MVEX

Minimal Video Extension to X
Version 6.0

Todd Brunhoff

Tektronix, Inc.
Tektronix Research Laboratories

X Window System is a trademark of M.I.T.

Copyright © 1989, 1990, 1991 Tektronix, Inc.

Permission to use, copy, modify, and distribute this document for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice are retained, and that the name of Tektronix not be used in advertising or publicity pertaining to this document without specific, written prior permission. Tektronix makes no representations about the suitability of this document or the protocol defined in this document for any purpose. It is provided ‘‘as is’’ without express or implied warranty.

1. Overview

The spirit of this extension is to provide an X interface to the generally interesting aspects of displaying live video in windows, capturing graphics from windows and converting them to a video signal. Based on an earlier extension called VEX, this extension endevors to provide the minimal support needed for integrating video into the window system.

For applications that need access to the digitized video pixels, this extension allows normal pixmaps or windows with core visual classes to hold video, which in turn allows read and write access to the pixels via the core protocol or other extensions such as the X Image Extension. For hardware that does not actually digitize any pixels, MVEX introduces two new visual classes: `VideoColor` and `VideoGray`. Their purpose is to express the visual aspects of a visible but untouchable video picture, hence drawing on a window with one of these visual classes has no effect and the pixels are undefined.

Today, video input and output hardware has complex limitations and capabilities; the capabilities do not always scale in expected ways, nor do the limitations always make sense. The MVEX extension can provide large amounts of information about these limitations and capabilities. It is intended that the client, through the use of convenience routines in the MVEXlib implementation, will be able to easily determine what it can and can't do without the overhead of a round-trip request to the server.

2. Terminology

Full-motion video

The display of video on the screen at a rate at least as fast as that defined by the encoded video signal. For example NTSC is defined as (roughly) 30 frames per second and 60 fields per second.

Opaque pixels

Some video input hardware uses a single frame buffer to hold both video pixels and graphic pixels. If the video pixels are stored in a form that does not match a core visual class, then the meaning of those pixels, from a client perspective, is said to be opaque.

Still video

A single, static frame acquired from a video signal.

Video digitization

A process that converts a video image from an encoded signal into a digital form having a supported depth and format.

Video input

The term *video input* in general refers to an incoming signal that is decoded or digitized to produce a picture on the workstation display. In this document, video input refers to the digitizing or decoding hardware, and it is identified by a VIDEOIN id. Logically, a client will connect an output device, such as a video disk player, to a video input using the ChangeConnectivity request, and then use `RenderVideo` to direct in which drawable the picture will appear.

Video input region (VIR)

This is defined as the subset of pixels in a VW or a pixmap that is actually being updated by a `RenderVideo` request.

Video output

The term *video output* in general refers to signals produced by encoding workstation graphics. In this document, video output refers to the encoding hardware, and it is identified by a VIDEOOUT id. Logically, a client will connect an input device, such as a video tape recorder, to a video output using the ChangeConnectivity request, and then use `CaptureGraphics` to direct from which drawable the graphics will be captured.

Video output region (VOR)

This is defined as the set of pixels on the screen or in a pixmap that are being encoded by a `CaptureGraphics` request, whether it is accomplished by reading pixels from a frame buffer or by analog conversion.

Video window (VW)

A video window is defined as any window actively displaying video or one that has the potential to display nothing but video. Specifically,

- a window having a visual class `VideoColor` or `VideoGray`.
- a window with a core visual class, and actively servicing a `RenderVideo` request.

3. Visuals

There are three important classes of video input hardware. Each class produces a video picture on the screen using a different method, and the methods affects the visual class used to represent the window or pixmap that contains it. MVEX attempts to recognize all three classes:

- Core visual pixels. Many instances of hardware choose to represent a video picture in a frame buffer that is no different than most graphic frame buffers. In this case nothing new need be added, because one of the core visual classes is sufficient.
- Inaccessible pixels. Some low-cost video input hardware is contained in a separate subsystem where either pixels are never digitized, or if they are, the pixels are available only at low serial line speeds. The analog video picture is merged with the workstation graphics analog signal such that the picture is updated very quickly. Therefore, this hardware will continue to be reasonable as long as video is not a standard option for every X display. For this, MVEX introduces two new visual class, `VideoGray` and `VideoColor`, whose semantics essentially describe a write-only visual: video pixel values cannot be read. This implies that pixmap access for these visual types will never be offered.
- Opaque pixels. A few instances of hardware choose to represent digitized pixels as an encoded color difference, such as YUV; Y,R-Y,B-Y; or YIQ. This encoding is more compact than RGB, but cannot be represented with any of the core visual classes. Rather than invent more new color models and visual classes, MVEX allows the use of whatever core visual class matches the frame buffer's "normal" mode, but pixels that are modified by `RenderVideo` are opaque. This implies that while the pixels can be read, their meaning is undefined (note that a vendor may choose to provide more information about the meaning of these pixels via control mechanisms, described later). See the description for the RENDERMODEL type and the `RenderVideo` request.

Windows created with `VideoGray` display only the luminance portion of a video signal; windows created with `VideoColor` are able to display a color picture if chrominance information is available in the video signal. A side effect of this visual class is that the visible rendition of regions not covered by a `RenderVideo` request are undefined, including the window's border. However, it is expected that server implementors will pick reasonable defaults.

4. Types

VIDEOIN: 32-bit value (top three bits guaranteed to be zero)

VIDEOOUT: 32-bit value (top three bits guaranteed to be zero)

VIDEOIO: VIDEOIN or VIDEOUT

When a VIDEOIN or VIDEOOUT is supplied to a request, it must be one returned by the `CreateVideo` request, else `Match` error.

VREFERENCE: 32-bit value (top three bits guaranteed to be zero)

This is a special id supplied in a VIDEOGEOMETRY structure that uniquely identifies the corresponding resource.

FRACTION: [numerator, denominator: INT32]

The FRACTION type is used to represent a rational number; the denominator must be strictly positive (> 0) in all requests, else `Value` error.

FRACTIONRANGE: [num-base, num-inc, num-limit: INT32
num-type: {`Linear`, `Geometric`}
denom-base, denom-inc, denom-limit: INT32
denom-type: {`Linear`, `Geometric`}]

A FRACTIONRANGE is used in a PLACEMENT, described below. The set of fractions represented by FRACTIONRANGE depend on the types, and for each type, the range of numerators and denominators of each fraction are as follows:

`Linear`: n : {n = base to limit step inc

`Geometric`: n^{exp} : {n = base; exp = inc to limit step 1

A FRACTIONRANGE supplied by `QueryVideo` is guaranteed never to generate a fraction with a zero denominator, or a fractional numerator or denominator. For `Linear`, *num-inc* and *denom-inc* are guaranteed to be greater than zero, and the limit is always guaranteed to be *-base* plus some integral multiple of *-inc*. For `Geometric`, *num-inc* and *denom-inc* are guaranteed to be greater than or equal to zero. For example, a FRACTIONRANGE of {512, 128, 1024, `Linear`, 512, 512, 1024, `Linear`} creates a set containing

$$\left\{\frac{512}{512}, \frac{640}{512}, \frac{768}{512}, \frac{896}{512}, \frac{1024}{512}, \frac{512}{1024}, \frac{640}{1024}, \frac{768}{1024}, \frac{896}{1024}, \frac{1024}{1024}\right\}.$$

A FRACTIONRANGE of {2, 0, 2, `Geometric`, 1, 1, 127, `Linear`} creates the set

$$\left\{\left[\frac{2^0}{1} = \right]\frac{1}{1}, \frac{1}{2}, \frac{1}{3}, ..., \frac{1}{127}, \frac{2}{1}, \frac{2}{2}, ..., \frac{2}{127}, \frac{4}{1}, \frac{4}{2}, ..., \frac{4}{127}\right\}.$$

MARKER: [kind: {Timecode, Frame, None}
relative: BOOL
data: TIMECODE or FRAME or empty]

The MARKER is used by `RenderVideo`, `CaptureGraphics` and `StopVideo` to indicate a relative or absolute point when the request will take effect. If Timecode is specified, then the data is a TIMECODE structure. If Frame is specified, then the data is a FRAME structure. If None is specified, then the data is empty. If relative is true, then the marker is relative to the current timecode or frame; otherwise it designates an absolute timecode or frame. If relative is false, then the negative element in the TIMECODE or FRAME must be false.

Whether timecodes or frame numbers are available from the video media is outside the domain of MVEX; i.e., another system must be used to obtain media information and events.

FRAME: [negative: BOOL
frame: CARD32
field: CARD8]

The FRAME is used by the MARKER type, and indirectly by `RenderVideo`, `CaptureGraphics` and `StopVideo` to indicate a relative or absolute frame when the request will take effect. If negative is true, then it refers to a negative frame number.

TIMECODE: [negative: BOOL
hour, minute, second, frame, field: CARD8]

The TIMECODE is used by the MARKER type, and indirectly by `RenderVideo`, `CaptureGraphics` and `StopVideo` to indicate a relative or absolute time point when the request will take effect. If negative is true, then it refers to a negative timecode.

RECTANGLERANGE: [base, limit: RECTANGLE
x-inc, y-inc: INT16
width-inc, height-inc: CARD16
type: {`Linear`, `Geometric`}]

The RECTANGLERANGE is ultimately used in the VIDEOGEOMETRY type below. It describes the possible geometries of sources and destinations for `RenderVideo` and `CaptureGraphics` requests. The set of rectangles described by a single RECTANGLERANGE are all combinations of four sets derived for x, y, width and height. The members of the sets are determined based on the type as follows.

`Linear`:
- x = { base.x, base.x + x-inc, base.x + 2·x-inc, ..., limit.x }
- y = { base.y, base.y + y-inc, base.y + 2·y-inc, ..., limit.y }
- width = { base.width, base.width + width-inc, base.width + 2·width-inc, ..., limit.width }
- height = { base.height, base.height + height-inc, base.height + 2·height-inc, ..., limit.height }

`Geometric`:
- x = { $base.x^{x\text{-}inc}$, $base.x^{x\text{-}inc+1}$, ..., $base.x^{limit.x}$ }
- y = { $base.y^{y\text{-}inc}$, $base.y^{y\text{-}inc+1}$, ..., $base.y^{limit.y}$ }
- width = { $base.width^{width\text{-}inc}$, $base.width^{width\text{-}inc+1}$, ..., $base.width^{limit.width}$ }
- height = { $base.height^{height\text{-}inc}$, $base.height^{height\text{-}inc+1}$, ..., $base.height^{limit.height}$ }

A RECTANGLERANGE supplied by `QueryVideo` is guaranteed never to generate a rectangle with a fractional component. For `Linear`, the *width-inc* and *height-inc* values are guaranteed to be greater than zero, and the *x-inc* and *y-inc* are guaranteed to be non-zero. For example, the RECTANGLERANGE whose value is

[base = {0, 0, 320, 240}, limit = {960, 784, 320, 240}
x-inc = 16, y-inc = 1
width-inc = 0, height-inc = 0
type = `Linear`]

would describe a total of 47,885 {x, y, width, height} sets

{0, 0, 320, 240}
{16, 0, 320, 240}
...
{960, 0, 320, 240}
{0, 1, 320, 240}
{16, 1, 320, 240}
...
{960, 1, 320, 240}
etc.
{960, 784, 320, 240}

PLACEMENT: [frame-rate: FRACTION
source, destination: RECTANGLERANGE
x-scale, y-scale: FRACTIONRANGE
identity-aspect: BOOL]

The PLACEMENT type is used in the VIDEOGEOMETRY type described below. Each placement describes the possible source and destination rectangles used in `RenderVideo` and `CaptureGraphics` requests, but restricted to the scale factors described by the x-scale and y-scale elements. For video input, the frame-rate element describes the maximum nominal frame rate that the hardware can achieve in a `RenderVideo` request using a source and destination rectangle derived from the other elements. For video output, the frame-rate element describes the maximum nominal rate at which frames can be encoded with a `CaptureGraphics` request. It is important to stress that this is only a nominal rate and may vary due to other factors such as hardware contention.

The x-scale and y-scale FRACTIONRANGE specifies the possible ratios of

destination x pixels : source x pixels
destination y pixels : source y pixels

where the ratio is numerator (destination) divided by denominator (source). The source and destination rectangles derived must have width and height ratios described by x-scale and y-scale. The identity-aspect is true if the ratio of x-scale to y-scale is 1:1; i.e. they must be equal. For example, given a PLACEMENT of

frame-rate	=	[30, 1]
source	=	[base = {0, 0, 10, 15}, limit = {639, 479, 640, 480} x-inc = 1, y-inc = 1 width-inc = 1, height-inc = 1 type = `Linear`]
destination	=	[base = {0, 0, 320, 240}, limit = {960, 784, 320, 240} x-inc = 16, y-inc = 1 width-inc = 0, height-inc = 0 type = `Linear`]
x-scale	=	[num-{base,inc,limit} = {2, 0, 6} type = `Geometric` denom-{base,inc,limit} = {1, 1, 0} type = `Linear`]
y-scale	=	[num-{base,inc,limit} = {2, 0, 5} type = `Geometric` denom-{base,inc,limit} = {1, 1, 0} type = `Linear`]
identity-aspect	=	False

it implies that the source rectangles may be any with the constraints that

$$0 \le x \le 639$$

$$0 \le y \le 479$$

$$10 \le \text{width} \le 640$$

$$15 \le \text{height} \le 480$$

and the destination rectangles may be any selected from the earlier example. The combined rectangles are limited to those source-destination pairs where the fractions

$$x\text{–}scale = \frac{\text{destination width}}{\text{source width}} \quad \text{and} \quad y\text{–}scale = \frac{\text{destination height}}{\text{source height}}$$

can be reduced to one of the fractions from the respective sets

$$x\text{-}scale = \left\{ \frac{1}{1}, \frac{2}{1}, \frac{4}{1}, \frac{8}{1}, \frac{16}{1}, \frac{32}{1}, \frac{64}{1} \right\}.$$

$$x\text{-}scale = \left\{ \frac{1}{1}, \frac{2}{1}, \frac{4}{1}, \frac{8}{1}, \frac{16}{1}, \frac{32}{1} \right\}.$$

And for this example, using any of these source-destination pairs, the hardware can maintain a nominal 30 fps.

VIDEOABILITY: [frames-available: BOOL
frame-accurate: BOOL
n-saturation: CARD32
n-contrast: CARD32
n-hue: CARD32
n-bright: CARD32
saturation: LISTofFRACTIONRANGE
contrast: LISTofFRACTIONRANGE
hue: LISTofFRACTIONRANGE
bright: LISTofFRACTIONRANGE]

This type describes the abilities of video inputs and video outputs and is returned by the QueryVideoRequest. The frames-available is true if the videoio is capable of recognizing frames or timecodes. The frame-accurate is true if `RenderVideo`, `CaptureGraphics` and `StopVideo` can be performed on precise frame boundaries. The n-saturation, n-contrast, n-hue and n-bright length of the corresponding FRACTIONRANGE structures. If a list has zero length, then adjustments and queries are ignored.

Saturation is the amount of color information present in the video image. Small values result in black and white images; large values provide increasing amounts of color.

Contrast determines the dynamic range of information present in the video image. Small values result in an image of constant color and intensity; large values produce more contrast.

Hue shifts the phase of the video's color information relative to its reference, causing a shift in color. A middle value results in no shift, smaller values shift red towards blue and larger values shift red towards green.

Brightness determines the black level of the video signal. Small values result in a dimmer image, large values produce a brighter image.

VIDEOGEOMETRY: [signal-frame-rate: FRACTION
signal-field-rate: FRACTION
signal-width, signal-height: CARD16
concurrent-use: CARD16
reference-id: VREFERENCE
placement: LISTofPLACEMENT
ability: VIDEOABILITY]

This type describes the attributes of video inputs and video outputs and is returned by the `QueryVideo` request. The signal-frame-rate and signal-field-rate describe the number of frames per second and fields per second in the signal format. For input, the rate describes the signal received; for output, the signal produced.

For video input, the signal-width and signal-height describe the dimensions of the signal picture as if it were placed directly on the workstation screen. For video output, they describe the dimensions of a screen region whose pixels would map one-to-one to the

output signal. It is important to note that the frame-buffer holding the video picture, if any, may not actually have these dimensions; the numbers describe the extent of the source rectangle for video input, or the destination rectangle for the video output.

The ability describes some specific abilities for a video input or output.

For a video input and output, the concurrent-use indicates the number of times the resource may be in simultaneous use, and suggests the maximum number identifiers a client should create using `CreateVideo`. The server should publish a number that the hardware can support with respect to connectivity and quality of the input or output. Clients should use this number in deciding how many resources they may use, and a video resource manager should use this in deciding how to handle allocation among multiple clients.

The reference id uniquely identifies the resource, and this id is used by the `CreateVideo` request to create video input and output identifiers. If the same reference id appears in a VIDEOGEOMETRY returned from different screens, then the concurrent use for will be the same and a video manager can infer that one physical resource services both screens. This implies that usage of this resource on one screen reduces the available usage on another screen.

Each placement element in the list describes its own range of values. The complete range of placement parameters is derived from the entire list, but parts of one element in the list may not be combined with another. For example, source and destination rectangles described in one placement may not be combined with scale factors described in another placement.

RENDERMODEL: [depth: CARD8
visual-id: VISUALID
opaque: BOOL
red-mask, green-mask, blue-mask: CARD32]

This type is used by `QueryVideo` to describe the depth and visual class of a potential source for `CaptureGraphics` or destination for `RenderVideo`. The visual id is one returned by the connection setup or the `QueryVideo` request. The boolean, *opaque*, is true when a video input is used to render to a drawable created with this visual id, and the pixel values created cannot be interpreted by a core visual class.

The red, green and blue mask are those used for `CaptureGraphics` when the source is a pixmap and the colormap is `None`; and for `RenderVideo` when the destination is a pixmap and *opaque* is false. A server may supply a RENDERMODEL with a visual id of None for depths that can only be captured from pixmaps; and a RENDERMODEL may be supplied with red-, green-, and blue-mask set to zero; but not both in the same RENDERMODEL.

It is conceivable that a system will support a depth for pixmaps, but not for windows. A server may support a gray scale `CaptureGraphics` for this depth by supplying a RENDERMODEL with visual id None and red- green- and blue- masks with identical bits set.

Note that if *opaque* is true, then the visual id is guaranteed to have a core visual class.

OWNER: [wid: WINDOW
vid: VIDEOIO]

This type is used by the `ChangeOwnership` request to establish ownership for any of the video resources: input, output and devices. The specified window identifies the client, and the videoid indicates the resource of which it gains ownership.

5. Errors

MVEX introduces one new error type.

Error	Description
`Video`	A value for a VIDEOIO argument does not name a VREFERENCE or provided by `QueryVideo`.

6. Requests

QueryVideo

wid: WINDOW
==>
screen: CARD8
video-depths: LISTofDEPTHS
allowed-depths: LISTofRENDERMODEL
inattr: LISTofVIDEOGEOMETRY
outattr: LISTofVIDEOGEOMETRY
video-input-models: LISTofBITMASK {`Pixmap, Window`}
video-output-models: LISTofBITMASK {`Pixmap, Window, Composite`}
clip-size: LISTofRECTANGLE
input-overlap: BOOL
capture-overlap: BOOL
io-overlap: BOOL
time: TIMESTAMP

Errors: `Window, Match`

This request returns information about video hardware for the screen associated with the specified window. The screen is returned as the first element in the reply. If there is no video hardware available for the screen, a `Match` error is returned.

The *video-depths* specify what depths/visuals are unique for video input and output. These are guaranteed to be different from those returned in the X connection setup; the list may be null; pixmaps are supported for each depth listed. Further, the presence of a MVEX extension in a server may cause the LISTofFORMAT provided by the connection setup to be extended with additional formats that would allow `GetImage` and `PutImage` access to windows or pixmaps created with depths and visuals not published in the connection setup. The definition of DEPTH is included in the core protocol's description of the connection information.

Visual ids found in *video-depths* having a core visual class (`PseudoColor`, `TrueColor`, etc.) imply that graphic requests with corresponding windows are expensive. Thus, it may be that pixels must undergo software translation before or after graphic requests. Depths and visual ids listed in *allowed-depths* that are selected from the X connection setup imply that they are not expensive, even though they may also be used for `RenderVideo` and `CaptureGraphics` requests.

The *allowed-depths* specify what depths and visual ids are supported for use with video input and video output. These depths and visual ids include those listed in video-depths plus appropriate ones selected from those provided by the X connection setup. The list is in no particular order. For each unique depth, at most one red-, green-, blue-mask will have non-zero values. This avoids ambiguity over which mask to use for a given depth when a client requests `RenderVideo` or `CaptureGraphics` with colormap None.

The *inattr* list the attributes of the decoder or digitizer used by the video inputs. The outattr list the attributes of the encoder used by the video outputs. The entries in both lists correspond one-to-one with the unique video input renderers and video output encoders; the

length of these lists imply the number of rows in video input models and video output models. `CreateVideo` references these resources by this implied index, starting from 0.

The list of video input models should be interpreted as a two dimensional array, column moving fastest and represents the relationship between the video input resources and the set of depths and visualids used to create video windows (VW). The columns are labeled left-to-right with the list of *allowed-depths*, and the rows are labeled top-to-bottom with the list of video inputs. Each cell is a bitmask containing zero or more true bits.

`Window` is asserted if a `RenderVideo` request may specify a window with the intersecting depth/visual as a destination and the intersecting video input as the source. If `Pixmap` is asserted at the intersection of an *allowed-depth* and a video input, then a `RenderVideo` request may specify a pixmap with that depth as a destination and that video input as the source (see `RenderVideo` for a discussion of pixmap pixel values). There is guaranteed to be at least one `Window` or `Pixmap` assertion in every row. In addition, there is guaranteed to be no more than one `Pixmap` assertion for every unique depth, per row: this avoids ambiguity for the RENDERMODEL use of *opaque* and the red-, green- and blue-masks.

If a window is created with a visual id whose type is `VideoGray` or `VideoColor`, then the window's pixels are always undefined. A `RenderVideo` will not change window's pixel values (although the picture may still be visible).

For example, the following array means that a VW for video input #1 must be depth 12, `TrueColor`; or depth 8, `VideoColor`; pixmaps of depth 24 can be used as a destination for a `RenderVideo` request. The #2 video input can only support pixmaps of depth 12.

		`VideoColor` depth 8	`TrueColor` depth 12	`None` depth 24
Video in	#1	`Window`	`Window`	`Pixmap`
	#2	0	`Pixmap`	0

The list of video-output-models should be interpreted as a two dimensional array, column moving fastest, and represents the relationship between the video output resources and the set of depths and visual ids that each may capture. The columns are labeled left-to-right with the list of *allowed depths*, and the rows are labeled top-to-bottom with the list of video outputs. Each cell is a bitmask containing zero or more true bits.

If `Window` is asserted, then a video output can capture the intersecting depth/visual in a window, including the border. If `Composite` is asserted, and the BLEND extension is present, and a window having the intersecting depth/visual is specified in a `CaptureGraphics` request, and the window is a member of a blend triple, then the window's contents will be captured after its primaries are blended. If `Composite` is asserted, then `Window` will be asserted for the same cell; The assertion of `Composite` is guaranteed to agree with the information returned by QueryBlend. If `Composite` is not asserted, but `Window` is asserted, then capture from a window having the intersecting depth/visual will always consist of unblended pixels from the source window and its descendants of the same depth, whether or not the window is a member of a blend triple. If a descendant is a member of a blend triple, the result is undefined: it is hardware dependent. If `Pixmap` is asserted, then a pixmap with the intersecting depth can be captured (see `CaptureGraphics` description of the interpretation of pixmap pixel values). There is guaranteed to be at least one `Window` or `Pixmap` assertion in every row. In addition, there is guaranteed to be no more than one `Pixmap` assertion for every unique depth, per row: this avoids ambiguity for the use of the red-, green- and blue-masks.

For example, if the video hardware has two video outputs, one able to capture depth 12, `TrueColor`, blended windows, and one able to capture depth 24 pixmaps, then the array

would look like:

		TrueColor depth 12	TrueColor depth 24
Video out	#1	Composite, Window	0
	#2	0	Pixmap

The *clip-size* list corresponds to the list of video inputs (*inattr*), and describes the smallest rectangle that can be used to clip a VIR partly occluded by other windows. If either the width or the height are 0, then the entire VIR must be displayed and cannot be clipped. This is meant as advisory for clients and window managers as they choose window placement.

If *input-overlap* is false, then VIRs may not overlap; true otherwise. *Capture-overlap* is false if the hardware does not allow VORs to overlap; true otherwise. *Io-overlap* element is false if the hardware does not allow a VIR and a VOR to overlap; true otherwise. Note that io-overlap may be true, but video-output-models determines if a VIR may be captured or not.

In all cases, if *clip-size*, *input-overlap*, or *io-overlap* constraints are violated, the content of the violated regions is undefined; if *capture-overlap* or *io-overlap* is violated, then the output signal for violated regions are undefined and is hardware dependent; no errors are returned to the offending request, but a `VideoViolation` event may be generated. In addition, other constraints that MVEX cannot express may be violated, and a `VideoViolation` event may be generated. Requests that may violate constraints are `RenderVideo`, `CaptureGraphics`, `MapWindow`, `UnmapWindow`, `MapSubwindows`, `UnmapSubwindows`, `ConfigureWindow`, `CirculateWindow`, `DestroyWindow`, DestroySubwindows, `ReparentWindow`.

The *time* element can be used to prevent race conditions between the delivery of a `VideoChange` event and the use of some other MVEX requests. The value of the timestamp is constant until there is a change in the list of video devices, and is then replaced with the current server time.

CreateVideo

id: VIDEOIO
reference: VREFERENCE

Errors: `Alloc`, `IDChoice`, `Video`

This request creates the specified identifier handle for the video input or output resource associated with *reference*. This resource is freed on connection close for the creating client. The number of identifiers created for a single client corresponding to a particular input or output is not limited to the concurrent-use for that resource as supplied by `QueryVideo`. However, the number of ids in use at once is limited by the concurrent use limit and ownership.

RenderVideo

source: VIDEOIN
destination: DRAWABLE
src-x, src-y, dest-x, dest-y: INT16
src-width, src-height, dest-width, dest-height: CARD16
full-motion: BOOL
priority: CARD8
subwindow-mode: {`ClipByChildren`, `IncludeInferiors`}
in, out: MARKER

Errors: `Drawable`, `Match`, `Value`, `Video`

The destination must be a valid drawable id, else `Drawable` error. The specified region of the source video signal is decoded and directed to the destination. The visible portion of

the destination rectangle becomes a VIR.

In general, it is intended that when a window of visual class `VideoGray` or `VideoColor` is the destination, the pixels of the video picture are inaccessible.

When a window is created with a visual id from a RENDERMODEL whose *opaque* member is true, then the intent is to say that digitized pixels are accessible, but they are not defined. When the destination is a window having a core visual class, the intent is that the video pixels are fully accessible, but happen to change "very frequently". The same is true for a pixmap depth (note that there is only one RENDERMODEL per depth; see `QueryVideo`).

If the destination is a window, then `Window` must be asserted in the *video-input-models* returned by `QueryVideo` for the intersecting visual/depth of the window and the specified video input, else `Match` error. If the window is visible, then the visible pixels are displayed. If the destination is an unmapped window, then the window's pixels may be rendered, depending on the value of backing-store for the window.

If the destination is a pixmap, then `Pixmap` must be asserted in the video-input-models for the intersecting *allowed-depth* of the pixmap and the specified video input, else `Match` error. In this case, if *opaque* is false, then the red-, green- and blue-mask define the meaning of the bits after a frame has been rendered. The source and destination must belong to the same screen, else `Match` error. If *opaque* is true, then the pixels are modified, but are undefined.

The src-x, src-y, src-width and src-height describe the extent of the source rectangle in the video picture. The dest-x, dest-y, dest-width and dest-height describe the rectangle in the destination drawable to receive the video picture. The source video picture will be scaled by

$$\text{x scale} = \frac{\text{dest−width}}{\text{src−width}} \qquad \text{y scale} = \frac{\text{dest−height}}{\text{src−height}}$$

If the source and destination rectangles and scaling factors do not match one of those provided in a single PLACEMENT for the specified video input source, a `VideoViolation` event may be generated. If the destination rectangle is valid, but some or all of it is outside the extent of the destination drawable, the result is clipped to a granularity advertised by `QueryVideo`, and a `VideoViolation` event may be generated. If the source rectangle is valid, but some or all of it is outside the extent of the video picture, the result in the corresponding region of the destination is undefined: it is hardware dependent.

The full-motion element determines whether the video input signal is continually updated or not. If it is true, then the server will try to achieve the maximum nominal rate advertised by the `QueryVideo` request for the specified scale factors and placement. If it is False, then a single still video frame will be rendered, as if the `RenderVideo` request was immediately followed with a `StopVideo` request specifying the same drawable. However, a full-motion element of False is an aid to the server to avoid unnecessary digitization.

Priority (0 to 100 inclusive, else `Value` error), represents the percentage of video bandwidth that the server should try to preserve. A value of 0 tells the server that graphic output to the screen should have top priority and that the video decoding may occasionally miss some incoming or outgoing frames. A value of 100 gives video decoding top priority meaning that the graphics performance may be degraded. It is advisory only and does not guarantee anything.

Subwindow-mode controls the treatment of the window's children. For `ClipByChildren`, the destination is clipped by all viewable children. For `IncludeInferiors`, viewable inferiors of the destination that overlap the VIR are updated with pixels from the video picture if the destination and its inferiors have a

matching visual/depth; otherwise, the result in the inferiors is undefined.

The in and out marker is used by `RenderVideo` to indicate a relative or absolute point when the request will start or stop. The request will wait indefinitely for the in marker to appear and will begin on that frame. The request will wait indefinitely for the out marker and will stop on the frame immediately preceeding. The in marker may be kind None, meaning that the request begins immediately; the out marker may be kind None, meaning that the request will only terminate in response to a `StopVideo` request or some other implicit stimulous listed below. If timecodes and frame numbers are not available, and an in or out point is specified it will behave as if the marker is never encountered.

If the destination window is clipped by siblings or children, a `VideoViolation` event may be generated if an overlap or clipping constraint is violated; see `QueryVideo`.

A `RenderVideo` will stop for a variety of reasons: after rendering one frame with fullmotion false, when a constraint has been violated, and when the environment changes. For the last, a `VideoOverride` event will be generated. Given a request specifying VIDEOIN V and drawable D, the following changes in the environment will cause the request to stop, and the `VideoOverride` event to be delivered:

- An explicit `StopVideo` request with drawable D and render.
- a change state of the window D such that it is no longer viewable (defined by the core protocol)
- a change in the size of the window D with bitgravity ForgetGravity
- destruction of D
- a connection close for the client that created V, thereby causing destruction of the identifier V
- another `RenderVideo` request specifying V as a source (note that for every VREFERENCE, there may be many VIDEOIN; only reuse of the same V will stop the earlier request)
- another `RenderVideo` request specifying D as a destination

If a client does not have ownership, but a `RenderVideo` request was successful, then the request is subject to being stopped under one other circumstance:

- the concurrent use for video input resource is exhausted and a client requests an additional resource by issuing a `RenderVideo` request; the new client need not have ownership

Clients that do not need access to digitized pixels should use a window having a `VideoGray` or `VideoColor` visual class as a destination. This allows the server to display the video picture using the most efficient means. With all visual classes, there is no guarantee that the signal is actually digitized.

If pixels are accessible (`Composite` set for the intersecting video input and render model; see `QueryVideo`), then `CopyArea`, `CopyPlane` and `GetImage` requests with a source region within a VIR are guaranteed to return a single frame with respect to the video signal. Note that the digitized picture may come from a signal whose frame is composed of fields, and that there may be temporal differences between fields. More device-specific information about the format of the picture may be available from one of the video input's controls.

If the specified window changes size while the request is still active, the destination x and y the original request will be adjusted to follow the bit gravity of the destination, but the width, height and scale are not adjusted. If the window changes size or root window coordinates, the result is clipped appropriately, and a `VideoViolation` event may be generated. In the case of ForgetGravity, a window size change will cause an implied `StopVideo` with an action of `Render` to be issued on the window.

Graphics may be drawn on a VW, but if the window is servicing an active `RenderVideo` request, the results are undefined.

CaptureGraphics

source: DRAWABLE
destination: VIDEOOUT
cmap: COLORMAP or `None`
src-x, *src-y*, *dest-x*, *dest-y*: INT16
src-width, *src-height*, *dest-width*, *dest-height*: CARD16
full-motion: BOOL
priority: CARD8
subwindow-mode: {`ClipByChildren`, `IncludeInferiors`}
in, *out*: MARKER

Errors: `Drawable`, `Match`, `Value`, `Video`, `Colormap`

The specified region of the source is encoded into a video signal by the destination encoder. If the source is a window, then `Window` must be asserted in the video-output-models returned by `QueryVideo` for the intersecting visual/depth of the window and the specified video output, else `Match` error. If the source is a pixmap, then `Pixmap` must be asserted in the video-output-models for the intersecting depth of the pixmap and the specified video output, else `Match` error.

If the source is a window, *cmap* must be `None` (else `Match` error), and the window's colormap or the installed colormap is used, depending on the video output hardware. If the source is a pixmap and the colormap is not `None`, then the visual id of the RENDERMODEL that intersects with the specified video output must be defined (else `Match` error), and the visual id of the colormap must match the RENDERMODEL (else `Match` error). The colormap defines the visual and colormap for the encoded picture. If the source is a pixmap and the colormap is `None`, then the interpretation of the pixel values is provided by the red-, green- and blue-mask for the RENDERMODEL. The source and destination must belong to the same screen, else `Match` error.

The src-x, src-y, src-width and src-height describe the extent of the source rectangle of the region captured. The dest-x, dest-y, dest-width and dest-height describe the rectangle in the destination video picture to receive the graphics. The source graphics region will be scaled by

$$\text{x scale} = \frac{\text{dest-width}}{\text{src-width}} \qquad \text{y scale} = \frac{\text{dest-height}}{\text{src-height}}$$

If the source and destination rectangles and scaling factors do not match one of those provided in a single PLACEMENT for the specified video output destination, a `VideoViolation` event may be generated. If the destination rectangle is valid, but some or all of it is outside the extent of the video picture, those pixels are clipped in the output video picture. If the source rectangle is valid, but some or all of it is beyond the extent of the source drawable, the result in the encoded video signal is undefined: it is hardware dependent.

The full-motion element determines whether the video output signal is constantly updated or not. If it is true, then the server will try to achieve the maximum rate advertised by the `QueryVideo` request for the specified scale factor and placement. If it is False, then a single still video frame will be captured. It is just as if the `CaptureGraphics` request was immediately followed with a `StopVideo` request specifying the same drawable; however, a full-motion element of False is an aid to the server to avoid extra encoding.

Priority is a value (0 to 100 inclusive, else `Value` error), representing the percentage of video bandwidth that the server should try to preserve. A value of 0 tells the server that graphic output to the screen should have top priority and that the video capture process may occasionally miss some incoming or outgoing frames. A value of 100 gives video capture

top priority meaning that the graphics performance may be degraded. It is advisory only and does not guarantee anything.

Subwindow-mode controls the capture of the window's children. If `ClipByChildren` is specified, the source is clipped by all viewable children only if `Composite` is not asserted in the video-output-models for the video output destination. If `Composite` is asserted, subwindow-mode will be effectively `IncludeInferiors`, regardless of its value.

If `IncludeInferiors` is specified for the subwindow-mode, then each visible region of an inferior of the source that overlaps the VIR may captured, but only if one or both of the following are true:

- The inferior is the same depth, visual and window type as the source window.
- `Composite` is asserted in the video-output-models for the video output destination.

Otherwise, the source rectangle is clipped.

If the source rectangle is clipped by siblings or children, the appearance of the clipped regions in the encoded picture are undefined: it is hardware dependent.

The in and out marker is used by `CaptureGraphics` to indicate a relative or absolute point when the request will start or stop. The request will wait indefinitely for the in marker to appear and will begin on that frame. The request will wait indefinitely for the out marker and will stop on the frame immediately preceeding. The in marker may be kind None, meaning that the request begins immediately; the out marker may be kind None, meaning that the request will only terminate in response to a `StopVideo` request or some other implicit stimulous listed in `RenderVideo`. If timecodes and frame numbers are not available, and an in or out point is specified it will behave as if the marker is never encountered.

Any time a `CaptureGraphics` is stopped because of an environment change, a `VideoOverride` event will be generated. See the description under `RenderVideo` for what constitutes an environment change.

If the specified window changes size while the request is still active, only the source x and y of the original request will be adjusted to follow the bit gravity of the source. If the specified window changes size or root window coordinates a `VideoViolation` event may be generated. If a portion of the new rectangle is beyond the extent of the source, the result in the output signal is undefined. In the case of ForgetGravity, a window size change will cause an implied `StopVideo` with an action of `Capture` to be issued on the window.

StopVideo

idlist: LISTofDRAWABLE
action: BITMASK {`Render, Capture`}
marker: MARKER

Errors: `Drawable, Value`

Each id must be a valid drawable, else `Drawable` error. If more than one id is in error, which one is reported is arbitrary.

The currently active renders and/or captures are stopped on all of the listed ids, even if one of the ids produces an error. If `Render` is specified, then `RenderVideo` requests on the listed ids are terminated. If `Capture`, is specified, then `CaptureGraphics` requests on the listed ids are terminated. Both may be specified. If neither `RenderVideo` or `CaptureGraphics` are currently active on a id, no error is reported.

The marker is used by `StopVideo` to indicate a relative or absolute point when the request will take effect: all comments about `StopVideo`'s actions refer to when it takes effect. The the request will wait indefinitely for the marker to appear and will take effect on

the frame immediately preceeding. The marker may be kind None, meaning that the request takes immediately. If timecodes or frame numbers are not available, and a marker is specified, it will behave as if the marker is never encountered.

If a `RenderVideo` on window W having a visual of type `VideoColor` or `VideoGray` is followed by a `StopVideo` on W, then the visible result on the screen is undefined, because it is hardware dependent. If a `RenderVideo` on a window W having a core visual class is followed by a `StopVideo` on W, the result in W is the pixels take on the value of the last frame acquired: they can be manipulated by other X requests.

In all cases, when it takes effect, this request will generate a `VideoOverride` event for each drawable that is servicing an active `RenderVideo` or `CaptureGraphics` request.

ChangeOwnership

owners: LISTofOWNER

Errors: `Window`, `Video`, `Match`

Note: this request is designed for the use of a video resource manager, not a client. Conventions should be developed such that a client designates interest in video resources, perhaps through properties, and the video manager allocates ownership as appropriate.

For each member of the list, the client that created the specified window is given ownership of the specified resource; if the client no longer exists, no action is taken. Owership for a video input or video output means that subsequent `RenderVideo` or `CaptureGraphics` requests are guaranteed not to be redirected, respectively. In addition, all controls associated with the resource may only be changed by the owning client. The number of clients given ownership of a single video input or output must not exceed the concurrent use limit for that resource as supplied by `QueryVideo`, else `Match` error. Note that a VIDEOIO specified in a OWNER is simply an alias for a VREFERENCE; a VIDEOIO created by any client may be used to specify the resource; and the window is used to designate the client-owner.

If a `RenderVideo` or `CaptureGraphics` request is issued for a resource, and one or more instances of the resource are in use by a client that does not have ownership, then a `StopVideo` request will be implied as described by `RenderVideo` and `CaptureGraphics`, regardless of whether the last issuing client has ownership for that resource or not. A `VideoOverride` event will be generated on the drawable. Under the same circumstances, if the issuing client does not have ownership, and all instances of the resource have ownership, the request will not take effect and a `VideoRequest` event will be generated with *redirect* set to TRUE.

If a video input or output is in use for less than its specified concurrent use limit, then a request requiring that resource will always succeed, regardless of ownership. Conversely, if a resource is utilized to its concurrent use limit, and the client utilizing the resource does not have ownership, and a client having ownership issues a request that requires the resource, then the owning client takes precedence, and a `VideoOverride` event will be generated, as appropriate.

Non ownership is achieved by designating the root window as the client-owner or by destroying the previously designated window.

SelectVideoEvents

id: VIDEOIO or DRAWABLE
mask: BITMASK
`{VideoRedirect, SyncNotify, OverrideNotify, ViolationNotify}`

Errors: `Drawable`, `Video`, `Access`, `Value`, `Alloc`

This requests selects events for the specified resource. Four of the masks are for drawables: `VideoRedirect`, `OverrideNotify` and `ViolationNotify`; the `SyncNotify` mask is for a VIDEOIO. If any of the drawable masks are included with any of the VIDEOIO masks, a `Value` error occurs. If the id is not a drawable for any of the drawable masks, a `Drawable` error occurs. If the id is not a VIDEOIO for `SyncNotify`, a `Video` error occurs.

Selecting `VideoRedirect` will allow a client to receive `VideoRequest` events generated on the screen of the specified drawable; only one client may select the event on a given screen, else `Access` error. After a client has selected the event, to maintain control, the client must assert `VideoRedirect` in every subsequent `SelectVideoEvents` request that specifies a drawable; the first request by the client specifying a drawable on the same screen without `VideoRedirect` asserted will release control. This selection is intented for use by a video resource manager.

Selecting `SyncNotify` allows the client to receive `VideoSync` events generated on the specified VideoIO. Note that VideoDevices do not generate `VideoSync` events.

Selecting `OverrideNotify` will allow client to receive `VideoOverride` events generated on the named drawable.

Selecting `ViolationNotify` will allow the client to receive `VideoViolation` events generated on the named drawable.

If any of the bits generates an error, which error is returned is not specified, and the state of selected events does not change.

QueryVideoAttributes

id: VIDEOIO
mask: BITMASK {Saturation, Contrast, Hue, Bright}
=>
setting: LISTofFRACTION

Errors: `Video`, `Access`, `Value`, `Alloc`

This request returns one or more of the saturation, contrast, hue or brightness attributes for a videoio. The bits in the bitmask determine how many fractions will be returned in the fraction list. To query one of the attributes, the resource's VIDEOGEOMETRY must have had a non-zero count of FRACTIONRANGES for the corresponding attribute; else `Value` error.

ChangeVideoAttributes

id: VIDEOIO
mask: BITMASK
{Saturation, Contrast, Hue, Bright}
setting: LISTofFRACTION

Errors: `Video`, `Access`, `Value`, `Alloc`

This request changes one or more of the saturation, contrast, hue or brightness attributes for a videoio. The bits in the bitmask determine how many fractions will be present in the fraction list. To set one of the attributes, the resource's VIDEOGEOMETRY must have had a non-zero count of FRACTIONRANGES for the corresponding attribute; else `Value` error. The FRACTION provided must be one described by the FRACTIONRANGE, else `Value` error. Settings are remembered for owning clients such that whenever an owner's `RenderVideo` or `CaptureGraphics` request is active using the set resource, the last setting made by that client is guaranteed.

7. Core Requests

`CreateWindow`

< this is a core protocol request >

If the MVEX extension is present, this request will understand two new visual classes: `VideoGray` and `VideoColor`. These are used to create one instance of a video window (VW). Visuals specified in a `CreateWindow` request must be one supported by the screen, and may be supplied by `QueryVideo`.

8. Events

Every MVEX event also contains the least-significant 16 bits of the sequence number of the last request issued by the client that was (or is currently being) processed by the server.

VideoChange

screen: CARD8
vio: VIDEOIO
state: {`Online, Offline`}
time: TIMESTAMP

This event is reported unconditionally to any client that has issued a `QueryVideo` request on the specified screen. It is generated when the named video resource has changed its availability; e.g., an existing video output becomes unavailable. If a resource moves from one screen to another, an event with state `Online` will be generated for one screen and an event with state `Offline` will be generated for the other. Issuing a `QueryVideo` request will yield a correct resources. New video identifiers are guaranteed not to have the same id as resources that were once listed, except for resources that go off line and return later: these resources will retain their previous id.

The time is what would be reported by the `QueryVideo` request.

VideoViolation

id: DRAWABLE
vid: VIDEOIO
action: ACTIONMASK
where ACTIONMASK is {`Scale, Placement, Clip, Overlap`}
state: {`Success, Subset, Fail`}
time: TIMESTAMP

This event is reported to the client selecting `ViolationNotify` on the drawable. The event is generated for the specified drawable and video input or output when scaling, placement, clipping and overlap constraints are violated. If state is `Success`, then the listed violations caused no change in the display or video output for the specified drawable, although some other drawable may be affected; If state is `Subset`, then some subset of the unobscured regions are still being displayed or captured; if the state is `Fail`, then the originating `RenderVideo` and `CaptureGraphics` will be cancelled (no `VideoOverride` event is generated). Note that for pixmaps, `Overlap` will never be asserted.

The time is the current server time.

VideoSync

id: DRAWABLE
vid: VIDEOIO
state: {`Acquired, Lost`}
time: TIMESTAMP

This event is reported to the client selecting `SyncNotify` on the vid.

It is generated following a `RenderVideo` request with the specified id and video input, whenever the sync is lost or gained. When selected, at least one `VideoSync` event is

guaranteed following a `RenderVideo` request, if one complete frame has been rendered before a `StopVideo` request is issued.

The event is also generated following a `CaptureGraphics` request with the specified id and video output, whenever the output signal begins to be produced. When selected, at least one `VideoSync` event is guaranteed following a `CaptureGraphics` request, if one complete frame has been generated before a `StopVideo` request is issued.

Note that if the original request specified fullmotion false, and the request succeeded, this will be the only event generated.

The time is the current server time.

VideoOverride

id: DRAWABLE
vid: VIDEOIO
time: TIMESTAMP

This event is reported to the client selecting `OverrideNotify` on the id. It is generated following a `RenderVideo` or `CaptureGraphics` request when there is a change in the environment. See `RenderVideo` and `CaptureGraphics` for a description of the conditions.

The time is the current server time.

VideoRequest

request: {`Render, Capture`}
redirect: BOOL
source: VIDEOIN or DRAWABLE
destination: VIDEOOUT or DRAWABLE
cmap: COLORMAP or `None`
src-x, src-y, dest-x, dest-y: INT16
src-width, src-height, dest-width, dest-height: CARD16
full-motion: BOOL
priority: CARD8
subwindow-mode: {`ClipByChildren, IncludeInferiors`}
time: TIMESTAMP
in, out: MARKER

This event is reported to the client selecting `VideoRedirect`. The request element indicates whether the request was `RenderVideo` or `CaptureGraphics`. Every `RenderVideo` or `CaptureGraphics` request using a resource for which the requesting client does not have ownership, and the concurrent use limit has been reached, and there are no non-owners using the resource, will generate a `VideoRequest` event with redirect set to true. Events are generated for requests that do not match this criteria with redirect set to false. This event is intended to be used by video resource managers.

The time is the current server time.