateMax < lDestPixelRateMax) &&
    (lSourcePixelRateMax < lStreamPixelRateMax) )
{
      ValidateFormatParams.dwPeakBandwidth   = (vmcWORD)lSourcePixelRateMax;
}
else if ((lStreamPixelRateMax < lDestPixelRateMax) &&
         (lStreamPixelRateMax < lSourcePixelRateMax) )
{
      ValidateFormatParams.dwPeakBandwidth   = lStreamPixelRateMax;
}
else if ((lDestPixelRateMax < lStreamPixelRateMax) &&
         lDestPixelRateMax < lSourcePixelRateMax) )
{
      ValidateFormatParams.dwPeakBandwidth   = lDestPixelRateMax;
}

/*============================================================*/
/*                                                            */
/* Validate horizontal scale factors between source and       */
/* destination. The achievable size may be limited by source  */
/* and destination scaling capabilities and bandwidth         */
/* availability on VMC or within devices.                */
```

```
/*                                                  */
* Request Source to Verify Stream Configuration entry          */
/* and the the destination                              */
/*                                                  */
/*==============================================================*/
Ret = ValidateHorizontalScaling(hTransmitter, &ValidateFormatParams);
if ( (Ret != VMCERR_NONE) && (Ret != VMCERR_BANDWIDTH_LIMITED) )
{
   return vmcFALSE;
}

Ret = ValidateHorizontalScaling(hReceiver, &ValidateFormatParams);
if ( (Ret != VMCERR_NONE) && (Ret != VMCERR_BANDWIDTH_LIMITED) )
{
   return vmcFALSE;
}

/* Get destination width actually achieved. This is the maximum possible  */
lDestWidth = (vmcLONG) (lprDestRect->iRight - lprDestRect->iLeft);

if ( (lDestWidth != (vmcLONG)iRequestedWidth) || (Ret == VMCERR_BANDWIDTH_LIMITED) )
{

   /*=============================================================================*/
   /*                                                                             */
   /* Requested stream width was  not achieved, need to find if we can achieve a more */
   /*  exact match.        */
    /* Required Width will not be aligned if we are bandwdith limited or the scaling */
    /*  capabilities are not  */
    /* accurate enough for the required width.                                     */
    /*                                                                             */
    /* lDestWidth is set to the maximum achieveable but requested destination size  */
    /* has not been realised  */
    /* need to adjust source scale factor so that stream and destination scaling */
    /*  capabilities are aligned.   */
    /*                                                                             */
    /* Request destination to provide maximum image width it can achieve,        */
    /* based on limited source output  */
    /* Adjust scale factors so that scaling is increased (if possible) at destination  */
    /*                                                                             */
    /*=============================================================================*/

      /* Align stream width based on  destination scaler capabilities */
      AdjustHorizontalScaleFactors(lpDestScalerCaps,
                           lpSourceScalerCaps,
                           lprSrcRect,
                           lprStreamRect,
                           lprDestRect,
                           (vmcLONG)iRequestedWidth);

      /* Request Source and destination to re-evaluate horizontal scale factors */
       Ret = ValidateHorizontalScaling(hTransmitter, &ValidateFormatParams);
      if ( (Ret != VMCERR_NONE) && (Ret != VMCERR_BANDWIDTH_LIMITED) )
      {
         return vmcFALSE;
      }

       Ret = ValidateHorizontalScaling(hReceiver, &ValidateFormatParams);
      if ( (Ret != VMCERR_NONE) && (Ret != VMCERR_BANDWIDTH_LIMITED) )
      {
         return vmcFALSE;
      }
   }

   /* Pick up any adjustments to scaling factors made by Source and destination devices */

    *lpStreamConfig = ValidateFormatParams.StreamConfig;

   return vmcTRUE;

}
```

- Determining Vertical Scale Bands

The following illustrates how a source type driver should determine the number of vertical scale bands in response to the DRV_VMC_QUERY_NUM_SCALE_BANDS message.

```
/* calculate number of vertical scale bands to be supported */
VMC_ERR  GetNumMGScalerBands ( LPQUERY_NUM_SCALE_BAND_PARAMS lpScaleParams )
{
   vmcLONG  lSourceHeight, lStreamHeight, lScaleFactor;
   vmcLONG  lIntegerFactor;

   /* Calculate number of vertical scale bands */
   lStreamHeight = (vmcLONG)(lpScaleParams->rStreamRect.iBottom -
                   lpScaleParams->rStreamRect.iTop);
   lSourceHeight = (vmcLONG)(lpScaleParams->rSourceRect.iBottom -
                   lpScaleParams ->rSourceRect.iTop);

   /* calculate Vertical scale factor and restrict if necessary */
   lScaleFactor = ((lStreamHeight * VMC_SCALE_FACTOR)/lSourceHeight);
   if (lScaleFactor >  MAX_Y_SCALE_FACTOR)
   {
      lScaleFactor =  MAX_Y_SCALE_FACTOR;
   }

   /* calculate number of bands */
   lIntegerFactor = (lScaleFactor - 1L) / VMC_SCALE_FACTOR;
   lpScaleParams->iNumBands = (vmcINT)(lIntegerFactor) + 1;

   return VMCERR_NONE;
}
```

- Source Device Scale Factor Validation

The following example illustrates how a source type driver would validate its scale factors in response to a DRV_VMC_VALIDATE_HORIZONTAL_SCALING message received for the Stream Manager. A similar method is used to respond to the DRV_VMC_VALIDATE_VERTICAL_SCALING message except that no bandwidth restriction checks are required.

```
VMC_ERR  ValidateSourceHorizontalScaling(LPVALIDATE_SCALING_PARAMS   lpValidateParams)
{

   vmcLONG                         lSourceHeight;
   vmcLONG                         lSourceWidth;
   vmcLONG                    lStreamWidth, lStreamHeight;
   vmcWORD                         wBytesPerPixel;
   double                     dStreamPeakPixelRate;
   vmcLONG                         lSourceXScaleFactor, lSourceYScaleFactor;
   double                          dReduction;
   vmcLONG                         lNewXFactor;
   LPVIDEO_SCALER_CAPS           lpSourceScalerCaps;
   LPVMC_RECTINFO             lprSourceRect, lprStreamRect;
   vmcDWORD                dwPeakBandwidth;
   vmcINT                     iBand;
   VMC_ERR                    Ret;


    /* Assume Success */
    Ret = VMCERR_NONE;

   /* Check we support requested stream format */

   switch (lpValidateParams->StreamConfig.dwStreamFormat)
   {
      case VMCRGB_332:
            wBytesPerPixel = 1;
            break;
```

```
      case VMCRGB_565:
            wBytesPerPixel = 2;
            break;

      case VMCRGB_888:
            wBytesPerPixel = 4;
            break;
      default:
            return VMCERR_INVALID_STREAM_FORMAT;
   }


   /* Check we support requested stream mode */
   switch (lpValidateParams->StreamConfig.wStreamMode)
   {
      case VSM_INTERLACED_VIDEO:
      case VSM_NON_INTERLACED_VIDEO_EVEN_FIELD:
      case VSM_NON_INTERLACED_VIDEO_ODD_FIELD:
         break;

      default:
         return VMCERR_INVALID_STREAM_FORMAT;
   }

   /* set up pointers to scaler capabilities */
   lpSourceScalerCaps  = &(lpValidateParams->SourceScalerMode);

   /* Get Bandwidth Limit - imposed by stream manager */
   dwPeakBandwidth = lpValidateParams->dwPeakBandwidth;

   /* Get Stream Scaling Factors */
   lprStreamRect  = &(lpValidateParams->StreamConfig.rStreamRect);
   lprSourceRect  = &(lpValidateParams->StreamConfig.rSourceRect);

   /* Get required scaling factors set up by Stream Manager */
   lSourceWidth  = (vmcLONG)(lprSourceRect->iRight - lprSourceRect->iLeft);
   lStreamWidth  = (vmcLONG)(lprStreamRect->iRight - lprStreamRect->iLeft);

   /* calculate bandwidth for this format and size */
   lSourceHeight =  (vmcLONG)(lprSourceRect->iBottom - lprSourceRect->iTop);
   lStreamHeight =  (vmcLONG)(lprStreamRect->iBottom - lprStreamRect->iTop);
   lSourceYScaleFactor = ( lStreamHeight * VMC_SCALE_FACTOR) / lSourceHeight;

    /* Set stream width based on our capabilities */
   SetWidthFromSource(  lSourceWidth,
                  lStreamWidth,
                  vmcFALSE,
                  lpSourceScalerCaps,
                  &lStreamWidth,
                  &lSourceXScaleFactor);

   /* calculate bandwidth for this format and size */
   lSourceYScaleFactor = ( lStreamHeight * VMC_SCALE_FACTOR) / lSourceHeight;
   iBand =  (vmcINT)(((lSourceYScaleFactor - 1L) / VMC_SCALE_FACTOR) + 1);


    dStreamPeakPixelRate = (double)(((double)lSourceWidth * (double)1000000L) /
(double)64L);
    dStreamPeakPixelRate = (double)((double)dStreamPeakPixelRate  *
(double)(lSourceXScaleFactor)) / VMC_SCALE_FACTOR;
    dStreamPeakPixelRate *= (double)iBand;

   /* Verify that the Bandwidth is within the supplied Limit */
   if ((vmcLONG)dStreamPeakPixelRate > (vmcLONG)dwPeakBandwidth)
   {
      /*=========================================================================*/
      /* Bandwidth Limited.                                                      */
      /* calculate the maximum width we can achieve for current vertical scale factor */
      /* This must always be on a boundary that we can achieve                   */
      /*                                                                         */
      /*=========================================================================*/
```

```
      dReduction = ( (((double)dwPeakBandwidth +1) * (double)VMC_SCALE_FACTOR) /
                  (double)dStreamPeakPixelRate);

      /* Can Only reduce X for this vertical scale band */
      /* define a new maximum image width based on limited bandwidth */
      lNewXFactor  = (lSourceXScaleFactor * (vmcLONG)dReduction) / VMC_SCALE_FACTOR;
      lStreamWidth = ((vmcLONG)((vmcLONG)lSourceWidth * (vmcLONG)lNewXFactor) /
                     VMC_SCALE_FACTOR);

       /* Set stream width based on our capabilities */
       /* As a transmitter, we must ensure that stream width is achievable with our scaler
capabilities */
       /* i.e if we have an integer scaler, stream width must be integer multiple of
source width */
      SetWidthFromSource(  lSourceWidth,
                     lStreamWidth,
                     vmcTRUE,
                     lpSourceScalerCaps,
                     &lStreamWidth,
                     &lSourceXScaleFactor);

      /* Set new bandwidth consumed by this configuration */
      /* in Pixels per second */
       dStreamPeakPixelRate = (double)(((double)lSourceWidth * (double)1000000L) /
                              (double)dwLineRate);
       dStreamPeakPixelRate = (double)((double)dStreamPeakPixelRate  *
                              (double)(lSourceXScaleFactor)) / VMC_SCALE_FACTOR;
       dStreamPeakPixelRate *= (double)iBand;

       /* Indicate we are bandwidth limited */
       Ret = VMCERR_BANDWIDTH_LIMITED;
   }

   /* Set stream size according to achievable scale factors */
   lprStreamRect->iRight = lprStreamRect->iLeft + (vmcINT)lStreamWidth;

   /* set up Pixel Rate for this stream */
    lpValidateParams->StreamConfig.dwStreamPeakPixelRate = (vmcDWORD)dStreamPeakPixelRate;

   return Ret;
}


vmcVOID  SetWidthFromSource(  vmcLONG lSourceWidth,
                     vmcLONG lStreamWidth,
                     vmcBOOLEAN  bLimited,
                     LPVIDEO_SCALER_CAPS lpSourceScalerCaps,
                     vmcLPTR lpAdjustedStreamWidth,
                     vmcLPTR  lpAdjustedXScaleFactor)
{
   vmcLONG  lSourceXScaleFactor, lIntegerFactor;
   vmcLONG  lImageWidth;

   lSourceXScaleFactor = ( lStreamWidth * VMC_SCALE_FACTOR) / lSourceWidth;
    switch (lpSourceScalerCaps->wXScaleGranularity)
    {
      case SG_CAN_SCALE_PRECISELY:
         lImageWidth = lStreamWidth;
         break;

      case SG_CAN_SCALE_IN_INTEGERS:
         /* Adjust horizontal scale factor to be integer multiple */
         if (lSourceXScaleFactor < VMC_SCALE_FACTOR)
         {
            /* scaling down */
            if (bLimited == vmcTRUE)
            {
            }
            lIntegerFactor = (lSourceWidth * VMC_SCALE_FACTOR) / lStreamWidth;
            lIntegerFactor = ((lIntegerFactor - 1L) / VMC_SCALE_FACTOR) + 1L;
            lSourceXScaleFactor = VMC_SCALE_FACTOR / lIntegerFactor;
            lImageWidth = lSourceWidth / lIntegerFactor;
```

```
         }
         else
         {
            /* scaling up - but bandwidth limited - do not overscale    */
            /*  underscale if necessary */
            if (bLimited == vmcTRUE)
            {
               lIntegerFactor = ((lSourceXScaleFactor)/VMC_SCALE_FACTOR);
            }
            else
            {
               lIntegerFactor = ((lSourceXScaleFactor - 1L)/VMC_SCALE_FACTOR) + 1L;
            }
            lSourceXScaleFactor = lIntegerFactor * VMC_SCALE_FACTOR;
            lImageWidth = lSourceWidth * lIntegerFactor;
         }
         break;

        default:
         lImageWidth = lSourceWidth;
         lSourceXScaleFactor = VMC_SCALE_FACTOR;
         break;
   }


   /* Keep Horizontal scale factors within Capabilities */
   if (lSourceXScaleFactor > (vmcLONG)lpSourceScalerCaps->wXScaleFactorMax)
   {
      lSourceXScaleFactor = (vmcLONG)lpSourceScalerCaps->wXScaleFactorMax;
   }
   else if (lSourceXScaleFactor < (vmcLONG)lpSourceScalerCaps->wXScaleFactorMin)
   {
      lSourceXScaleFactor = (vmcLONG)lpSourceScalerCaps->wXScaleFactorMin;
   }


   if (lImageWidth > (vmcLONG)MAX_IMAGE_WIDTH)
   {
      /* Line Length Limited */
      lStreamWidth = MAX_IMAGE_WIDTH;
      SetWidthFromSource(  (vmcLONG) lSourceWidth,
                     (vmcLONG) lStreamWidth,
                     (vmcBOOLEAN) bLimited,
                     (LPVIDEO_SCALER_CAPS) lpSourceScalerCaps,
                     (vmcLPTR) &lStreamWidth,
                     (vmcLPTR) &lSourceXScaleFactor);
      lImageWidth =  lStreamWidth;
   }


   *lpAdjustedStreamWidth = lImageWidth;
   *lpAdjustedXScaleFactor= lSourceXScaleFactor;
}
```

- Destination Device Scale Factor Validation.

```
VMC_ERR ValidateDestHorizontalScaling( LPVALIDATE_SCALING_PARAMS lpValidateParams)
{

   vmcINT               iStreamMode;
   LPVIDEO_SCALER_CAPS     lpScalerCaps;
   LPVIDEO_STREAM_CONFIG   lpStreamConfig;
   LPVMC_RECTINFO       lprDestRect, lprSrcRect;
   vmcDWORD          dwPeakBandwidth;
   vmcLONG              lDestHeight, lSourceHeight;
   vmcLONG              lDestWidth, lSourceWidth;
   vmcWORD              wBytesPerPixel;
   double               dBandWidth;
   vmcLONG              lImageWidth;
   double               dReduction;
```

```
vmcLONG              lNewXFactor, lNumBands;
vmcLONG              lDestXScaleFactor, lDestYScaleFactor;
vmcINT               iBand;
VMC_ERR              Ret;
HWND              hDeskTopWnd;
vmcWORD              wDisplayBitsPerPixel;


Ret = VMCERR_NONE;

/* Retrieve the Stream Mode */
lpStreamConfig    = &(lpValidateParams->StreamConfig);
lpScalerCaps   = &(lpValidateParams->DestScalerMode);
   iStreamMode       = lpStreamConfig->wStreamMode;

/* get scaling parameters */
lprSrcRect     = &(lpStreamConfig->rStreamRect);
lprDestRect    = &(lpStreamConfig->rDestRect);


/* Get required scaling factors */
lDestHeight = (vmcLONG)(lprDestRect->iBottom - lprDestRect->iTop);
lSourceHeight = (vmcLONG)(lprSrcRect->iBottom - lprSrcRect->iTop);
lDestWidth   = (vmcLONG)(lprDestRect->iRight - lprDestRect->iLeft);
lSourceWidth = (vmcLONG)(lprSrcRect->iRight - lprSrcRect->iLeft);


/* Get the display width and height - use the context of desktop window */
hDeskTopWnd = GetDesktopWindow();
if (hDeskTopWnd != (HWND)NULL)
{
   HDC   hDC;

   /* Get the device context */
   hDC = GetDC(hDeskTopWnd);

   /* Get the colour depth */
   wDisplayBitsPerPixel = (vmcWORD)GetDeviceCaps(hDC, BITSPIXEL);

   /* Release the context */
   ReleaseDC(hDeskTopWnd, hDC);
}

/* calculate available VRAM bandwidth */
 switch (wDisplayBitsPerPixel)
 {
   case 8:
      dwPeakBandwidth = MAX_VRAM_BANDWIDTH * 4;
      break;

   case 16:
      dwPeakBandwidth = MAX_VRAM_BANDWIDTH * 2;
      break;

   default:
      dwPeakBandwidth = MAX_VRAM_BANDWIDTH;
      break;
 }


/* validate the stream format */
switch (lpStreamConfig->dwStreamFormat)
{
   case VMCRGB_332:
      wBytesPerPixel = 1;
      break;

   case VMCRGB_565:
      wBytesPerPixel = 2;
      break;
```

```
   case VMCRGB_888:
      wBytesPerPixel = 4;
      break;
   default:
      return VMCERR_INVALID_STREAM_FORMAT;
}



/*
/*
 * Keep Horizontal scale factors within Capabilities
 * in Non-Interlaced mode scaling optionas are 1:1, 1:2 and 2:1
 * in interlaced mode, scaling is always 1:1
 */

lDestXScaleFactor = (lDestWidth * VMC_SCALE_FACTOR) / lSourceWidth;
/* Adjust horizontal scale factor to be integer multiple */
if (lDestXScaleFactor < VMC_SCALE_FACTOR)
{
   /* scaling down -- will overscale if required factor  not integer multiple*/
   lNumBands = (lSourceWidth * VMC_SCALE_FACTOR) / lDestWidth;
   lNumBands = ((lNumBands - 1L) / VMC_SCALE_FACTOR);
   lDestXScaleFactor = VMC_SCALE_FACTOR / lNumBands;
}
else
{
   /* scaling up - if not an exact multiple, always over scale */
   lNumBands = ((lDestXScaleFactor - 1L)/VMC_SCALE_FACTOR) + 1L;
   lDestXScaleFactor = lNumBands * VMC_SCALE_FACTOR;
}

/* Keep horizontal scaling within capabilities */
if (lDestXScaleFactor >= (vmcLONG)lpScalerCaps->wXScaleFactorMax)
{
   /* clamp scale factor to maximum achieveable */
   lDestXScaleFactor = (vmcLONG)lpScalerCaps->wXScaleFactorMax;
}
else if (lDestXScaleFactor < (vmcLONG)lpScalerCaps->wXScaleFactorMin)
{
   /* clamp scale factor to minimum achieveable */
   lDestXScaleFactor <= (vmcLONG)lpScalerCaps->wXScaleFactorMin;
}

lImageWidth = ( (vmcLONG)lSourceWidth * (vmcLONG)(lDestXScaleFactor));
lImageWidth = ( (vmcLONG)lImageWidth / VMC_SCALE_FACTOR);
if (lImageWidth > (vmcLONG)MAX_IMAGE_WIDTH)
{
   /* Line Length Limited */
   lDestXScaleFactor = ((vmcLONG)MAX_IMAGE_WIDTH * VMC_SCALE_FACTOR) /
                  (vmcLONG)lSourceWidth;
   lImageWidth = ( ((vmcLONG)lSourceWidth * (vmcLONG)(lDestXScaleFactor)) /
                  VMC_SCALE_FACTOR );
}



/* calculate Vertical scale factor  */
lDestYScaleFactor = (lDestHeight * VMC_SCALE_FACTOR) / lSourceHeight;
lNumBands = (lDestYScaleFactor - 1L)/VMC_SCALE_FACTOR;
iBand = (vmcWORD)(lNumBands) + 1;


   /* calculate bandwidth for this format and size */
   /* Stream Pixel Rate in Pixels per second */
 dBandWidth = ((double)lpStreamConfig->dwStreamPeakPixelRate  *
               (double)(lDestXScaleFactor)) /
               (double)VMC_SCALE_FACTOR;
dBandWidth = dBandWidth * (double)iBand;


/* Verify that the Bandwidth is within the supplied Limit */
```

```
   if ((double)dBandWidth > (double)dwPeakBandwidth)
   {
      /* calculate the maximum we can achieve */
      dReduction = ( (((double)dwPeakBandwidth +1) * (double)VMC_SCALE_FACTOR) /
               (double)dBandWidth);

      /*=============================================================================*/
      /* Can Only reduce X for this vertical scale band                              */
      /* This new width defines a new maximum image width for the stream as a whole */
      /* This is not guaranteed to be aligned with source and destination scaling
capabilities      */
      /* The Stream Manager needs to recalculate sizes based on this limitation  */
      /*                                                                             */
   /*=============================================================================*/
      lNewXFactor  = (lDestXScaleFactor * (vmcLONG)dReduction) / VMC_SCALE_FACTOR;
      lImageWidth = ((vmcLONG)((vmcLONG)lSourceWidth * (vmcLONG)lNewXFactor) /
                     VMC_SCALE_FACTOR);

      /* Indicate to the stream manager that we are bandwidth limited */
      Ret = VMCERR_BANDWIDTH_LIMITED;
   }


   /* return achieveable destination width to Stream Manager */
   lprDestRect->iRight = lprDestRect->iLeft + (vmcINT)lImageWidth;

   return Ret;
}
```

- Adjusting Scale Fcators

An example of adjusting horizontal scale factors is illustrated below. Adjusting Vertical scale factors would use an identical method.

```
vmcBOOLEAN  AdjustHorizontalScaleFactors(    LPVIDEO_SCALER_CAPS  lpDestScalerCaps,
                                LPVIDEO_SCALER_CAPS  lpSrcScalerCaps,
                                LPVMC_RECTINFO       lprSrcRect,
                                LPVMC_RECTINFO       lprStreamRect,
                                LPVMC_RECTINFO       lprDestRect,
                                vmcLONG              lRequestedWidth)
{
   vmcLONG     lStreamWidth, lAchievedWidth, lDestWidth, lSourceWidth;
   vmcWORD     wSourceGranularity, wDestGranularity;
   vmcBOOLEAN  bAdjusted;


   /* Dest specifies an image size that is, if possible, equal to or greater than the
required destination width */
    /* If dest is bandwidth limited however, the achieved width may be less than the
required Width */
    lSourceWidth  = (vmcLONG) (lprSrcRect->iRight - lprSrcRect->iLeft);
    lStreamWidth   = (vmcLONG) (lprStreamRect->iRight - lprStreamRect->iLeft);
    lAchievedWidth = (vmcLONG) (lprDestRect->iRight - lprDestRect->iLeft);
    if (lAchievedWidth > lRequestedWidth)
    {
      lDestWidth = lRequestedWidth;
    }
    else
    {
      lDestWidth = lAchievedWidth;
    }

   /* Get scaler capabilities */
   wSourceGranularity = lpSrcScalerCaps->wXScaleGranularity;
   wDestGranularity   = lpDestScalerCaps->wXScaleGranularity;

   /* assume adjustment will be made */
   bAdjusted = vmcTRUE;

   if ( (wSourceGranularity == SG_CAN_SCALE_IN_INTEGERS) &&
         (wDestGranularity == SG_CAN_SCALE_IN_INTEGERS) )
   {
```

```
      /* Make Stream Width multiple of Source Width */
      SetDimensionFromSourceAndDest(   lpDestScalerCaps->wXScaleFactorMax,
                              lpSrcScalerCaps->wXScaleFactorMin,
                              lSourceWidth, lStreamWidth,
                              lDestWidth,
                              &lStreamWidth,
                              &lDestWidth);

   }
   else if ( (wSourceGranularity == SG_CAN_SCALE_PRECISELY) &&
             (wDestGranularity == SG_CAN_SCALE_IN_INTEGERS) )
   {
      /* Adjusted Stream Width to be Integer multiple of destination */
      SetDimensionFromDest(lDestWidth, lStreamWidth, &lStreamWidth);
   }
   else if ( (wSourceGranularity == SG_CAN_SCALE_IN_INTEGERS) &&
             (wDestGranularity == SG_CAN_SCALE_PRECISELY) )
   {
      /* Adjust Stream Width to be integer multiple of source */
      SetDimensionFromSource(lSourceWidth, lStreamWidth, &lStreamWidth);
   }
   else if (   ((wSourceGranularity == SG_CAN_SCALE_IN_INTEGERS) &&
             (wDestGranularity == SG_CANNOT_SCALE)) ||
             ((wSourceGranularity == SG_CAN_SCALE_PRECISELY) &&
             (wDestGranularity == SG_CANNOT_SCALE)) )
   {
      /* Clamp destination size to that achieved by source device */
      lStreamWidth = lDestWidth;
   }
   else if (   ((wSourceGranularity == SG_CANNOT_SCALE) &&
                (wDestGranularity == SG_CAN_SCALE_IN_INTEGERS)) ||
             ((wSourceGranularity == SG_CANNOT_SCALE) &&
             (wDestGranularity == SG_CAN_SCALE_PRECISELY)) )
   {
      /* Clamp Stream size to that achieved by source device */
      lStreamWidth = lSourceWidth;
   }
   else
   {
      bAdjusted = vmcFALSE;
   }


   /* Set any adjusted scale factors */
   lprDestRect->iRight = lprDestRect->iLeft + (vmcINT)lDestWidth;
   lprStreamRect->iRight = lprStreamRect->iLeft + (vmcINT)lStreamWidth;

    return bAdjusted;
}
```

Appendix E. VMC Structure Definitions

VMC Spec. Version Number

```
#define VMC_VERSION_NUMBER(0x0100)
```

VMC Error Packet

VESA Defined Error Codes

```
#define VMCERR_NONE                                0
#define VMCERR_UNSUPPORTED_MESSAGE                 1

#define VMCERR_NO_STREAMS_AVAILABLE                2
#define VMCERR_INVALID_STREAM_TYPE                 3
#define VMCERR_INVALID_STREAM_OBJ                  4
#define VMCERR_STREAM_ENABLED                      5
#define VMCERR_STREAM_NOT_ENABLED                  6
#define VMCERR_STREAM_NOT_CONFIGURED               7
#define VMCERR_INVALID_CONFIGURATION               8
#define VMCERR_INVALID_STREAM_FORMAT               9
#define VMCERR_INVALID_SCALE_FACTORS               10
#define VMCERR_TRANSMITTER_ALREADY_ALLOCATED       11
#define VMCERR_INSUFFICIENT_BANDWIDTH              12
#define VMCERR_NO_SUCH_DEVICE                      13
#define VMCERR_NON_STREAM_WRITE_NOT_SUPPORTED      14
#define VMCERR_NON_STREAM_READ_NOT_SUPPORTED       15

#define VMCERR_INVALID_DEVICE_OBJ                  16
#define VMCERR_CANNOT_ALLOCATE_DEVICE_CONTEXT      17
#define VMCERR_INVALID_CONTEXT_TYPE                18
#define VMCERR_INVALID_DEVICE_INSTANCE             19
#define VMCERR_CONTEXT_STILL_ASSOCIATED            20
#define VMCERR_INVALID_SCALER_MODE                 21
#define VMCERR_INVALID_STREAM_MODE                 22
#define VMCERR_STREAM_ALREADY_DEALLOCATED          23
#define VMCERR_INVALID_DEVICE_MODE                 24
#define VMCERR_CLIPPING_RECTANGLES_USED            25
#define VMCERR_BANDWIDTH_LIMITED                   26

#define VMCERR_FAILED_TO_ALLOCATE_MEMORY           27
#define VMCERR_FAILED_TO_FREE_MEMORY               28
#define VMCERR_FAILED_TO_LOCK_MEMORY               29
#define VMCERR_FAILED_TO_UNLOCK_MEMORY             30

#define VMCERR_BAD_PARAMETERS                      31
#define VMCERR_BAD_RESULT                          32
#define VMCERR_INVALID_DRIVER_HANDLE               33
#define VMCERR_MESSAGE_NOT_RECIEVED                34

#define VMCERR_FAILED_TO_OPEN_DRIVER               35
#define VMCERR_FAILED_TO_CLOSE_DRIVER              36

#define VMCERR_DEVICES_NOT_AVAILABLE               37

#define VMCERR_STREAM_STILL_ASSOCIATED             38

#define VMCERR_MAX_ERRORS                          38/* Number of VESA defined error codes */
```

Vendor Defined Error Codes

```
#define VMCERR_USER                                1000 /* Vendor defined error codes start here */

typedef struct
{
```

```
    vmcDWORD    dwSize;
    vmcHANDLE   hMessageOriginator;
    vmcWORD     wErrorCode;
} VMC_ERROR_PARAMS;
typedef VMC_ERROR_PARAMS vmcFAR *LPVMC_ERROR_PARAMS;
```

Member	Description
dwSize	Size (in bytes) of this structure
dwMessageOriginator	handle of error message originator
wErrorCode	Returned error code

VMC DEVICECAPS

VMC Device Mode Values

```
#define VMC_CAN_TRANSMIT      (0x0001)
#define VMC_CAN_RECEIVE   (0x0002)
```

VMC Stream Width Values

```
#define VMC_8BIT_DEVICE          0x0001)
#define VMC_16BIT_DEVICE   (0x0002)
#define VMC_32BIT_DEVICE   (0x0004)

typedef struct tagVMCDEVICECAPS
{
    vmcDWORD    dwSize;
    vmcDWORD    dwStructRevID;
    vmcWORD     wVMCDevModes;
    vmcDWORD    dwMaxRate;
    vmcWORD         wVMCDevID;
    vmcWORD     wVMCDevBufferSize;
    vmcWORD     wVMCDevWidth;
    vmcDWORD    dwVMCDevMinGra nt;
    vmcDWORD    dwVMCDevMaxLatency;
} VMC_DEVICECAPS;
typedef VMC_DEVICECAPS     vmcFAR *LPVMC_DEVICECAPS;
```

Member	Description
dwSize	Size (in bytes) of this structure
dwStructRevID	structure revision = 0x0100
wVMCDevID	VMC device ID - value read back from the device VMC ID register. ID's are allocated by the assigned VMC controller following a reset.
dwMaxRate	Maximum rate at which device can source/sink data to/fom VMC
wVMCDevModes	Device modes - defines if the device can act as transmitter, receiver or both. Takes a bit mask of VMC_CAN_TRANSMIT and VMC_CAN_RECEIVE.
wVMCDevBufferSize	device FIFO size (in DWORDS)
wVMCDevWidth	device VMC bus width. either VMC_8BIT_DEVICE, VMC_16BIT_DEVICE or VMC_32BIT_DEVICE.
dwVMCDevMinGrant	specifies (in bus clock periods) the minimum sustained grant time required by the device.
dwVMCDevMaxLatency	specifies (in bus clock periods) the maximum latency that can be tolerated by the device once it releases the token.

VMC BANDWIDTH INFO

VMC CLOCK RATE DEFINITIONS

```
#define VMC_33MHZ_CLOCK_RATE    (0x0021)
```

```
#define VMC_25MHZ_CLOCK_RATE    (0x0019)

#define MAX_MTRT      (4)
```

VMC_BANDWIDTHINFO

```
typedef struct tagVMC_BANDWIDTHINFO
{
    vmcDWORD    dwSize;
    vmcWORD     wChannelClockRate;
    vmcDWORD    dwMTRT;
    vmcDWORD    dwTotalBandwidth;
    vmcDWORD    dwFreeBandwidth;
    vmcDWORD    dwMaxAllocation;
} VMC_BANDWIDTHINFO;
typedef VMC_BANDWIDTHINFO     vmcFAR *LPVMC_BANDWIDTHINFO;
```

Member	Description
dwSize	Size (in bytes) of this structure
dwChannelClockRate	VMC clock rate [VMC_33MHZ_CLOCK | VMC_25MHZ_CLOCK]
dwMTRT	MTRT in uS.
dwTotal Bandwidth	Total bandwidth in Clock cycles (dwMTRT * dwChannelClockRate)
dwFreeBandwidth	Free number of clock cycles
dwMaxAllocation	Maximum clock cycles that can be allocated to a device.

VMC_DEVICE_INSTANCE CAPABILITIES

VMC Device Type Definitions

```
#define DT_VMC_VIDEO_DEVICE  (0x00000001L)
```

VMC Video Device Context Type Definitions

```
#define CT_GRAPHIC_SINK         (0x00000001L)
#define CT_GRAPHIC_SOURCE       (0x00000002L)
#define CT_VIDEO_SOURCE         (0x00000004L)
#define CT_VIDEO_SINK           (0x00000008L)
```

VMC Context Sub type definitions

```
#define ST_NULL                 (0x00000000L)
```

VMC Graphic System Specific Context Subtype Definitions

```
#define ST_ON_SCREEN        (0x00000001L)
```

VMC Video Source Specific Context Subtype Definitions

```
#define ST_VIDEO_DECODER                    (0x00000001L)
#define ST_CODEC     _SOURCE                (0x00000002L)
#define ST_FX_PROCESSOR_SOURCE    (0x00000004L)
```

VMC Video Sink Specific Context Subtype Definitions

```
#define ST_VIDEO_ENCODER          (0x00010000L)
#define ST_CODEC     _SINK              (0x00020000L)
#define ST_FX_PROCESSOR_SINK    (0x00040000L)

typedef struct
{
    vmcDWORD    dwSize;
    vmcDWORD    dwStructRevID;
    vmcCHAR     szVendor[MAX_VENDOR_LEN + 1];
    vmcCHAR     szProductName[MAX_STRING_LEN +1];
    vmcCHAR     szBoardInfo[MAX_STRING_LEN+1];
    vmcDWORD    dwBoardInfo;
    vmcDWORD    dwRevID;
```

```
    vmcWORD     wBoardInstance;
    vmcWORD     wVMCDeviceID;
    vmcDWORD    dwVMCDeviceType;
    vmcDWORD    dwVMCContextCaps;
    vmcDWORD    dwContextSubTypeCaps;
} VESA_DEVICECAPS;
typedef VESA_DEVICECAPS vmcFAR  *LPVESA_DEVICECAPS;
```

Member	Description
dwSize	Size (in bytes) of this structure
dwStructRevID	structure revision = 0x0100
szVendor	NULL terminated string defining name of vendor
szProduct Name	NULL terminated string defining product associated with device
szBoardInfo	NULL terminated string defining additional Vendor board information
dwBoardInfo	Vendor defined board information
wBoardInstance	Board Instance associated with device
dwRevId	vendor specific device revision ID
wVMCDeviceID	VMC device id allocated to this device on bus initialization
dwVMCDeviceType	VESA defined device Type
dwVMCContextCaps	VESA defined context capabilites supported by this device
dwContextSubTypeCaps	VESA defined context subtypes

DEVICE CONTEXT IDENTIFIER

```
typedef struct
{
    vmcDWORD    dwSize;
    vmcDWORD    dwContextType;
    vmcDWORD    dwContextSubType;
    vmcHANDLE   hContext;
    vmcHANDLE   hDriver;
} DEVICE_OBJ;
typedef DEVICE_OBJ vmcFAR *LPDEVICE_OBJ;
```

Member	Description
dwSize	Size (in bytes) of this structure
dwContextType	type of context - [CT_VIDEO_SOURCE | CT_VIDEO_SINK | CT_GRAPHIC_SINK | CT_GRAPHIC_SOURCE]
dwContextSubtype	context subtype
hContext	unique Context handle, allocated by device type drivers.
hDriver	Handle to Device Type Driver used to manage the specified context.

DEVICE CONTEXT CAPABILITIES

```
/*  Video Stream Mode Definitions  */
#define VSM_INTERLACED_VIDEO                        (0x0001)
#define VSM_NON_INTERLACED_VIDEO_EVEN_FIELD   (0x0002)
#define VSM_NON_INTERLACED_VIDEO_ODD_FIELD        (0x0004)


typedef struct
{
    vmcDWORD    dwSize;
    vmcWORD     wStreamModes;
}  VIDEO_CONTEXT_CAPS;
typedef VIDEO_CONTEXT_CAPS vmcFAR *LPVIDEO_CONTEXT_CAPS;
```

Member	Description
dwSize	Size (in bytes) of this structure

wStreamModes	bit mask of defined Stream modes supported by the context

VIDEO STREAM MODE CAPABILITIES

```
/*    Video Stream Format Definitions      */
#define VMCRGB_8_INDEXED       (0x00000001L)
#define VMCRGB_768_LUT         (0x00000002L)
#define VMCGREYSCALE_8     (0x00000004L)
#define VMCRGB_332             (0x00000008L)
#define VMCRGB_565             (0x00000010L)
#define VMCRGB_888             (0x00000020L)
#define VMCRGB_A888                (0x00000040L)
#define VMC_RESERVED       (0x00000080L)
#define VMCYUV_422             (0x00000100L)
#define VMCYUV_411             (0x00000200L)
#define VMCYUV_420             (0x00000400L)


/*    Supported Clip Mode Capabilities     */
#define CM_NOT_SUPPORTED     (0x00000001L)
#define CM_CLIPRECTS             (0x00000002L)

/* Supported Scaler Mode capabilities  */
#define SM_NOT_SUPPORTED               (0x00000001L)
#define SM_PROGRESSIVE_SCAN            (0x00000002L)
#define SM_INTERLACED                  (0x00000004L)
#define SM_NON_INTERLACED          (0x00000008L)


typedef struct
{
    vmcDWORD    dwSize;
    vmcDWORD    dwStreamFormats;
    vmcDWORD    dwClipCaps;
    vmcWORD     wClipRects;
    vmcDWORD    dwScalerModes;

}  VIDEO_STREAM_CAPS;
typedef VIDEO_STREAM_CAPS vmcFAR *LPVIDEO_STREAM_CAPS;
```

Member	Description
dwSize	Size (in bytes) of this structure
dwStreamFormat	bit mask of VMC Stream formats supported by this Stream Mode
dwScalerModes	bit mask of supported scaler modes for this stream mode
dwClipCaps	bit mask of supported context clip capabilities for this stream mode.
wCliprects	maximum number of clipping rectangles supported by the device in this stream mode.

VIDEO SCALER CAPABILITIES

```
/*    X & Y Scaling Mode definitions   */
#define HSC_CANNOT_SCALE_X                 (0x00000001L)
#define HSC_CAN_REPLICATE_DECIMATE_X       (0x00000002L)
#define HSC_CAN_INTERPOLATE_FILTER_X       (0x00000004L)

#define VSC_CANNOT_SCALE_Y                 (0x00010000L)
#define VSC_CAN_REPLICATE_DECIMATE_Y       (0x00020000L)
```

```
#define VSC_CAN_INTERPOLATE_FILTER_Y        (0x00040000L)

/* Scaling Granularity definitions */
#define SG_CANNOT_SCALE                     (0x0001)
#define SG_CAN_SCALE_PRECISELY                  (0x0002)
#define SG_CAN_SCALE_IN_INTEGERS            (0x0004)

typedef struct
{
    vmcDWORD    dwSize;
    vmcDWORD    dwScaleCaps;
    vmcWORD     wVStart;
    vmcWORD     wHStart;
    vmcWORD     wXScaleFactorMax;
    vmcWORD     wXScaleFactorMin;
    vmcWORD     wXScaleGranularity;
    vmcWORD     wYScaleFactorMax;
    vmcWORD     wYScaleFactorMin;
    vmcWORD     wYScaleGranularity;
    vmcDWORD    dwPeakInputPixelRate;
    vmcDWORD    dwPeakOutputPixelRate;
} VIDEO_SCALER_CAPS;
typedef VIDEO_SCALER_CAPS vmcFAR *LPVIDEO_SCALER_CAPS;
```

Member	Description
dwSize	Size (in bytes) of this structure
wVStart	number of lines required for vertical scaler initialisation - normally applies to receivers - should be set null if not required.
wHStart	number of pixels required for horizontal scaler initialisation - normally applies to receivers - should be set null if not required.
dwScaleCaps	bit mask of X & Y capabilities defining the scaler mode - one X and one Y scale capability must be set to define the scaler mode.
wXScaleFactorMax	Maximum Horizontal Scale factor supported by context, in percent (100% = 1:1).
wXScaleFactorMin	Minimum Horizontal Scale factor supported by context, in percent (100% = 1:1)
wXScaleGranularity	Horizontal Scaler increment/decrement factor in %.
wYScaleFactorMax	Maximum Vertical Scale factor supported by context, in percent (100% = 1:1)
wYScaleFactorMin	Minimum Vertical Scale factor supported by context, in percent (100% = 1:1)
wYScaleGranularity	Vertical Scaler accuracy (Precisely = to 1% accuracy)
dwPeakInputPixelRate	Peak rate at which pixels enter device
dwPeakOutputPixelRate	Peak Rate at which pixels can leave device

VMC_CLIPINFO

```
typedef struct
{
    vmcDWORD            dwSize;
    RGNDATAHEADER       rdh;
    char                buffer[1];
}VMC_CLIPINFO;
typedef VMC_CLIPINFO vmcFAR *LPVMC_CLIPINFO;
```

Member	Description
dwSize	Size (in bytes) of this structure
rdh	DCI RGNDATA clipping info header.
buffer	array of clipping rectangles.

VMC STREAM Definitions

VMC Stream Types

#define VST_VIDEO_STREAM (0x0001)

Stream Object Definition

```
typedef struct
{
    vmcDWORD    dwSize;
    vmcDWORD    dwStreamType;
    vmcWORD     wStreamID;
    vmcHANDLE   hDriver;
} STREAM_OBJ;
typedef STREAM_OBJ vmcFAR *LPSTREAM_OBJ;
```

Member	Description
dwSize	Size (in bytes) of this structure
dwStreamType	Stream type - only VST_VIDEO_STREAM at present.
wStreamID	unique Stream ID - allocate by Stream Manager.
hDriver	handle to Stream Manager Driver used to manage this stream.

Stream Rectangle Definition

```
typedef struct tagVMCRECTINFO
{
    vmcDWORD    dwSize;
    vmcINT      iLeft;
    vmcINT      iTop;
    vmcINT      iRight;
    vmcINT      iBottom;
} VMC_RECTINFO;
typedef VMC_RECTINFO vmcFAR *LPVMC_RECTINFO;
```

Member	Description
dwSize	Size (in bytes) of this structure
iLeft	Specifies the X-coordinate of the upper left corner of a rectangle.
iTop	Specifies the Y-coordinate of the upper left corner of a rectangle.
iRight	specifies the X-coordinate of the lower right corner of a rectangle
iBottom	specifies the Y-Coordinate of the lower right corner of a rectangle

VMC Stream Configuration

```
typedef struct
{
    vmcDWORD        dwSize;
    vmcDWORD        dwStreamFormat;
    vmcWORD         wBitsPerPixel;
    vmcWORD         wStreamMode;
    vmcDWORD        dwSourceScaleMode;
    vmcDWORD        dwDestScaleMode;
    VMC_RECTINFO    rSourceRect;
    VMC_RECTINFO    rStreamRect;
    VMC_RECTINFO    rDestRect;
    vmcDWORD        dwStreamPeakPixelRate;
}  VIDEO_STREAM_CONFIG;
```

typedef VIDEO_STREAM_CONFIG vmcFAR *LPVIDEO_STREAM_CONFIG;

Member	Description
dwSize	Size (in bytes) of this structure
dwStreamFormat	Stream data format (one of the formats defined in the device stream capabilities)
wBitsPerPixel	Data format pixel depth
wStreamMode	VSM_INTERLACED, VSM_NON_INTERLACED_ODD_FIELD or VSM_NON_INTERLACED_EVEN_FIELD
dwSourceScaleMode	defines source scaler mode (one of the modes defined for the device stream capabilities)
dwDestScaleMode	defines destination scaler mode (one of the modes defined for the device stream capabilities)
rSourceRect	required Source image size
rStreamRect	required Stream Image size
rDestRect	required Destination Image size
dwStreamPeakPixelRate	peak peak rate available to a stream

Stream Bandwidth

```
typedef struct
{
    vmcDWORD    dwSize;
    vmcDWORD    dwSourcePixelRate;
    vmcDWORD    dwVMCPixelRate;
    vmcDWORD    dwDestPixelRate;
} STREAM_BANDWIDTHINFO;
typedef STREAM_BANDWIDTHINFO    vmcFAR *LPSTREAM_BANDWIDTHINFO;
```

Member	Description
dwSize	Size (in bytes) of this structure
dwSourcePixelRate	Bandwidth required by current configuration
dwVMCPixelRate	Bandwidth allocated to or available to stream
dwDestPixelRate	Bandwidth required through destination given current configuration

Appendix F. Message Parameter Buffer Definitions

This section defines the parameters associated with each Stream Manager and Device Type Driver message. Parameter buffer definitions appear under the relevant message heading. Structure members returned to the caller are highlighted in bold.

GENERIC VMC MESSAGES

```
#define DRV_VMC_GEN_RESERVED          DRV_RESERVED

#define DRV_GET_ERROR_TEXT          (DRV_VMC_GEN_RESERVED + 1)
#define DRV_QUERY_VMC_VERSION       (DRV_VMC_GEN_RESERVED + 2)
```

```
/* DRV_GET_ERROR_TEXT */
typedef struct tagERRORTEXTPARAMS
{
   vmcWORD  wErrorCode;
   vmcWORD  wStringLength;
   vmcCHAR   szErrorText[MAX_ERROR_TEXT_LEN + 1];
} ERROR_TEXT_PARAMS;
typedef ERROR_TEXT_PARAMS vmcFAR *LPERROR_TEXT_PARAMS;
```

Member	Description
wErrorCode	VESA defined ID of error code. The maximum error code value is defined by VMCERR_MAX_ERRORS.
wStringlength	number of characters (excluding NULL terminator), returned to caller
szErrorText	NULL terminated error text, returned to caller.

```
/* DRV_QUERY_VMC_VERSION */
typedef struct
{
   vmcDWORD     dwSize;
   vmcWORD  wVersion;
} QUERY_VERSION_PARAMS;
typedef QUERY_VERSION_PARAMS vmcFAR *LPQUERY_VERSION_PARAMS;
```

Member	Description
dwSize	Size (in Bytes) of this structure
wVersion	VMC software spec. version supported by driver, returned to caller. Hi BYTE = version, LO BYTE = revision

STREAM MANAGER MESSAGES

```
#define DRV_VSM_RESERVED                    (DRV_RESERVED  + 0x100)

#define DRV_VSM_ENABLE                      (DRV_VSM_RESERVED + 1)
#define DRV_VSM_DISABLE                        (DRV_VSM_RESERVED + 2)
#define DRV_VSM_SET_STREAM_CLIP_DATA        (DRV_VSM_RESERVED + 3)
#define DRV_VSM_ALLOCATE_STREAM                (DRV_VSM_RESERVED + 4)
#define DRV_VSM_DEALLOCATE_STREAM              (DRV_VSM_RESERVED + 5)
#define DRV_VSM_ENABLE_STREAM               (DRV_VSM_RESERVED + 6)
#define DRV_VSM_DISABLE_STREAM                 (DRV_VSM_RESERVED + 7)
#define DRV_VSM_ATTACH_STREAM_DEVICE           (DRV_VSM_RESERVED + 8)
#define DRV_VSM_DETACH_STREAM_DEVICE           (DRV_VSM_RESERVED + 9)
#define DRV_VSM_CONFIGURE_STREAM            (DRV_VSM_RESERVED + 10)
```

```
#define DRV_VSM_GET_VMC_BANDWIDTHINFO          (DRV_VSM_RESERVED + 12)
#define DRV_VSM_QUERY_STREAM_CONFIGURATION     (DRV_VSM_RESERVED + 13)
#define DRV_VSM_NON_STREAM_WRITE               (DRV_VSM_RESERVED +  14)
#define DRV_VSM_NON_STREAM_READ                (DRV_VSM_RESERVED + 15)
```

/* DRV_VSM_ALLOCATE_STREAM */

```
typedef struct
{
  vmcDWORD    dwSize;
  vmcDWORD    dwStreamType;
  vmcWORD     wStreamID;
} ALLOCATE_STREAM_PARAMS;
typedef ALLOCATE_STREAM_PARAMS vmcFAR *LPALLOCATE_STREAM_PARAMS;
```

Member	Description
dwSize	Size (in bytes) of this structure
dwStreamType	Type of Stream to allocate - currently, VST_VIDEO_STREAM is the only valid type.
wStreamID	Stream identifier, returned to caller

/* DRV_VSM_DEALLOCATE_STREAM */

```
typedef struct
{
  vmcDWORD    dwSize;
  STREAM_OBJ  oStream;
} DEALLOCATE_STREAM_PARAMS;
typedef DEALLOCATE_STREAM_PARAMS vmcFAR *LPDEALLOCATE_STREAM_PARAMS;
```

Member	Description
dwSize	Size (in bytes) of this structure
oStream	Stream to release, passed in by caller.

/* DRV_VSM_ATTACH_STREAM_DEVICE */

```
typedef struct
{
  vmcDWORD        dwSize;
  STREAM_OBJ      oStream;        /* stream ID obtained through allocateStream */
  DEVICE_OBJ      oDevice;        /* Device context */
  vmcBOOLEAN      bIsTransmitter; /* vmcTRUE if stream transmitter */
  vmcBOOLEAN      bIsClipDevice;  /* vmcTRUE if Stream clip device */
} ATTACH_STREAM_DEVICE_PARAMS;
typedef ATTACH_STREAM_DEVICE_PARAMS vmcFAR *LPATTACH_STREAM_DEVICE_PARAMS;
```

Member	Description
dwSize	Size (in bytes) of this structure
oStream	Stream object with which to associate device
oDevice	Device Context to associate with specified stream
bIsTransmitter	vmcTRUE if device is the stream transmitter, otherwise vmcFALSE.
bIsClipDevice	vmcTRUE if the device is to handle clipping, otherwise vmcFALSE.

/* DRV_VSM_DETACH_STREAM_DEVICE */

```
typedef struct
{
  vmcDWORD        dwSize;
```

```
   STREAM_OBJ        oStream;       /* stream ID obtained through allocateStream */
   DEVICE_OBJ        oDevice;       /* Device context  to detach */
} DETACH_STREAM_DEVICE_PARAMS;
typedef DETACH_STREAM_DEVICE_PARAMS vmcFAR *LPDETACH_STREAM_DEVICE_PARAMS;
```

Member	Description
dwSize	Size (in bytes) of this structure
oStream	Stream object with which to de-associate device
oDevice	Device Context to de-associate

```
/* DRV_VSM_ENABLE_STREAM       */
typedef struct
{
   vmcDWORD     dwSize;
   STREAM_OBJ  oStream;
} ENABLE_STREAM_PARAMS;
typedef ENABLE_STREAM_PARAMS vmcFAR *LPENABLE_STREAM_PARAMS;
```

Member	Description
dwSize	Size (in Bytes) of this structure
oStream	Stream Object to enable.

```
/* DRV_VSM_DISABLE_STREAM       */
typedef struct
{
   vmcDWORD     dwSize;
   STREAM_OBJ  oStream;
} DISABLE_STREAM_PARAMS;
 typedef DISABLE_STREAM_PARAMS vmcFAR *LPDISABLE_STREAM_PARAMS;
```

Member	Description
dwSize	Size (in Bytes) of this structure
oStream	Stream object to disable

```
/* DRV_VSM_CONFIGURE_STREAM  */
typedef struct
{
   STREAM_OBJ                oStream;
   VIDEO_STREAM_CONFIG       VideoStreamParams;
} CONFIGURE_STREAM_PARAMS;
typedef CONFIGURE_STREAM_PARAMS vmcFAR *LPCONFIGURE_STREAM_PARAMS;
```

Member	Description
oStream	Stream object to configure
VideoStreamParams	stream specific configuration data, defined by the caller.

```
/* DRV_VSM_GET_VMC_BANDWIDTHINFO */
typedef struct
{
   vmcDWORD                  dwSize;
   VMC_BANDWIDTHINFO     VMCBandwidthInfo;
} GET_VMC_BANDWIDTH_PARAMS;
typedef GET_VMC_BANDWIDTH_PARAMS    vmcFAR *LPGET_VMC_BANDWIDTH_PARAMS;
```

Member	Description
dwSize	Size (in Bytes) of this structure
VMCBandwidthInfo	VMC bandwidth Info Block, returned to caller

```
/*    DRV_VSM_QUERY_STREAM_CONFIGURATION  */
typedef struct
{
   vmcDWORD                    dwSize;
   STREAM_OBJ                  oStream;
   vmcBOOLEAN                  bScaleAtSource;
   vmcBOOLEAN                  bMaintainAspect;
   VIDEO_STREAM_CONFIG             VideoStreamParams;
   VIDEO_STREAM_CONFIG         SuggestedStreamParams;
   STREAM_BANDWIDTHINFO    StreamBandwidthParams;
} QUERY_STREAM_CONFIG_PARAMS;
typedef  QUERY_STREAM_CONFIG_PARAMS    vmcFAR *LPQUERY_STREAM_CONFIG_PARAMS;
```

Member	Description
dwSize	Size (in Bytes) of this structure.
oStream	Stream Object to query stream configuration
bScaleAtSource	vmcTRUE if caller wishes scaling (if possible) to be performed by source device. Otherwise, vmcFALSE.
bMaintainAspect	vmcTRUE if caller wishes the calculated video scale factors to maintain aspect ratio. Otherwise, set to vmcFALSE.
VideoStreamParams	Required stream configuration, defined by caller
SuggestedStreamParams	Stream configuration that can be supported given bandwidth and device restrictions -> returned to caller
StreamBandwidthParams	Information to indicate stream bandwidth requirements-> returned to caller

```
/* DRV_VSM_SET_STREAM_CLIP_DATA   */
typedef struct
{
  vmcDWORD          dwSize;
  DEVICE_OBJ        oStream;
  VMC_CLIPINFO      ClipData;
} SET_STREAM_CLIP_DATA_PARAMS;
typedef SET_STREAM_CLIP_DATA_PARAMS vmcFAR *LPSET_STREAM_CLIP_DATA_PARAMS;
```

Member	Description
dwSize	Size (in Bytes) of this structure.
oStream	Stream Object to set clipping data for
ClipData	Stream Clip data

```
/* DRV_VSM_NON_STREAM_WRITE   */
typedef struct
{
  vmcDWORD     dwSize;
  vmcWORD      wVMCDeviceID;
  vmcDWORD     dwDeviceAddress;
  vmcDWORD     dwDwordsToWrite;
  vmcDPTR      lpDeviceData;
} NON_STREAM_WRITE_PARAMS;
```

typedef NON_STREAM_WRITE_PARAMS vmcFAR *LPNON_STREAM_WRITE_PARAMS;

Member	Description
dwSize	Size (in Bytes) of this structure.
wVMCDeviceID	ID of VMC device to write to
dwDeviceAddress	24-bit start address in the destination device
dwDwordsToWrite	number of 32bit words to write.
lpDeviceData	pointer to buffer containing 32-bit data values to write.

```
/* DRV_VSM_NON_STREAM_READ   */
 typedef struct
 {
   vmcDWORD     dwSize;
   vmcWORD      wVMCDeviceID;
   vmcDWORD     dwDeviceAddress;
   vmcDWORD     dwDwordsToRead;
   vmcDPTR      lpDeviceData;
 } NON_STREAM_READ_PARAMS;
typedef NON_STREAM_READ_PARAMS vmcFAR *LPNON_STREAM_READ_PARAMS;
```

Member	Description
dwSize	Size (in Bytes) of this structure.
wVMCDeviceID	ID of VMC device to read from
dwDeviceAddress	24-bit start address in the destination device to read
dwDwordsToRead	number of 32bit values to read.
lpDeviceData	pointer to buffer to receive 32-bit device data

DEVICE CONTEXT MESSAGES

```
#define DRV_VMC_RESERVED                        (DRV_RESERVED + 0x200)

#define DRV_VMC_CREATE_CONTEXT                      (DRV_VMC_RESERVED + 1)
#define DRV_VMC_DESTROY_CONTEXT                 (DRV_VMC_RESERVED + 2)
#define DRV_VMC_QUERY_NUM_VMC_DEVICES           (DRV_VMC_RESERVED + 3)
#define DRV_VMC_RESET                           (DRV_VMC_RESERVED + 4)
#define DRV_VMC_STREAM_PREPARE                      (DRV_VMC_RESERVED + 5)
#define DRV_VMC_STREAM_UNPREPARE                (DRV_VMC_RESERVED + 6)
#define DRV_VMC_ENABLE_STREAM                   (DRV_VMC_RESERVED + 7)
#define DRV_VMC_DISABLE_STREAM                  (DRV_VMC_RESERVED + 8)
#define DRV_VMC_QUERY_VENDOR_INFO                   (DRV_VMC_RESERVED + 9)
#define DRV_VMC_QUERY_VMC_INFO                  (DRV_VMC_RESERVED + 10)
#define DRV_VMC_QUERY_CONTEXT_CAPS              (DRV_VMC_RESERVED + 11)
#define DRV_VMC_QUERY_STREAM_CAPS                   (DRV_VMC_RESERVED + 12)
#define DRV_VMC_QUERY_SCALER_CAPS                   (DRV_VMC_RESERVED + 13)
#define DRV_VMC_ASSOCIATE_STREAMID                  (DRV_VMC_RESERVED + 14)
#define DRV_VMC_DISASSOCIATE_STREAMID               (DRV_VMC_RESERVED + 15)
#define DRV_VMC_SET_CLIP_DATA                       (DRV_VMC_RESERVED + 16)
#define DRV_VMC_QUERY_NUM_SCALE_BANDS           (DRV_VMC_RESERVED + 17)
#define DRV_VMC_QUERY_CONTEXT_BANDWIDTH         (DRV_VMC_RESERVED + 18)
#define DRV_VMC_VALIDATE_VERTICAL_SCALING       (DRV_VMC_RESERVED + 19)
#define DRV_VMC_VALIDATE_HORIZONTAL_SCALING         DRV_VMC_RESERVED +  20)
```

```
/*     DRV_VMC_CREATE_CONTEXT */
typedef struct
{
   vmcDWORD     dwSize;
   vmcDWORD     dwContextType;
```

```
  vmcDWORD    dwContextSubType;
  vmcWORD     wDeviceInstance;
  vmcHANDLE   hContext;
} CREATE_CONTEXT_PARAMS;
typedef CREATE_CONTEXT_PARAMS vmcFAR *LPCREATE_CONTEXT_PARAMS;
```

Member	Description
dwSize	Size (in bytes) of this structure
dwDeviceContextType	Context type to be created - current values are CT_VIDEO_SOURCE and CT_VIDEO_SINK for video subsystems and CT_GRAPHIC_SINK and CT_GRAPHIC_SOURCE for graphic subsystems.
dwContextSubType	Subtype to be created - current values are ST_VIDEO_DECODER, ST_VIDEO_ENCODER, ST_CODEC_SOURCE, ST_CODEC_SINK, ST_FX_PROCESSOR_SOURCE and ST_FX_PROCESSOR_SINK for video subsystems and ST_ON_SCREEN for graphic subsystems.
wDeviceInstance	Logical VMC Device Instance Id supported by the type driver. A VMC Client should obtain the number of logical devices supported by a driver through the DRV_VMC_QUERY_NUM_VMC_DEVICES message. The capabilities for each logical device should be verified before creating a context.
hContext	Device Context ID, returned to caller

```
/*    DRV_VMC_DESTROY_CONTEXT   */
typedef struct
{
  vmcDWORD    dwSize;
  DEVICE_OBJ  oDevice;
} DESTROY_CONTEXT_PARAMS;
typedef DESTROY_CONTEXT_PARAMS vmcFAR *LPDESTROY_CONTEXT_PARAMS;
```

Member	Description
dwSize	Size (in Bytes) of this structure.
oDevice	Device context object to be destroyed.

```
/*    DRV_VMC_QUERY_NUM_VMC_DEVICES          */
typedef struct
{
  vmcDWORD    dwSize;
  vmcWORD     wNumVMCDevices;
} QUERY_NUM_VMC_DEVICES_PARAMS;
typedef QUERY_NUM_VMC_DEVICES_PARAMS vmcFAR *LPQUERY_NUM_VMC_DEVICES_PARAMS;
```

Member	Description
dwSize	Size (in Bytes) of this structure.
wNumVMCDevices	Number of VMC Devices supported by Device Type Driver

```
/* DRV_VMC_STREAM_PREPARE */
typedef struct
{
  vmcDWORD                dwSize;
  DEVICE_OBJ              oDevice;
```

```
  VIDEO_STREAM_CONFIG       VideoStreamParams;
} STREAM_PREPARE_PARAMS;
typedef STREAM_PREPARE_PARAMS vmcFAR *LPSTREAM_PREPARE_PARAMS;
```

Member	Description
dwSize	Size (in Bytes) of this structure
oDevice	Device context to prepare
VideoStreamParams	Stream specific configuration parameters

```
/* DRV_VMC_STREAM_UNPREPARE     */
typedef struct
{
  vmcDWORD     dwSize;
  DEVICE_OBJ   oDevice;
} STREAM_UNPREPARE_PARAMS;
typedef STREAM_UNPREPARE_PARAMS vmcFAR *LPSTREAM_UNPREPARE_PARAMS;
```

Member	Description
dwSize	Size (in Bytes) of this structure.
oDevice	Device context to unprepare

```
/*    DRV_VMC_ENABLE_STREAM   */
typedef struct
{
  vmcDWORD     dwSize;
  DEVICE_OBJ   oDevice;
  vmcWORD  wGTR;
} ENABLE_CONTEXT_PARAMS;
typedef ENABLE_CONTEXT_PARAMS vmcFAR *LPENABLE_CONTEXT_PARAMS;
```

Member	Description
dwSize	Size (in Bytes) of this structure.
oDevice	Device Context to Enable
wGTR	GTR to be programmed into device in support of latest stream configurations.

```
/*    DRV_VMC_DISABLE_STREAM  */
typedef struct
{
  vmcDWORD     dwSize;
  DEVICE_OBJ   oDevice;
  vmcWORD      wGTR;
} DISABLE_CONTEXT_PARAMS;
typedef DISABLE_CONTEXT_PARAMS vmcFAR *LPDISABLE_CONTEXT_PARAMS;
```

Member	Description
dwSize	Size (in Bytes) of this structure.
oDevice	Device Context to Disable
wGTR	GTR value to be programmed into device to support remaining streams.

```
/*    DRV_VMC_QUERY_VENDOR_INFO    */
typedef struct
{
  vmcDWORD              dwSize;
```

```
  vmcWORD          wDeviceInstance;
  VESA_DEVICECAPS  VESADevCaps;
} QUERY_VENDOR_INFO_PARAMS;
typedef QUERY_VENDOR_INFO_PARAMS vmcFAR *LPQUERY_VENDOR_INFO_PARAMS;
```

Member	Description
dwSize	Size (in Bytes) of this structure.
wDeviceInstance	Logical Device Instance. A VMC Client should obtain the number of logical devices supported by a driver through the DRV_VMC_QUERY_NUM_VMC_DEVICES message.
VESADevCaps	Vendor information device caps for the specified logical Device Instance, returned to caller.

```
/*    DRV_VMC_QUERY_VMC_INFO  */
typedef struct
{
  vmcDWORD         dwSize;
  DEVICE_OBJ       oDevice;
  VMC_DEVICECAPS   VMCDevCaps;
} QUERY_VMC_DEVINFO_PARAMS;
typedef QUERY_VMC_DEVINFO_PARAMS vmcFAR *LPQUERY_VMC_DEVINFO_PARAMS;
```

Member	Description
dwSize	Size (in Bytes) of this structure.
oDevice	Device context for which to obtain DEVCAPS.
VMCDevCaps	VMC device information for specified context, returned to caller

```
/*    DRV_VMC_QUERY_CONTEXT_CAPS */
typedef struct
{
  vmcDWORD          dwSize;
  DEVICE_OBJ        oDevice;
  VIDEO_CONTEXT_CAPS  ContextCaps;
} QUERY_CONTEXT_CAPS_PARAMS;
typedef QUERY_CONTEXT_CAPS_PARAMS vmcFAR *LPQUERY_CONTEXT_CAPS_PARAMS;
```

Member	Description
dwSize	Size (in Bytes) of this structure.
oDevice	Device Context for which to obtain Context Capabilities.
ContextCaps	Video Context capabilities , returned to caller

```
/*    DRV_VMC_QUERY_STREAM_CAPS  */
typedef struct
{
  vmcDWORD          dwSize;
  DEVICE_OBJ        oDevice;
  vmcWORD           wStreamMode;
  VIDEO_STREAM_CAPS   StreamCaps;
} QUERY_STREAM_CAPS_PARAMS;
typedef QUERY_STREAM_CAPS_PARAMS vmcFAR *LPQUERY_STREAM_CAPS_PARAMS;
```

Member	Description
dwSize	Size (in Bytes) of this structure.
oDevice	Device Context associated with stream.
wStreamMode	Stream mode to query - must be one of the stream modes supported by the context.
wStreamCaps	Video Stream capabilities, returned to caller

```
/*    DRV_VMC_QUERY_SCALER_CAPS    */
typedef struct
{
   vmcDWORD              dwSize;
   DEVICE_OBJ            oDevice;
   vmcWORD               dwStreamFormat;
   vmcWORD               wBitsPerPixel;
   vmcWORD               wStreamMode;
   vmcDWORD              dwScalerMode;
   VIDEO_SCALER_CAPS     ScalerCaps;
} QUERY_SCALER_CAPS_PARAMS;
typedef QUERY_SCALER_CAPS_PARAMS vmcFAR *LPQUERY_SCALER_CAPS_PARAMS;
```

Member	Description
dwSize	Size (in Bytes) of this structure.
oDevice	Device Context associated with stream mode.
wStreamFormat	VMC Data format
wBitsPerPixel	pixel depth for the specified data format
wStreamMode	Stream mode associated with the scaler
dwScalerMode	scaler mode to query - must be one of the scaler modes supported by the defined stream mode and device context.
ScalerCaps	Video Stream capabilities, returned to caller

```
/* DRV_VMC_QUERY_CONTEXT_BANDWIDTH */
typedef struct
{
   vmcDWORD          dwSize;
   DEVICE_OBJ        oDevice;
   vmcWORD           wStreamMode;
   vmcWORD           wStreamFormat;
   vmcWORD           wBitsPerPixel;
   vmcDWORD          dwScalerMode;
   VMC_RECTINFO      rSourceRect;
   VMC_RECTINFO      rStreamRect;
   vmcDWORD          dwPeakBandwidth;
} QUERY_CONTEXT_BANDWIDTH_PARAMS;
typedef QUERY_CONTEXT_BANDWIDTH_PARAMS vmcFAR *LPQUERY_CONTEXT_BANDWIDTH_PARAMS;
```

Member	Description
dwSize	Size (in Bytes) of this structure.
oDevice	Device Context associated with scaler
wStreamMode	Stream mode associated with the context
wStreamFormat	VMC Data format
wBitsPerPixel	Pixel depth associated with specified data format.
dwScalerMode	scaler mode associated with the context

rSourceRect	Source Image Size
rStreamRect	Stream Image Size
dwPeakBandwidth	Peak Bandwidth required by specified context configuration, returned to caller

```
/* DRV_VMC_QUERY_NUM_SCALE_BANDS */
typedef struct
{
   vmcDWORD          dwSize;
   DEVICE_OBJ        oDevice;
   vmcWORD           wStreamMode;
   vmcWORD           wStreamFormat;
   vmcWORD           wBitsPerPixel;
   vmcDWORD          dwScalerMode;
   VMC_RECTINFO      rSourceRect;
   VMC_RECTINFO      rStreamRect;
   vmcINT            iNumBands;
} QUERY_NUM_SCALE_BAND_PARAMS;
typedef QUERY_NUM_SCALE_BAND_PARAMS vmcFAR *LPQUERY_NUM_SCALE_BAND_PARAMS;
```

Member	Description
dwSize	Size (in Bytes) of this structure.
oDevice	Device Context to query
wStreamMode	Stream mode associated with the context
wStreamFormat	Data format
wBitsPerPixel	pixel depth for the specified data format
dwScalerMode	Scaler mode associated with the context
rSourceRect	Source Image Size
rStreamRect	Stream Image Size
iNumBands	Number of vertical scaler bands returned to caller

```
/ *DRV_VMC_VALIDATE_HORIZONTAL_SCALING  */
/* and */
/* DRV_VMC_VALIDATE_VERTICAL_SCALING      */

typedef struct
{
   vmcDWORD                dwSize;
   DEVICE_OBJ              oDevice;
   vmcWORD                 wScalerBand;
   VIDEO_STREAM_CONFIG     StreamConfig;
   VIDEO_SCALER_CAPS       SourceScalerMode;
   VIDEO_SCALER_CAPS       DestScalerMode;
   vmcDWORD                dwPeakBandwidth;
} VALIDATE_SCALING_PARAMS;
typedef VALIDATE_SCALING_PARAMS vmcFAR *LPVALIDATE_SCALING_PARAMS;
```

Member	Description
dwSize	Size (in Bytes) of this structure.
oDevice	Device Context to Validate scaling for
wScaleBand	Vertical Scale Band

StreamConfig	Stream Configuration requested by caller. NOTE: While this structure is filled in by the caller, any limitations detected by the driver in validating the scale factors are made to the stream config structure and passed back to the caller.
SourceScalerMode	Source scaler mode capabilities
DestScalerMode	destination scaler mode capabilities
dwPeakBandwidth	Bandwidth available to device context, as defined by th stream manager.

```
/*    DRV_VMC_ASSOCIATE_STREAMID   */
typedef struct
{
   vmcDWORD     dwSize;
   DEVICE_OBJ   oDevice;
   STREAM_OBJ  oStream;
   vmcBOOLEAN  bIsTransmitter;
   vmcBOOLEAN  bIsClipDevice;
} ASSOCIATE_STREAMID_PARAMS;
typedef ASSOCIATE_STREAMID_PARAMS vmcFAR *LPASSOCIATE_STREAMID_PARAMS;
```

Member	Description
dwSize	Size (in Bytes) of this structure.
oDevice	Device context to associate with stream.
oStream	Stream to associate with the device context
bIsTransmitter	vmcTRUE if device has been assigned stream transmitter, otherwise vmcFALSE.
bIsClipDevice	vmcTRUE if device assigned to perform stream clipping, otherwise vmcFALSE.

```
/*    DRV_VMC_DISASSOCIATE_STREAMID   */
typedef struct
{
   vmcDWORD     dwSize;
   DEVICE_OBJ   oDevice;
} DISASSOCIATE_STREAMID_PARAMS;
 typedef DISASSOCIATE_STREAMID_PARAMS vmcFAR *LPDISASSOCIATE_STREAMID_PARAMS;
```

Member	Description
dwSize	Size (in Bytes) of this structure.
oDevice	Device context to deassociate from stream.

```
/* DRV_VMC_SET_CLIP_DATA   */
 typedef struct
 {
   vmcDWORD          dwSize;
   DEVICE_OBJ        oDevice;
   VMC_CLIPINFO      ClipData;
 } SET_CLIP_DATA_PARAMS;
 typedef SET_CLIP_DATA_PARAMS vmcFAR *LPSET_CLIP_DATA_PARAMS;
```

Member	Description
dwSize	Size (in Bytes) of this structure.
oDevice	Device context to set clipping data for

ClipData	Clip data

PRIVATE DRIVER MESSAGES

```
/*
 *Vendor Specific mesage definitions
 * The following define the block of available vendor-specific messages available for
 * hardware vendors to add messages to support extra, non-VMC-standard features.
 */
#define DRV_VENDOR_RESERVED          DRV_RESERVED + 0x1000
#define DRV_VENDOR_STOP          DRV_RESERVED + 0x1200
```

End Of Document