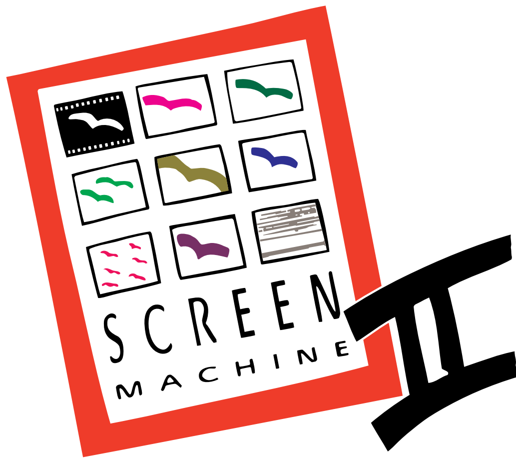
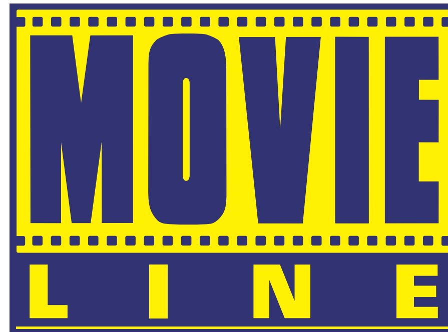
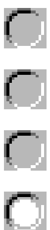
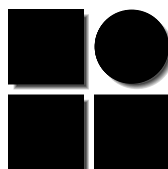




VIDEO Developer Kit

Reference Manual





interpersonal -
computing GmbH

NeXT Version 2.0

November 1995

The firm interpersonal computing GmbH reserves all rights in regard to the Video Developer Kit. The acquisition of the Video Developer Kit does not authorize the distribution of the kernel driver. Any licencing agreements for a kernel´s Runtime version are granted solely by interpersonal computing GmbH. Any failures to comply with the reserved rights will result in an immediate termination of all held licencing rights or agreements.

The interpersonal-computing GmbH reserves the right to change or vary in any way the kernel and/or the Kit´s functions without notice. The firm interpersonal computing GmbH assumes no responsibility or liability for any arising complications or incompatibilities as a result of these changes.

Screen Machine II® is a registered trademark of FAST Multimedia AG, München.
Movie Machine ® is a registered trademark of FAST Multimedia AG, München.
NeXT™, NEXTSTEP™ and NEXTIME™ are registered trademarks of NeXT Computer Inc., Redwood City.
Copyright© interpersonal-computing GmbH, 1993 München.

interpersonal-computing GmbH
NeXT Center München
Oettingenstraße 2
80538 München

Tel. : ++49 (0)89 22 33 75
Fax. : ++49 (0)89 22 33 76
e-Mail : sm@interpc.de (international <20 kB)

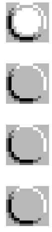
Contents

1 **Concepts** 5

- Introduction. 6
- The VDOboards 7
- The Display Details 11
- Things you should always do 13
- Split Memory Mode 14
- Integration Into NEXTSTEP 16
- Examples 17
- Animations 22

2 **Reference** 23

- Classes Overview. 24
- VDOboards. 26
- SMControl 30
- MMControl 55
- SMControlAudio 59
- SMControlChooserController 62
- SMWindow 64
- SMView 68
- SMPaletteView 75
- SMViewDragging 83
- SMYUVImageRep 84
- SMFLMImageRep 93
- MMFLMImageRep 101
- SMTVControl 105
- SMTV_AFC_Table 117
- SMTV_AFC_TableList 120



3 SMPalette 123
InterfaceBuilder 124
Palettes 125
Features 128
Building an Application 130

4 Appendix 133
Variable Types and Constants 134
The FLM file format 141
The YUV Color Model 144

Chapter 1

Concepts

Introduction

The Video Developer Kit is a collection of powerful objects bringing live video to the desktop and into your applications. Live video is what multimedia is all about. Where up to now it was only possible to have heterogeneous networks, e-mail and distributed objects, there are now the VDOboards, a suite of video overlay boards of the second generation, with dedicated special purpose video processors that result in a unrivaled picture quality; developed by FAST Multimedia AG a leader in video integration.

Together with NEXTSTEP, the VDOboards driver and the developer kit, the world of live video is at the developers fingertips. Any application is a candidate for live video now; where you used to integrate e-mail and drag&drop you can now place live video.

We have built a well integrated environment with a straight forward class hierarchy that makes development as easy as possible. Much care has been taken to make the integration into NEXTSTEP as seamless as possible.

The examples supplied only show a subset of the possibilities that the VDOboards offers. They are the best way to learn about the VDOboards capabilities. From then on the road is open to develop applications for picture analysis, security observations, video capture, documentation or just for watching CNN in a window for the latest events while trading bonds.

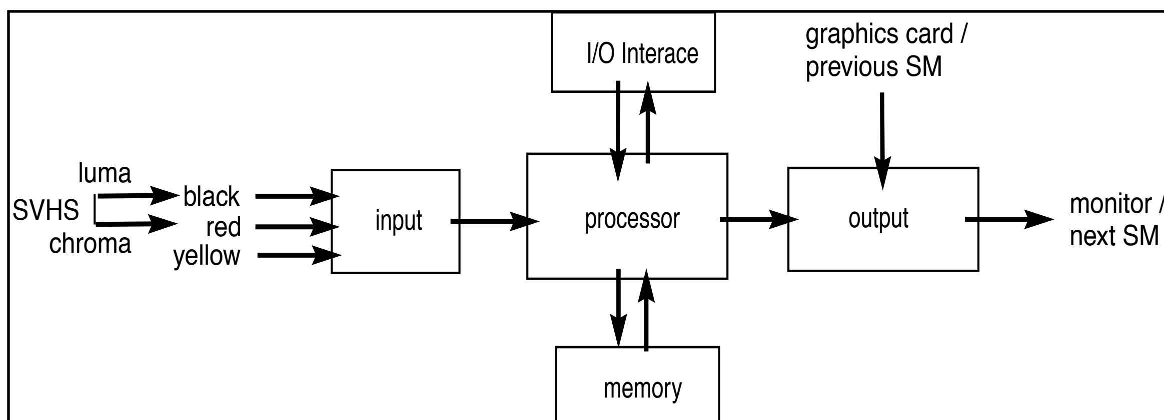
The VDOboards

One of the new features of the Relase 2.0 kit is that it now supports more than just the Screen Machine II. Here is a list of the boards supported by the VDOboards driver (current version with the release of this kit). All these boards support video overlay on graphic cards with a maximum resolution of 1280x1024.

Board	**Special Features**
Screen Machine II	1MB Video memory, only board capable of capturing both fields in full resoution. The video is digitized in 4:2:2.
Movie Machine	Inexpensive, includes Tuner & Videotext. (Not available in the US). The video is digitized in 4:1:1.
Movie Machine PRO	Supports video out, and has a builtin Tuner. The video is digitized in 4:1:1.
Movie Machine II	Supports Video out uses 4:2:2 versus the 4:1:1 of the MM PRO. A JPEG and MPEG option can be installed on the board. The boards supports mixing of input channels with special effects and alpha dissolves and wipes. Has built in Tuner & Videotext.
FPS 60	Has a JPEG option built in (compression & decompression) as well as video out. The MPEG option can be installed on this board. No special effects on video out. The video is digitized in 4:2:2.

NOTE: To use the options additional drivers are necessary. The options are not supported by this release of the kit. Contact us if compression board support is needed.

How the VDOboards work



In the input stage the video signal is converted into the digital YUV 4:2:2 (4:1:1 in the case of the MovieMachine and MovieMachine PRO) format. Video is transmitted in YUV, a format in which Y (the brightness) and UV (the color) are separated, unlike in the RGB or CMYK for-

mats. The final conversion for grabbed images from YUV to RGB is not handled by the board, but integrated into the SMYUVImageRep.
For a description of the YUV format see the Appendix of the YUV Image Format.

The conversion from the video signal to 4:2:2 YUV takes place at the rate of the incoming video, for PAL/SECAM at 50 fields and for NTSC at 60 fields per second. The type of conversion depends on the settings of the input converter. The processor then applies any scaling and filtering to the data and transfers it into the onboard memory.

The video processor also transfers the data from memory to the output stage, which mixes the video data with the graphics signal. The output stages handles features like brightness, saturation and contrast. This is why these settings will have no affect on the image in memory, and hence the grabbed images. (This is not entirely true, the SMFLMImageRep considers these parameters when converting from YUV to RGB). To retrieve data from the board the processor has a fourth interface to the I/O bus of the PC and this is the only bottleneck. It allows a maximum of 2-3 MB/s to be transferred, which is a far cry from the 20.8 MB/s needed for full frame rate of PAL, or the 17.6 MB/s for NTSC.

The I/O interface supports two modes for transferring images to and from a VDOboard. One is the so called I/O port access and is only 8 bit. The other is memory mapping (only supported by some chipsets when the system has more than 16 MB). Memory mapping is 16bit and suffers from a lot less overhead, increasing performance by a factor of 2 to 3.

How to use it

After correct setup and a connected video source everything is ready to be put into action. For a simple application we need only an instance of the control class.
In this application the video will be displayed somewhere on screen, outside a window :

```
#import <objc/Object.h>
#import <smkit/VDOboards.h>
#import <smkit/MMControl.h>
main(int argc, char**argv)
{
    NXRect aRect;
    char tmp[128], tmpChar;
    id control;

    // We need an instance of NXApp for the control class
    [Application new];

    // Get instance of control class
    control = [VDOboards
      newWithSelectionAndBoards:VMCBOARD_ALL_BOARDS
      withFeatures:VMCBOARD_ANY_FEATURES];
    //      Set input        Possible values are
    //      SM_INPUT_BLACK   SM_INPUT_RED
    //      SM_INPUT_YELLOW  SM_INPUT_SVHS
    [control setVideoInput:SM_INPUT_BLACK];
```

```
     // Set output window of 360 by 240 at 20.0,20.0
     NXSetRect(&aRect,20.0,20.0,360.0,240.0);
     [control setWindowFrame:aRect];

     // Turn video on
     [control setVideoOn:YES];

     // Wait for keypress
     gets(&tmpChar);

     // Turn video off
     [control setVideoOn:NO];

     // Free control & App
     [control free];
     [NXApp free];
}
```

With a simple loop you could animate the window:

```
for(i=20.0;i<100.0;i+=10.0)
{
     aRect.origin.x = i;
     [control setWindowFrame:aRect];
}
```

A more interesting feature are the readImage: and writeImage: methods. They allow you to capture the live video images or to display images in the live video window:

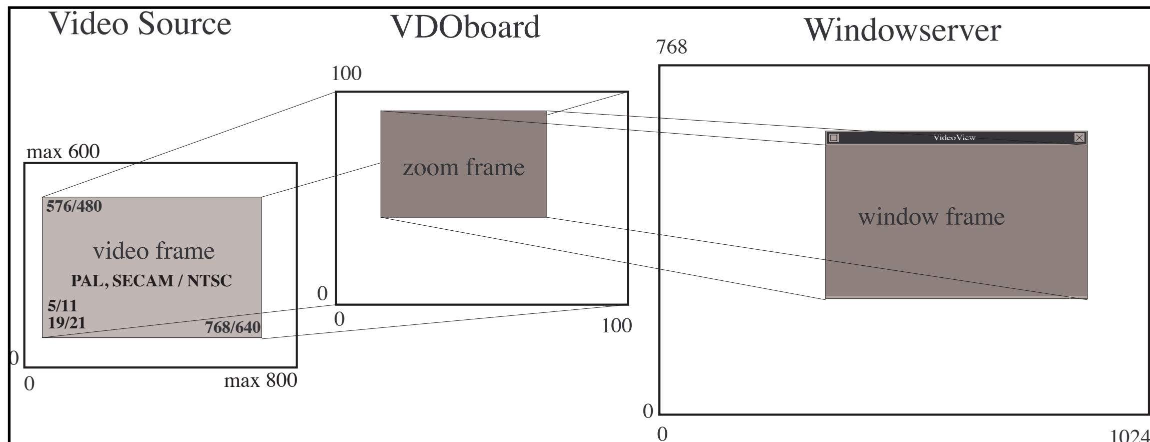
```
id   image;
NXStream *saveStream;

// Grab the current live picture as image
// Returns a newly allocated instance of NXImage
image = [control readImage];

// Open stream
saveStream = NXOpenMemory(NULL,0, NX_READWRITE);

// Write to stream
[image writeTIFF:saveStream];
NXFlush(saveStream);
NXSaveToFile(saveStream, "video.tiff");

// Free image
[image free];

// Close stream
NXCloseMemory(saveStream,NX_FREEBUFFER);
```

Here is some code to display an image in the video window:

```
NXStream *imageStream;
SMYUVImageRep * myImageRep;
NXImage *myImage;

myImageRep = [SMYUVImageRep alloc];
// Open stream (substitue MyImage.tiff with the name
// of your favourite picture.
imageStream = NXMapFile("MyImage.tiff", NX_READONLY);
if([myImageRep initTIFFDataFromStream: imageStream])
{
    myImage = [NXImage alloc];
    [myImage useRepresentation: myImageRep];
    [control writeImage: myImage];
    [myImage free];
}
// Close stream
NXCloseMemory(imageStream,NX_FREEBUFFER);
```

For a closer look at what can be done with the SM outside of NeXTSTEP take a look at the SM-Parser example. It uses almost all the features of the SMControl class, and together with its script capability, is quite a interesting tool.

The Display Details



Here is an overview of the three frames that can be set and how they determine the appearance on screen.

The video frame determines what portion of the digitized video gets written into the VDOboards memory. The maximum values depend on what system the input source is. Theoretically it is possible to set a selection of video to be displayed, although for this purpose it is better to use the zoom frame, unless VDOboards memory usage is a problem (see splitMemoryMode:). The SMControl class presets the video frame according to the specified system. If it is desired to digitize more or less of the bandwidth of the video, it is possible to change it with the setVideoFrame: method (This is best done in a subclass of SMControl in the setSystem: method). Although the theoretical maximum for PAL is 768 by 576, the default settings are 736 by 552 as most video sources do not have more bandwidth.

Here are the default values the SMControl class uses for the systems:

```
NTSC
x            =       5
y            =       11
width        =       640
height       =       480
horizontal range:0 - 650
vertical range:0 - 500

PAL/SECAM
x   =        19       // Offset to skip noise
y   =        24       // Offset to skip videotext
width=       736
height=      560
horizontal range:0 - 774
vertical range:0 - 590
```

The window frame sets the size at which the video is displayed on screen. The maximum size and position depend on the screen size. This is one of the major differences to the NeXT Dimension board, where the maximum size was determined by the type of board (PAL or NTSC). The live video is scaled to the size of the window frame. So live sizes from 16x8 to 1280x1024 are possible. If only a part of the video should be visible, there is the possibility to specify a zoom frame.

The zoom frame changes the portion of the video displayed on screen without having to change what is kept in memory. It is specified in percent of the window or video frame (doesn't make a difference). The portion of the video specified by the zoom frame is then mapped to the screen to the coordinates specified by the window frame.

To recap, the video frame is how much is digitized into VDOboards memory, the zoom frame is how much of that is displayed, and the window frame is where and how big it is displayed.

Things you should always do

To use live video of a VDOboard in your application there are at least two objects that you have to integrate. The first is a SMControl (or MMControl if you need video out) object for every VDOboard that you are going to use, and at least one SMView (or a subclass of it) for each SM-Control object to display the live video connected to the VDOboard. It is recommended to use a SMView or a subclass due to proper handling of the video size and position on screen.

The SMControl or MMControl object is allocated and initialized using factory methods of the VDOboards class, giving response whether or not any VDOboards are available (because other applications might already be using them) and asking if a VDOboard should be used anyway. Using the **+newWithSelectionAndBoards:withFeatures:** methods results in a panel being popped up, allowing the selection of a VDOboard if multiple boards are present.
To tell the SMControl which views it is going to handle, register every SMView to a SMControl by sending an addView: message to the SMControl. Any allocated and initialized SMView needs to know which SMControl it is using by sending a setControl: message to it. After that you can turn the video on by sending start: to a SMView.
If you turn video on by using the **setVideoOn:** method of the SMControl class, you should have specified one SMView to be the active view through a **setActiveView:** message sent to the appropriate SMControl.

Note: We do not recommend to start video display using a SMControl method if a SMView is used for display handling. Undesired effects may occur in further use.

Generally methods of the SMView should be preferred to those in the SMControl object, in which the view appearance is altered.
These first steps did nothing more than turn on video of a VDOboard controlled by a SMControl object in the bounds of a SMView.
There is one last thing you have to take care of. Every SMControl object has to be carefully freed before the application terminates, to make the VDOboard controlled by a SMControl object available for further use, and to remove the overlay from the screen.

NOTE: All instances of SMControl classes have to be freed using the **-free** method. It is not sufficient to use zone destruction.

Split Memory Mode

Unfortunately it gets more complicated from here. Due to the fact that on high-res screens (for theVDOboards this is anything above 800x600) the video can be displayed only at a reduced bandwidth. The grabbed images are also of a reduced bandwidth, although the Screen Machine II always digitizes the full bandwidth. To make it possible to display the video and grab images at full quality there is a special feature built into the SMControl class, the split memory mode. The trick is to split the 1 MB of memory on the board into two regions, one for the live image and one for the still or grabbed image. While viewing the video on screen the video is digitized into one part of the memory (from hereon referred to as the upper memory) of the board and displayed from there. When the picture is set to still or an image is grabbed, two fields of the live video are written into the other part (from here on referred to as lower part) of the memory, at the full bandwidth in the size desired. The limitations of this method are the amount of memory available. A full size PAL or SECAM frame is 864kB and NTSC 600kB, so for NTSC it is possible to store a full quality image in lower memory and store a half size image in upper memory for the live display (600kB + 300kB < 1Meg). For PAL/SECAM this is not possible, (864kB + 432kB > 1Meg). In this case trade-offs have to be made, either the full quality grabbed image has to be reduced in size, or the quality of the live picture reduced, to accommodate both in the SM memory. Even in split memory mode there are no limitations (up to the screen size) on how big the live (displayed) picture can be set. However it will tend to get coarse if size of the video in the upper memory region is significantly smaller than the displayed size.

The way it works is that you specify the size and the maximum width of the picture in the lower memory (size/width = height). With the setSplitMemGrabSize: call you can set the actual size of the picture to be grabbed. Setting sizes larger than the split memory region can result in unpredictable images. Setting grabSize wider than the maximum width, even if the total image would fit, also leads to unrecognizable images. The rest of the memory is used to display the video on screen.

The call to do this is setMemoryMode:mode offset:offset width:width.
The arguments are:
mode is SM_SPLIT_MEM to split the memory, or SM_MEM_NORM to reset.
offset is the start of the live video memory region (maximum width * maximum number of lines).
The value is in pixels, not bytes.
width is the maximum width of the grabbed image in the lower memory region.
This value has to be a multiple of 16. Usually this should be the value set with the -setVideoFrame: method. If you haven't set a video frame, query the active one with -videoFrame.

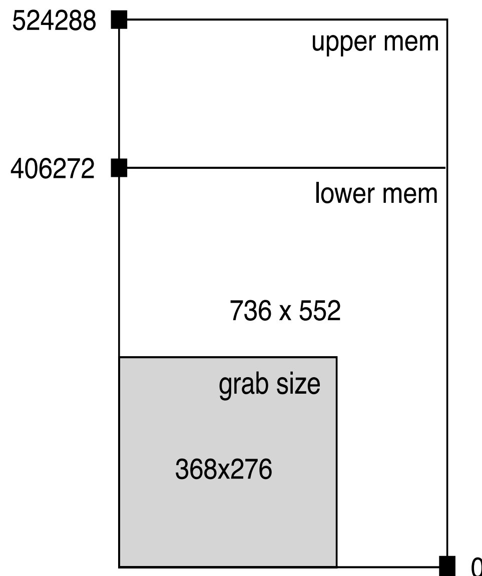
To recap, inorder to use the split memory mode you have to set the mode and divide the memory using the -setMemoryMode:offset:width : call. With this call you set the maximum size that is possible to be digitized in the lower memory. The actual size of the image is set with the **setSplitMemGrabSize:** call.

Example: Split memory mode for a PAL frame

```
// Set split mem bounds
//
// A PAL frame has a maximum size of 768x576. But we have a video frame
       of 736x560 set.
// And we want to be able to capture the complete frame, so we calculate
       the following
// value: 736*560 = 412160
[control setMemoryMode: SM_SPLIT_MEM  offset: 412160 width: 736]

// Set size of grabed image
//
// We now et the size of the image that will be digitize. The only
       restriction is
// the maximum size we determined for the setMemoryMode:offset:width:
       call.
// We want a half PAL frame = 368 x 276
// Width * height < offset
// The width has to be a multiple of 2
aSize.width = 368
aSize.height = 276
[control setSplitMemGrabSize: aSize];
```

You can also try this with the SMParser:

```
smp> mem 2 406272 736
smp> grabsize 368 276
```



(Sizes are in pixels, 1 pixel = 2 bytes)

Integration Into NEXTSTEP

So far we have described the basic functionality of the board, now lets focus on how to integrate the functionality in the existing NEXTSTEP environment. Ideally it should be possible to have the live video in a window, to drag around, miniaturize and hide it. For that purpose there are two more classes (in addition to the SMControl class) in the SMKit that handle just that and more. The dragging and miniaturizing is handled by the SMWindow class. It behaves just like a regular Window with a few extra features. These extra features are that it can handle more than one SMView (even if connected to different instances of the SMControl class), it can display the live video in the MiniWindow and can make resizing and miniaturizing smoother than the regular window class.

A SMView is a subclass of View that handles the resizing, printing and correct display of the video in a view. Additionally, it also enables drag&drop, dropping TIFF and FLM images on the view then will be displayed in the SMView, or dragging the stopped live video as a FLM,-TIFF or the FILENAME (Pastboardtypes) out of the view.

In conjunction with an SMView and a SMWindow, the SMControl class also makes clipping possible. Clipping is the process of masking out the parts of the video that are obscured by other objects. Noticeably, these are windows and panels that pop up, or are placed above the active video view.

The basic steps to create a NEXTSTEP application with live video from a VDOboard are to place a SMView in a SMWindow and connect the view to an instance of the SMControl or MMControl class. (For details take a closer look at the SMDemo example). Even easier is the use of the supplied SMPalette with its SMPaletteView. Just drag a SMPaletteView into a SMWindow, connect it to an instance of the SMControl class, connect a button to the **start:** method of the view and you have live video in the InterfaceBuilder test mode.

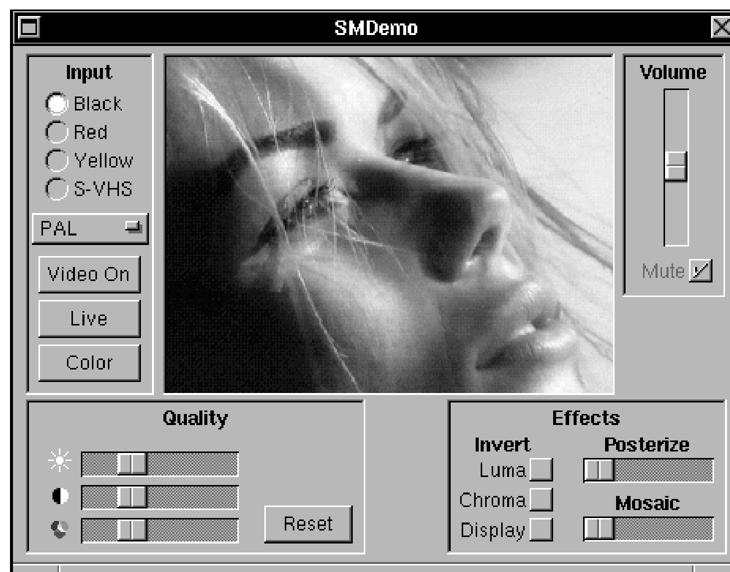
Examples

The source can be found in smkit/Examples.

1. SMDemo

SMDemo is a simple application with the following features :

• Uses SMWindow, SMView and SMControl
• Implements saving of images
• Allows various settings



The SMDemoControl.m file contains action methods for the various settings. For example, The radio buttons are connected to clickedInput: method.

```
- clickedInput:sender
{
    switch ([[sender selectedCell] tag])
    {
            case 0: [smCtrl setVideoInput:SM_INPUT_BLACK];
                    break;
            case 1: [smCtrl setVideoInput:SM_INPUT_RED];
                    break;
            case 2: [smCtrl setVideoInput:SM_INPUT_YELLOW];
                    break;
            case 3: [smCtrl setVideoInput:SM_INPUT_SVHS];
                    break;
    }
```

```
    return self;
}
```

The demo uses a SMView to display live video. For this we have to send the setVideoInput: messages directly to the SMControl object used for this SMView.

The most important thing to mention is to free any SMControl allocated in an application. The SMDemo takes care of that in the appWillTerminate: method.

```
- appWillTerminate:sender
{
    [smCtrl free];// it's very important to free the
                                  used SMControl class
                       // to free the sm for further use
    return self;
}
```

2. SMPaletteDemo

The SMPaletteDemo is the last example, and by far the simplest. All you need is about 10 minutes and the SMPalette.palette loaded into your InterfaceBuilder (and of course a VDOboard).

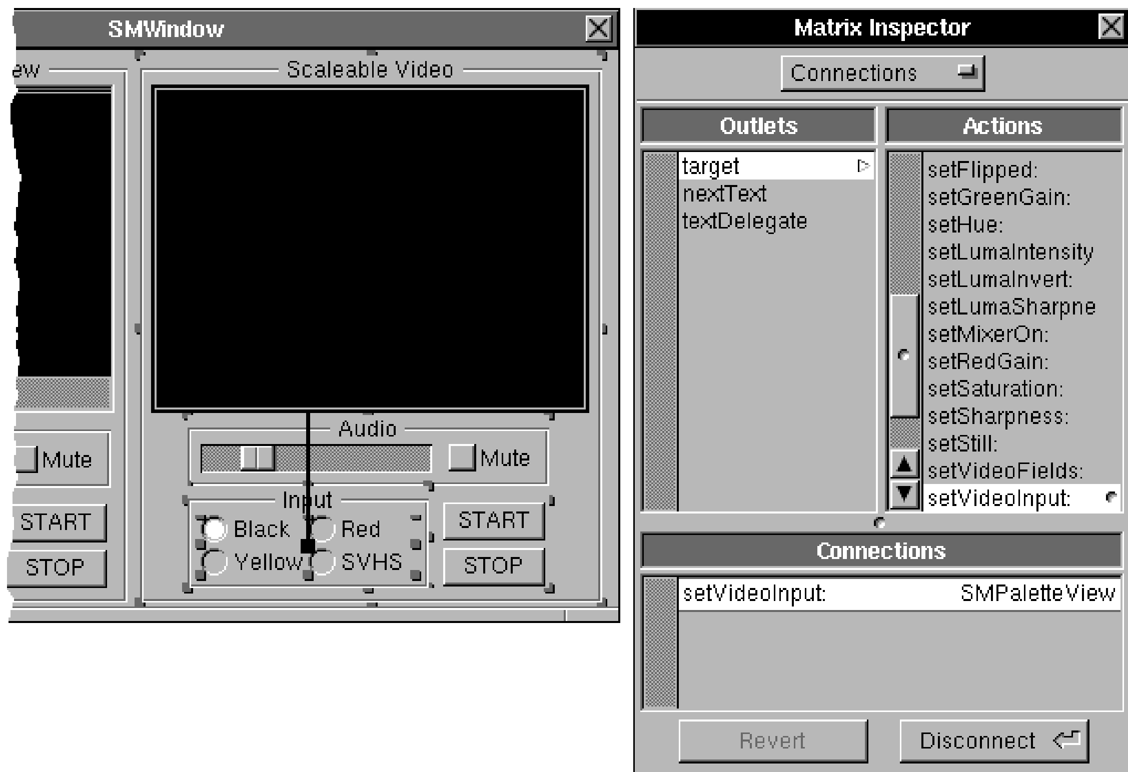
This is what we are set out to create.

Start a new application in InterfaceBuilder. (Command-n)

1. Delete the window supplied.
2. Drag an SMControl object from the SMPalette into the object suitcase.

2. Drag a SMWindow onto the screen.
3. Adjust its size to 498 by 338.
4. Drag a SMPaletteView into the window and size it to 640 by 480.
5. Drag a connection from the SMView to the SMControl object and connect the views's smControl outlet to the object:
6. Choose the SMView, bring up its Inspector (Command-1) and change its attributes to:

7. Group the view, change the title to 'No Title', offsets of zero and to a bezeld border.
8. Choose Size To Fit in Layout menu.
9. Group in ScrollView. Adjust the ScrollView to be just the size of the SMView.
10. Change size of ScrollView to 200 by 180.
11. Put a slider and a CheckButton (Call it mute) into the Window.
12. Drag a connection to the SMView and connect the target to the setAudioVolume: target method.
13. Drag a connection from the button to the view and connect the target to the setAudioMute: target method.
14. Group them and title it Audio.
15. Place it below the ScrollView.
16. Add a radio matrix of 2 by 2 to the window. Name them Black, Red, Yellow and S-VHS and assign them tags 0-3 respectively.
17. Drag a connection from the matrix to the SMView and connect the target to the setVideoInput: target method.



18. Group the matrix and title it Input. Place it below the Audio box.
18. Create two buttons and call them 'Start' and 'Stop'. Place them next to the Input box.
19. Drag a connection from the 'Start' button to the SMView and connect the target to the start: target method.
20. Drag a connection from the 'Stop' button to the SMView and connect the target to the stop: target method.
21. Group everything (ScrollView, Audio box, Input box and the two buttons). Title the group Video in a ScrollView.
22. Copy the complete group and place it next to it on the right side.
23. Rename it to Scalable Video.
24. Select the SMView and size it to 200 by 180
25. Bring up its Inspector (with Command-1) and change its attributes to:

scaled video ☑ grab on stop ☑ enable dragTo ☑ enable dragFrom ☑

26. Select the ScrollView and choose ungroup (Command-G) from the menu.
27. Adjust the view so it is placed correctly.
28. SAVE
29. You are ready. Hit Command-r to try it.

Note: For resizing to work correctly change the groups to be resizeable.

3. SplitIt

SplitIt is a application with the following features :

• Uses split memory mode
• Allows to change the spilt memory mode settings online

The boards of ///FAST use the YUV color space. The SM has a color depth of 4:2:2. This means that a sequence of 4 pixel has 4 Y (luminance), 2 U and 2 V (both chrominance) components, which leads to 8 components for a sequence of 4 pixel (= 2 components for 1 pixel). A 4:2:2 YUV color space is compared with a 24bit RGB color space. The YUV modes are described in Appendicies/A_.. and Appendicies/B_.. .

If you use higher screen resolutions (scaled switch is marked in Configure.app->setup) the SM scales the live video down to the half of its width. This has nothing to do width the memory the card has on board. This is only a problem of memory speed. Due to that fact the SM grabs only one half of the image information and the kit software doubles the width again (-> you get only the half image quality). One solution for that problem would be switching video scaling off just right before grabbing an image (setHorizontalScaled:NO). This will work quite well if you have set setVideoOn:NO before but does not solve the problem grabbing a already stopped video frame. In that case you have a "down scaled" image in the SM memory which can't be improved through scale switching.

Due to the reduced quality problem we integrated the split memory mode.

In split memory mode the memory is divided into two regions:
1. A "visible" region (upper mem) which contains the image data that is displayed on screen. If

you size the live video larger than the size of this region the display quality will get worth because the video will be scaled up to that size (same problem when sizeing the video larger than 640x480 in SM_MEM_NORM).
2. An "invisible" region (lower mem) in which the video data is composed in full quality (without scaling more than necessary to the recommended size). This happens whenever the card gets a grab or stop (still...) message. This "unscaled" data can be accessed using a readImage: method.

The split memory mode is designed to be used further on in the same application, when once set . You normally set it once at application startup. That's why you have to set the maximum width in a setMemoryMode::: call. The maximum width in combination with the offset specifies the maximum image size you can set (in setSplitMemGrabSize:) and request. Because of internal card memory management the maximum width has to be a multiplier of 16.

```
Calculate the offset :
     offset=(width*2) * height
```

This equals the way you would calculate the memory amount a RGB picture would take : rgb_mem=(width*3) * height. Remember (as described above) that in YUV 4:2:2 you have 2 Bytes per pixel.
In your case the maximum width shouldn't be more than 640 (appropriate for NTSC) and with a maximum height of 480 you get an offset of 614000. 434176 Bytes of memory are left for the upper mem. If you are using Hi-Res (horizontalScaled==YES) the SM needs only half memory size (as described above) for displaying live video. That means we can calculate the upper mem as:

```
     maximum_lossless_size=((width/2)*2)*height=width*height
     maximum_lossless_size<=434176=upper_mem
```

To retain the NTSC ratio = 640/480=4/3 we get :

```
     width*height<=upper_mem
     (height * (4/3))*height<=upper_mem
     height<=sqrt( 3/4 * upper_mem)
```

In this case :

```
     height=570 => width=760
```

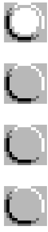
This is more than we need for displaying a NTSC signal.

When using horizontalScaled==NO we have to double the width for the calculation and we get :

```
     height=403, width=537
```

Do not misunderstand this result. You can size the display video larger but you get only image information for that size. For every larger size the video is scaled up (loss of information and quality implied).

Conclusion : The split memory mode offers the opportunity to grab images in different sizes and if horizontalScaled is used, in better quality than the live video on your monitor has. We know that this calculations are not easy to explain and to understand but for a further enhancement of the split memory mode (split the memory in more than 2 regions) it is necessary. We're thinking about the implementation of additional methods in the kit to make the handling easier. But even with "high level" methods you have to understand what's going on in the SM memory to get the results you want to have.



Animations

Since Steven Jobs has introduced NEXTIME at the NeXTWORLD EXPO in 1993, there has been a great push for video from the harddisk. The VDOboards driver is capable of both recording live video and playing back live video from harddisk, either in real time or at full resolution. Why not both? Well a 10 second video spot (NTSC which is 640x480 at 30 frames a second) would be 176 MB. This exceeds what an ISA bus can handle by an order of magnitude. Even if it would be possible, it isn't very desirable. Who has a 6 gigabytes for an hour of video? So the only reasonable solution is compression. For real time recording at the full resolution, the compression has to be on board. This is something we support with the CODY option for the ScreenMachine II and MovieMachine Series (a JPEG compressor/decompressor board). The compressed video is a factor of 10 to 20 times smaller. The problem with this kind of compression algorithm, is that it is asymmetrical. Meaning that it takes longer to compress the video than to decompress. This is why NeXT has developed a symmetrical algorithm based on wavelets. Not only is it faster, but the quality is also higher at the same rate of compression. Another nice feature is that the lossyness and hence the data rate can be adjusted during decompression, independent of the settings at compression time. This means that the video can be compressed at full quality and played back at a lesser quality on less powerful machines from the same set of data. Together with the CODY option and our VDOconverter productr you can record JPEG movies and transcode them into a wavlet compressed animation, capable of being played back using NEXTIME.

The next problem is to define a framework to play back the video with sound. Video and sound have to be synchronized and if the video falls back, frames have to be skipped and the frame rate or quality have to be adjusted. It has to define a file format, or a set of file formats supported.

NEXTIME will be just such a framework. It supplies a compression technique, allowing other compression algorithms to be integrated. It uses the Quictime file format, and can be extended to use others like AVI. But most importantly it handles the play back and recording with sound.

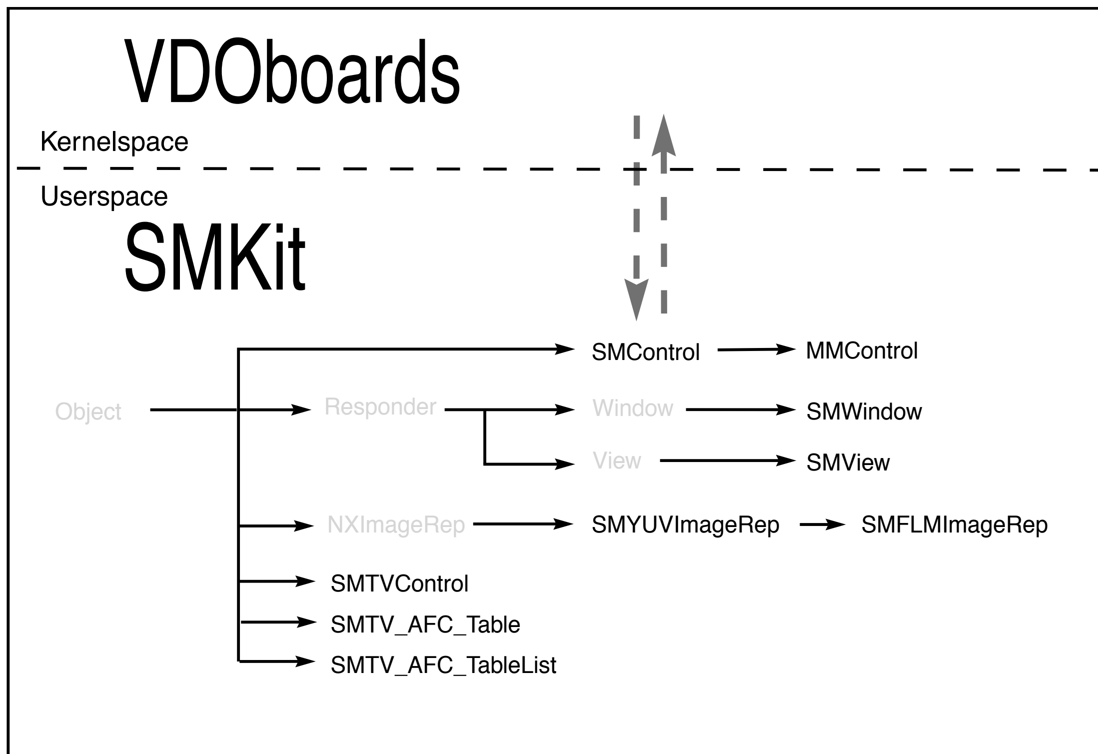
Now it would be possible to come up with a similar, possibly even compatible framework, but the time it would take to develop such a thing exceeds the time in which NEXTIME will hopefully become available from NeXT. When that happens, we will supply the adaptors and other software to integrate the VDOboards driver into NEXTIME. Until then it is not really possible to grab sequences of video and play them back from harddisk.

Latest word we have, is that the neccessary tools for NEXTIME will become available with NEXTSTEP 4.0

Chapter 2

Reference

Classes Overview



The Kernel / Driver
The kernel is a driverKit driver that is configured using the standard Configure.app. It is the low level connection to all boards. One kernel can service all installed boards. Access to the kernel is restricted to the SMControl class.

The SMControl class
The SMControl class is the connection to the board on the user level side. It controls all the features of the board as well as the access to the individual boards. There has to be one instance per connection to a board. This means if you have 3 Screen Machines II (of a maximum of 4) in your computer you have to have 3 instances of the SMControl class to use all 3 of them. It is not advised, though possible, to have more than one instance of a SMControl class control the same board (as in two separate applications) as this can lead to some unpredictable results. For the moment the Screen Machine is a non-shareable resource.

MMControl
The MMControl class extends the SMControl class to support the MovieMachine video out capabilities.

SMControlAudio
These methods control the optional audio hardware that can be connected to or be on board a VDOboard. This is either the optional Audio-on-Bracket, a TV-Tuner or the integrated audio of the Movie Machines.

SMControlChooserController
The SMControlChooserController is a simple way to let the user choose which VDOboard to use if multiple are present in the system.

SMWindow
SMWindow is a special subclass of Window that handles events necessary to ensure correct behavior when a window is moved, hidden or miniaturized. The window has two special features, one is that it allows the window to be dragged by the content view and the other that the icon of the window has the live video displayed in it.

SMView
SMView is a subclass of View that handles the display of live video on screen. It sets the position and size of the live video overlay and makes printing possible. It also implements drag&drop in the form that TIFF, EPS and FLM images can be dropped on the view, which will then be displayed in the video view by a video board.

SMPaletteView
SMPaletteView is a subclass of SMView that has a number of action methods corresponding to SMControl methods, allowing the direct connection to outlets in InterfaceBuilder. In this way it is possible to develop simple applications in IB without having to write a line of code.

SMPalette
The SMPalette is an InterfaceBuilder palette with a SMWindow, SMPaletteView and a SM-Control class. With it simple applications can be built in minutes and run just using Interface-Builder.

SMYUVImageRep, SMFLMImageRep, MMFLMImageRep
SMYUVImageRep + SMFLMImageRep are subclasses of NXImageRep that store the image data from the Screen Machine in YUV format. The SMFLMImageRep also saves the data in the FLM image format.

SMTVControl
SMTVControl is for the onboard tuners of the MovieLine or the optional tuner for the Screen-MachineII. It hast methods for tuning channels as well as reading predefined channel tables.

SMTV_AFC_Table
SMTV_AFC_Table is a class for holding afc tables for specific countries and television standards. This class can read the supplied afc tables (in Resources/AFCTables).

SMTV_AFC_TableList
SMTV_AFC_TableList is a class that read the supplied Countries.afcTable table and allows user selection by county and television standard.

VDOboards

Inherits From: **none**

Declared In: **<smkit/VDOboards.h>**

Class Description

This is an abstract class, as in that it has no instance methods. Its sole purpose is to return an initialized instance of the appropiate control class. (In the moment MM or SMControl). It also provides methods to query the VDOboards present. To make it possible to develop applications that are indpendant of a specific board present, we now support querying and initializing boards by a list of features. The VDOboards driver already supports over 5 different boards, with a substantial set of common functionality. In the ever changing world of PC hardware, this ensures that your application will probaly run on the newest board when it becomes available. So we strongly encourage you to use the feature flags instead of hardcoding your application to a specific board type.

Method Types

Factory methods	+ boardAvailable: + boardsPresentOfType:withFeatures: + closeAllBoards + getFeaturesOfBoard: + getKernelVersion:minor: + newForBoardNum: + newForBoardNum:fromZone: + newWithSelectionAndBoards:withFeatures: + newWithSelectionAndBoards:withFeatures:fromZone:

Class Methods

boardAvailable:

+ (BOOL)boardAvailable:(short)num

Returns YES if the VDOboard with the number num is available.

boardsPresentOfType:withFeatures:

**+ (unsigned int)boardsPresentOfType:(unsigned int)boardType
withFeatures:(unsigned int)features**

Returns the number of VDOboards present in the system that match the boardType and features. Possible values for boardTypes are:

VMCBOARD_ALL_BOARDS	If the type of board does not matter (use this together with the features flag).
VMCBOARD_MMPRO	For a Movie Machine PRO
VMCBOARD_SMII8	For a Screen Machine II
VMCBOARD_MM	For a Movie Machine
VMCBOARD_MMII	For a Movie Machine II
VMCBOARD_FPS60	For a FPS60 board.

The different board types can be ORed together for a greater selection.

Alternatively one can specify VMCBOARD_ALL_BOARDS and a list of features the board should support.

Possible flags for features are:

VMCBOARD_INPUTS	The minimum number of inputs the boards should have (0-7)<<8 i.e. 0x0300 for three inputs or more
VMCBOARD_OUTPUTS	The number of inputs the boards should have (0-7) << 16 i.e. 0x020000 for two outputs or more.

Additionally the type of each input or output can be specified

VMCBOARD_INPUT_SVHS	If any of the inputs supports SVHS
VMCBOARD_INPUT_NTSC	If any of the inputs support NTSC sources.
VMCBOARD_INPUT_PAL	If any of the inputs support PAL sources.
VMCBOARD_INPUT_SECAM	If any oft he inputs support SECAM sources.
VMCBOARD_OUTPUT_SVHS	If any of the outputs supports SVHS
VMCBOARD_OUTPUT_NTSC	If any of the outputs support NTSC sources.
VMCBOARD_OUTPUT_PAL	If any ofthe outputs support PAL sources.
VMCBOARD_OUTPUT_SECAM	If any of the outputs support SECAM sources.

And there is a list of special features each board supports.

VMCBOARD_AUDIO	That the board has audio output (usually from the Tuner). In case of the Screen Machine II this is the optional audio.
VMCBOARD_TUNER	That the board has a Tuner.

VMCBOARD_VT	If the board has Videotext (also referred to as teletext).
VMCBOARD_OVERLAY	If the board supports overlay (or inlay).
VMCBOARD_422	If the board uses 4:2:2 YUV sampling.
VMCBOARD_SQP	If the board has square vs CCIR 601 pixels.
VMCBOARD_FRAMES	If the board has enough memory to store full PAL frame instead of just fields.
VMCBOARD_ALPHA	If the board supports superimpsoing two video channels on video out with alpha. (Currently only supported by MM II)

The individual feature flags can be ORed together to select a more specific board.

If you want a specific board, specify VMCBOARD_ANY_FEATURES as the feature flag.

Example:

```
// Select any board that has at least 2 inputs and one output, a tuner
       and that uses 4:2:2 YUV sampling.

int num = [VDOboards boardsPresent:VMCBOARD_ALL_BOARD
     features:0x0200|0x010000|VMCBOARD_TUNER|VMCBOARD_422];

// Select any board that has at least 2 inputs and where one of the
       inputs support SVHS

int num = [VDOboards boardsPresent:VMCBOARD_ALL_BOARD
     features:0x0200|VMCBOARD_INPUT_SVHS];,
```

closeAllBoards

+ (void)closeAllBoards

Resets the usage count of all boards to 0. Essentially making all boards available again. The use of this methods should not be necessary, as the usage count is decermented automatically when a control class is freed, or when the PID that has last locked the board is not present anymore.

getFeaturesOfBoard:

+ (unsigned int)getFeaturesOfBoard:(short)num

Returns the features associated with the board number num.

getKernelVersion:minor:

+ getKernelVersion:(short*)*major* **minor:**(short*)*minor*

Returns the *major* and *minor* version of the VDOboards driver.

newForBoardNum:

+ newForBoardNum:(unsigned int)*i*

Returns a newly alloacted and initialized instance of the appropiate control class, if the board is present in the system. Regradless if the board is available or not. This is either an instance of the SMControl or MMControl class.

newForBoardNum:fromZone:

+ newForBoardNum:(unsigned int)*i* **fromZone:**(NXZone*)*zone*

Same a **+newForBoardNum**, except that the *zone* from which the control class is allocated can be specified.

newWithSelectionAndBoards:withFeatures:

+ newWithSelectionBoard:(unsigned int)*boardType*
withFeatures:(unsigned int)*features*

Returns a newly alloacted and initialized instance of the appropiate control class, according to the *boardType* and *features* specified. If more than one board os the requested type (and features) is present in the system, the user is presented with a modal panel, to select the board to use. If present in the same bundle as the calling class the SMMultiChoose.nib file is loaded from that bundle. If this .nib file is not present it will NOT display a panel and return the first available board instead.

See also: - boardsPresentOfType:withFeatures:

newWithSelectionAndBoards:withFeatures:fromZone:

+ newWithSelectionBoard:(unsigned int)*boardType*
withFeatures:(unsigned int)*features*
fromZone:(NXZone*)*zone*

Same as **+newWithSelectionBoard:withFeatures:**, except that the NXZone *zone* can be specified from which the instnce of the control class will be instantiated.

See also: newWithSelectionBoard:withFeatures:

SMControl

Inherits From: **Object**

Declared In: **smkit/SMControl.h**

Class Description

The SMControl class is the connection to the VDOboards driver. We have chosen not to document the driver interface to ensure compatibility even when the driver changes due to new operating system releases or performance tuning. Each instance of the class controls one VDOboard. The class provides access to all available functions of the board.

It can (among other things):

- Control the size and location of the video on screen.
- Control the appearance of the video on screen (color, brightness, contrast, ...).
- Control the optional audio hardware.
- Select the video input.
- Select TV system (NTSC, PAL, SECAM).
- Acquire single images or sequences from live video.
- Various digital effects (mosaic, posterization, chroma-keying, inverting, ...).
- Adjust the VDOboard overlay to the graphics system.

It is possible to write applications using just an instance of this class. To provide easier handling of the NEXTSTEP environment there are implementations of subclasses of the standard View and Window classes, as well as NXLiveVideoView compatible class. These are respectively the SMView and SMWindow classes. For further details, please refer to the documentation of the named classes.

Ranges and Default Values

			Min	Max	Neutral Off/Auto	Default	Stepsize	Comments
Output Stage (do not affect grabbed images)	Quality	Red gain	0.00	2.00	1.00	1.00	0.031	Adds or subtracts red from image
		Green gain	0.00	2.00	1.00	1.00	0.031	Adds or subtracts green from image
		Blue gain	0.00	2.00	1.00	1.00	0.031	Adds or subtracts blue from image
		Brightness	0.00	1.00	0.65	0.65	0.016	
		Saturation	0.00	1.00	0.67	0.65	0.016	
		Contrast	0.00	1.00	0.48	0.49	0.016	
		Hue	0.00	1.00	0.00	0.00	0.004	0.0 = 0˚, 1.0 = 360˚
		Sharpness	0.00	1.00	0.50	0.50	0.016	
InputStage (affect grabbed images)	Quality	Chroma intensity	0.00	1.00	0.50	0.52	0.010	Shifts lookup table for chroma values
		Luma intensity	0.00	1.00	0.50	0.46	0.010	Shifts lookup table for lume values
	Filters	Noise filter	0.00	1.00	0.00	0.00	0.250	Corrects for noisy input. (0 = Auto)
		Input filter	0.00	1.00	0.00	0.00	0.200	Corrects scaling of lines. (0 = Off, 0.2 = Auto)
		Bandpass	0.00	1.00	0.00	0.00	0.250	Enables bandpass filtering. (0 = Off)
		Prefilter	0.00	1.00	0.00	1.00	1.000	Enables prefiltering for a softer image
	Effects	Mosaic	1	32	1	0.00	1	Size of square which pixels are summed. (0 = Off)
		Posterization	0.00	1.00	0.00	0.00	0.143	Reduces ammount of distinct colors/shades in image
Audio	Volume		0.00	1.00	0.00	0.00	0.010	Volume
	Fader		0.00	1.00	0.50	0.50	0.010	Distribution front to back (0 = front, 1 = back)
	Balance		0.00	1.00	0.50	0.50	0.010	Distribution left to right (0 = left, 1 = right)
	Treble		0.00	1.00	0.50	0.50	0.010	
	Bass		0.00	1.00	0.50	0.50	0.010	

Instance Variables

```
port_t smPort;
NXRect videoFrame;
NXRect windowFrame;
NXRect zoomFrame;
NXSize grabSize;
short screenMachineNum;
List *viewList;
id activeView;
int port_address;
struct _Setup {
        short pll;
        unsigned char system:2;
        unsigned char interlaced:1;
        unsigned char scaled:1;
        unsigned char hasaudio:1;
        short xoffset;
        short yoffset;
} Setup;
struct _Mode {
        unsigned char mixer:1;
        unsigned char live:1;
        unsigned char video:1;
        unsigned char input:2;
        unsigned char color:1;
        unsigned char prefilter:1;
        unsigned char timebase:1;
        unsigned char invertChroma:1;
```

```
        unsigned char invertLuma:1;
        unsigned char flip:1;
        unsigned char frametype:2;
        unsigned char memory:2;
} Mode;
struct _Clipping {
        unsigned char isClipping;
} Clipping;
struct _ImageQuality {
        unsigned char red;
        unsigned char green;
        unsigned char blue;
        unsigned short brightness;
        unsigned char saturation;
        unsigned char hue;
        unsigned char contrast;
        unsigned char sharpness;
        unsigned char posterization;
        unsigned char bandpassfilter;
        unsigned char noisefilter;
        char inputfilter;
        char lumaintensity;
        char chromaintensity;
} ImageQuality;
struct _Image {
        char mode;
        char type;
        NXSize size;
} Image;
struct _Audio {
        unsigned char input;
        unsigned char mute;
        unsigned char volume;
        char balance;
        char fader;
        char treble;
        char bass;
} Audio;
```

smPort	Port to kernel server
videoFrame	Size of video acquisition
windowFrame	Size of displayed video on screen
zoomFrame	Size of crop frame relative to output
grabSize	Size of grabbed image
screenMachineNum	Number of VDOboard, the control class is acting on
viewList	List object with views connected to control class
activeView	Currently active view
port_address	Base address of boards for port IO
Setup.pll	Phased Lock Loop, controls width and position of video
Setup.system	System (PAL,NTSC,SECAM)

Setup.interlaced	Interlaced video
Setup.hasaudio	Audio present
Setup.xoffset	Horizontal offset of video
Setup.yoffset	Vertical offset of video
Mode.mixer	Mixer state
Mode.live	Video live state
Mode.video	Video on state
Mode.input	Current input
Mode.color	Color or BW display
Mode.prefilter	Input prefiltered
Mode.timebase	Timebase correction
Mode.invertChroma	Chroma inverted
Mode.invertLuma	Luma Inverted
Mode.flip	Picture upside down
Mode.frametype	Number and type of fields
Mode.memory	Split memory mode
Clipping.isClipping	Masking of windows
ImageQuality.red	Red color gain
ImageQuality.green	Green color gain
ImageQuality.blue	Blue color gain
ImageQuality.lumaintensity	Intensity of luma
ImageQuality.chromaintensity	Intensity of chroma
ImageQuality.brightness	Brightness
ImageQuality.saturation	Saturation
ImageQuality.hue	Hue
ImageQuality.contrast	Contrast
ImageQuality.sharpness	Sharpness
ImageQuality.posterization	Posterization
ImageQuality.bandpassfilter	Bandpass filter
ImageQuality.noisefilter	Noise filter
ImageQuality.inputfilter	Input filter
Image.mode	Format of image data
Image.type	Type of image rep
Image.size	Size of image
Audio.input	Audio input
Audio.mute	Audio muted
Audio.volume	Audio volume
Audio.balance	Audio balance
Audio.fader	Audio fader
Audio.treble	Audio treble
Audio.bass	Audio bass

Method Types

Init	- init - init: - reset - free - smNum - getKernelVersion: minor: - getSerial:
Setup	- setPll: - pll - setSystem: - system - setInterlaced: - isInterlaced - setHorizontalScaled: - isHorizontalScaled - setHorizontalOffset: - horizontalOffset - setVerticalOffset: - verticalOffset - setVideoFrame: - getVideoFrame: - setWindowFrame: - getWindowFrame:
Mode	- setInputFrameType: - inputFrameType - setVideoInput: - videoInput - setStill: - isStill - setMixerOn: - isMixed - setVideoOn: - isVideoOn - setColorOn: - isColor - setPreFilter: - isPreFiltered - setVCRTimebase: - isVCRTimebase - setChromaInvert: - isChromaInverted - setLumaInvert: - isLumaInverted - setFlipped: - isFlipped - inputType:

	- outputType: - numberOfInputs - numberOfOutputs
Imagequality	- setRedGain: - redGain - setGreenGain: - greenGain - setBlueGain: - blueGain - setLumaIntensity: - lumaIntensity - setChromaIntensity: - chromaIntensity - setBrightness: - brightness - setSaturation: - saturation - setHue: - hue - setContrast: - contrast - setSharpness: - sharpness - setBandpassFilter: - bandpassFilter - setNoiseFilter: - noiseFilter - setInputFilter: - inputFilter
Acquisition	- getImageSize: - readImage - writeImage: - readImageSelection: - setMemoryMode::: - memoryMode - setSplitMemGrabSize: - setImageType: - imageType
Effects	- setPosterization: - posterization - setMosaicWidth: - mosaicWidth - setZoomFrame: - getZoomFrame:
Audio (In SMControlAudio Category)	- hasAudio - setAudioMute:

- isAudioMute
- setAudioInput:
- audioInput
- setAudioVolume:
- volume
- setAudioBalance:
- balance
- setAudioFader:
- fader
- setAudioTreble:
- treble
- setAudioBass:
- bass

Instance Methods

bandpassFilter

- (float)**bandpassFilter**

Returns current value of bandpass filter, a float between 0.0 and 1.0 in steps of 0.25. The default is OFF (0.0).

See also: - setBandpassFilter:

blueGain

- (float)**blueGain**

Returns value of the blue gain in effect, a float between 0.0 and 2.0, in steps of 0.032, with a default of 1.0. A value of 1.0 is neutral and means that no alteration is in effect. Beware that values greater than 1.0 will cause the SMFLMImageRep to no longer accurately convert the images from FLM to TIFF.

See also: - setBlueGain:

brightness

- (float)**brightness**

Returns value of the brightness in effect, a float between 0.0 and 1.0, in steps of 0.016, with a default of 0.65. A value of 0.65 is neutral and means that no alteration is in effect.

See also: - setBrightness:

chromaIntensity

- (float)**chromaIntensity**

Returns value of the chroma intensity in effect, a float between 0.0 and 1.0, in steps of 0.010, with a default of 0.52.

See also: - setChromaIntensity:

contrast

- (float)**contrast**

Returns value of the contrast in effect, a float between 0.0 and 1.0, in steps of 0.016, with a default of 0.49. A value of 0.49 is neutral and means that no alteration is in effect.

See also: - setContrast:

free

- free

Frees the SMControl class and releases VDOboard for use with another SMControl class. It also unregisters all connected views. Since the SMControl is connected to hardware, it is important that the class is freed with this method, and not by the destroy zone mechanism used by the AppKit. To ensure proper handling call this method in the terminate: method of the Application class.

See also: - init,-init:,-initWithSelection

getImageSize:

- getImageSize:(NXSize *)*size*

Assigns *size* the proportions that an image would have if it would currently be read from the VDOboard.

See also: - setSplitMemGrabSize,-setWindowFrame

getKernelVersion: minor:

- getKernelVersion:(short *)*major* **minor:**(short *)*minor*

Sets the two variables passed to the *major* and *minor* revision of the kernel.
e.g. if major = 2 and minor = 4 the kernel revision is 2.4.

See also: + getKernelVersion:...

getSplitModeGrabSize:

- getSplitModeGrabSize:(NXSize *)*grabSize*

Assigns *grabSize* the currently set size for grabbed images in lower memory.

See also: - setSplitMemGrabSize:

getVideoFrame:

- getVideoFrame:(NXRect *)*theRect*

Assigns the currently active video frame to *theRect*.

See also: - setVideoFrame:

getZoomFrame:

- setZoomFrame:(NXRect *)*aRect*

Assigns *aRect* the currently active zoom frame. Returns self.

See also: - setZoomFrame:

getWindowFrame:

- getWindowFrame:(NXRect *)*aRect*

Assigns the currently active window frame to *aRect*.

greenGain

- (float)**greenGain**

Returns value of the green gain in effect, a float between 0.0 and 2.0 with a default of 1.0. A value of 1.0 is neutral and means that no alteration is in effect. Beware that values greater than 1.0 will cause the SMFLMImageRep to no longer accurately convert the images from FLM to TIFF.

See also: -blueGain:,-redGain,- setBlueGain:,- setGreenGain,- setRedGain

horizontalOffset

- (unsigned short)**horizontalOffset**

Returns the horizontal offset between the coordinate system of the VDOboard and the screen coordinate system.

See also: - setHorizontalOffset:

hue

- (float)**hue**

Returns the current value of hue in the range of 0.0 to 1.0 with a default of 0.0.

See also: - setHue:

imageType

-(char)**imageType**

Returns the type of ImageRep used by the **readImage:** and **readImageSelction:** methods. Possible values are SM_IMAGE_YUV and SM_IMAGE_FLM.

See also: - setImageType:,- readImage:,- readImageSelection:

init

- init

Looks for the next available VDOboard and initializes it to the default values. If no VDOboard is available it returns nil.

init:

- init:(short)*boardNum*

Initializes VDOboard with number *boardNum*. If the VDOboard with number *boardNum* is not available or not present it returns nil.

inputFilter

- (float)**inputFilter**

Returns current value of input filter in the range of 0.0 to 1.0 in steps of 0.200. A value of 0.0 means that automatic input filtering is enabled, a value of 0.200 that input filtering is disabled and anything above different levels of input filtering.

See also: - setInputFilter:

inputFrameType

- (unsigned char)**inputFrameType**

Returns currently active input frame type, one of SM_FRAME_BOTH, SM_FRAME_EVEN or SM_FRAME_ODD.

See also: - setInputFrameType:

inputType

- (unsigned int)**inputType:**(unsigned int)*inputNum*

Returns the type of input, input *inputNum* is, one of TYPE_FBAS, TYPE_SVHS or TYPE_F-BAS_SVHS. If a input is of type TYPE_FBAS_SVHS (i.e. supports both FBAS and SVHS) you select the FBAS input by setting input *inputNum*, and the SVHS input by setting *inputNum* + number of inputs the board has.

See also: - numberOfInputs

isChromaInverted

- (BOOL)**isChromaInverted**

Returns **YES** if input chroma values are being inverted.

See also: - setChromaInverted:

isColor

- (BOOL)**isColor**

Returns **YES** if color is enabled. Says nothing about whether color source is connected or if a color image is actually displayed. Returns **NO** if color is disabled.

See also: - setColorOn:

isFlipped

- (BOOL)**isFlipped**

Returns **YES** if vertical axis of image is flipped, otherwise **NO**; is mutually exclusive **with setMosaicWidth:**.

See also: - setFlipped:, - setMosaicWidth:

isHorizontalScaled

- (BOOL)**isHorizontalScaled**

Returns **YES** if horizontal scaling is active, otherwise **NO**.

See also: - setHorizontalScaled:

isInterlaced

- (BOOL)**isInterlaced**

Returns **YES** if interlaced input mode is selected. Otherwise returns **NO**.

See also: -setInterlaced:

isLumaInverted

- (BOOL)**isLumaInverted**

Returns **YES** if input luma values are being inverted.

See also: -setLumaInverted:

isMixed

- (BOOL)**isMixed**

Returns **YES** if mixer in the output stage is active, otherwise **NO**.

See also: -setMixerOn:

isPreFiltered

- (BOOL)**isPreFiltered**

Returns **YES** if input is prefiltered, otherwise **NO**.

See also: -setPreFiltered:

isStill

- (BOOL)**isStill**

Return **YES** if live video is halted, otherwise **NO**. Video is halted by **setStill:** or when an image is loaded into the VDOboard.

isVCRTimebase

- (BOOL)**isVCRTimebase**

Returns **YES** if timebase correction for VCRs is enabled, otherwise **NO**.

isVideoOn

- (BOOL)**isVideoOn**

Returns **YES** if live video is activated, otherwise **NO**. Says nothing about whether the correct input is selected or if a valid source is connected.

See also: -setVideoOn:

lumaIntensity

- (float)**lumaIntensity**

Returns value of the luma intensity in effect, a float between 0.0 and 1.0, in steps of 0.010, with a default of 0.46.

See also: -setLumaIntensity:

memoryMode

- (unsigned char)**memoryMode**

Returns currently active memory mode. One of SM_MEM_NORM or SM_MEM_SPLIT.

See also: -setMemoryMode:

mosaicWidth

- (short)**mosaicWidth**

Returns currently active mosaic width in the range of 1 to 16.

See also: -setMosaicWidth:

noiseFilter

- (float)**noiseFilter**

Returns current value of noise filter.

See also: -setNoiseFilter:

numberOfInputs

- (unsigned int)**numberOfInputs**

Returns the number of inputs the board has.

numberOfOutputs

- (unsigned int)**numberOfOutputs**

Returns the number of outputs the board has.

noiseFilter

- (float)**noiseFilter**

Returns current value of noise filter.

See also: -setNoiseFilter:

outputType:

- (unsigned int)**outputType:**(unsigned int)*outputNum*

Returns the type of output, output *outputNum* is, one of TYPE_FBAS, TYPE_SVHS or TYPE_FBAS_SVHS. If a output is of type TYPE_FBAS_SVHS (i.e. supports both FBAS and SVHS) you select the FBAS output by setting output *outputNum*, and the SVHS output by setting *outputNum* + number of outputs the board has.

NOTE: Most boards only have 1 output, or if they have 2 outputs then they are active simultaneously.

See also: - numberOfOutputs

pll

- (unsigned short)**pll**

Returns value for phased locked loop of output mixer in the range of 0 to 1024.

See also: -setPll:

posterization

- (float)**posterization**

Returns current value of posterization in the range of 0.0 to 1.0 in steps of 0.143. 0.0 means no posterization is in effect.

See also: -setPosterization:

readImage

- (NXImage *)**readImage**

Returns a newly allocated NXImage with a ImageRep representation of the current live picture displayed by the VDOboard. Whether the image contains a YUV or FLMImageRep is set by the **setImageType:** method. This is not a shared image, it is up to the application to free the returned NXImage.

See also: -readImageSelection:, -writeImage:

readImageSelection:

- (NXImage *)**readImageSelection:**(NXRect)*imageRect*

Returns a newly allocated NXImage containing a selection, specified by *imageRect*, of the current live picture displayed by the VDOboard. Whether the image contains a YUV or SM-FLMImageRep is set by the **setImageType:** method. This is not a shared image, it is up to the application to free this representation. The *imageRect* is specified in the window frame coordinate system, meaning that the lower left is the origin and that the maximum width and height are those of the current window frame.

See also: -readImage:, -writeImage:

redGain

- (float)**redGain**

Returns value of the blue gain in effect, a float between 0.0 and 2.0 with a default of 1.0. A value of 1.0 is neutral and means that no alteration is in effect. Beware that values greater than 1.0 will cause the **FLMImageRep** to no longer accurately convert the images from FLM to TIFF.

See also: -setRedGain,- setGreenGain, -setBlueGain:

reset

- reset

Resets values of the VDOboard to defaults.

See also: -init:,-init,-initWithSelection

saturation

- (float)**saturation**

Returns value of the saturation in effect, a float between 0.0 and 1.0, in steps of 0.016, with a default of 0.65. A value of 0.50 is neutral and means that no alteration is in effect.

See also: -setSaturation:

setBandpassFilter:

- setBandpassFilter:(float)*val*

Sets value of bandpass filter to *val*. The range is 0.0 to 1.0, in steps of 0.25. 0.0 turns bandpass filtering off.

See also: -bandpassFilter:

setBlueGain:

- setBlueGain:(float)*val*

Changes the output image blue component according to *val*. The range is from 0.0 to 2.0, in steps of 0.031, with 1.0 being neutral. A smaller value means less blue and a larger more blue in the picture. Has no effect on the image in memory of the VDOboard.

See also: -blueGain, -redGain, -greenGain, -setGreenGain:, -setRedGain:

setBrightness:

- setBrightness:(float)*val*

Sets brightness of output image according to *val*. The range is from 0.0 to 1.0 in steps of 0.016, with 0.65 being neutral. A smaller value means a darker picture and a larger a lighter picture. Has no effect on the image in memory.

See also: -brightness

setChromaIntensity:

- setChromaIntensity:(float)*val*

Sets the chroma intensity of the input decoder to *val*. A higher value means a more saturated representation, and a lower a colorless/grayscale representation. The range is from 0.0 to 1.0 - neutral is 0.50 and the default 0.52. Belongs to the set of controls affecting the input stage and hence how the image is digitized into memory.

See also: -setBandpassFilter:

setChromaInvert:

- setChromaInvert:(BOOL)*state*

If *state* is YES, the chroma values of the input will be inverted. Affects the image stored in memory. If *state* is NO, the input remains unaltered.

See also: -isChromaInverted

setColorOn:

- setColorOn:(BOOL)*state*

If *state* is YES, the chroma of the live video is enabled. For a color picture to appear, a color source has to be connected, the correct system has to be set, and the quality controls have to be set accordingly. (Notably the chroma intensity and saturation.) Another reason for a lack of color is if the black input is selected with a SVHS source connected. If *state* is NO, color is disabled. The video can be color regardless of the current WindowServer depth, as long as a color monitor is connected.

See also: isColorOn

setContrast:

- setContrast:(float)*val*

Sets contrast of output image according to *val*. The range is from 0.0 to 1.0 in steps of 0.016, with 0.49 being neutral. A smaller value reduces the contrast and a larger a higher contrast of the picture. Has no affect on the image in memory.

See also: -contrast

setFlipped:

- setFlipped:(BOOL)*state*

If *state* is set to YES, the vertical axis of the live picture will be flipped at the input stage. This results in the live picture and the image in memory being upside down. If *state* is NO, the image will remain unchanged. Do not use when mosaic is active, since it will produce unpredictable results.

See also: -isFlipped

setGreenGain:

- setGreenGain:(float)*val*

Changes the output image green component according to *val*. The range is from 0.0 to 1.0, with 0.5 being neutral. A smaller value means less green and a larger value more green in the picture. It has no affect on the image in memory.

See also: -greenGain, -blueGain, -redGain, -setBlueGain:, -setRedGain:

setHorizontalOffset:

- setHorizontalOffset:(unsigned short)*val*

Sets horizontal offset of the VDOboard coordinate system to coordinate system of the screen to *val*. It is used to align the two coordinate systems.

See also: -horizontalOffset

setHorizontalScaled:

- setHorizontalScaled:(BOOL)*state*

Sets input decoder to half the width of the video image. Reduces the bandwidth of picture by 50%. Used on system above a screen resolution of 800x600 to allow the PLL to set to the correct aspect ratio. Setting *state* to YES enables horizontal scaling, NO disables it.

See also: -isHorizontalScaled

setHue:

- setHue:(float)*val*

Sets hue of output image according to *val*. The range is from 0.0 to 1.0, with 0.0 being neutral. 0.0 corresponds to 0 change and 1.0 a 360 change in the color angle.

See also: -hue

setImageType:

- setImageType:(char)*type*

Sets *type* of ImageRep used by the **readImage:** and **readImageSelection:** method. Possible values are SM_IMAGE_YUV and SM_IMAGE_FLM. Returns **self**.

See also: -imageType, -readImage, -readImageSelection:

setInputFilter:

- setInputFilter:(float)*val*

Sets value of input filter to *val*. Range is from 0.0 to 1.0, where 0.0 is the default. A value of 0.0 turns automatic input filtering on.

See also: - inputFilter

setInputFrameType:

- setInputFrameType:(unsigned char)*val*

Sets input frame type to *val*. Possible values are SM_FRAME_BOTH, SM_FRAME_EVEN, SM_FRAME_ODD. With the input frame, one can select if the image is generated only for the odd or even fields, or from both fields. Both fields result in a better image resolution, but, on images with a lot of movement, it leads to stripes in the image caused by movement in the image between fields. For pictures with fast movement, one should select odd or even fields.

See also: -inputFrameType

setInterlaced:

- setInterlaced:(BOOL)*state*

Sets input decoder to interlaced mode. Effectively halves the bandwidth by halving the height of the video image in memory. Only useful for interlaced video systems, which are currently not supported by NEXTSTEP. If *state* is set to YES, interlaced mode is selected.

See also: - isInterlaced

setLumaIntensity:

- setLumaIntensity:(float)*val*

Sets the luma intensity of the input decoder to *val*. A higher value means a brighter representation and a lower a darker a representation. The range is from 0.0 to 1.0, neutral being 0.5 and the default 0.46. Belongs to the set of controls affecting the input stage and hence how the image is digitized into memory.

See also: - lumaIntensity

setLumaInvert:

- setLumaInvert:(BOOL)*state*

If *state* is YES, the luma values of the input will be inverted. Affects the image in memory of the VDOboard. If state is NO, the input values will remain unchanged.

See also: - isLumaInverted

setMemoryMode:

- setMemoryMode:(char)*mode* **offset:**(long)*offset* **width:**(short)*width*

Sets memory mode of the VDOboard to *mode*, where *mode* is one of SM_MEM_SPLIT or SM_MEM_NORM. When mode is SM_MEM_NORM parameters *offset* and *width* are ignored.
e.g. [control setMemoryMode:SM_MEM_NORM offset:0 width:0];

For the SM_MEM_SPLIT mode, *offset* and *width* specify the amount of memory set aside for grabbing high quality images independant of the size of the live video view.

offset and *width* are in pixels. *width* should be set to the maximum width that the images will have. (Set by **setSplitMemGrabSize:**). *width* has to be a multiple of 16 and has to be in the range of 16 < *width* < videoFrame width.
offset = *width* * maximum height of grabbed images.

The Screen Machine II has room for 524288 pixels in memory (other boards: 262144). So the offset has to be less than 524288. Also keep in mind that the Screen Machine II requires some memory to display the live video. The remaining memory 524288 - offset will be used for this purpose. If the offset is set to high the live video quality will degrade noticeably.
See the Concepts section of the manual for more details on the split memory mode.

Note: At the moment SM_MEM_SPLIT only works on Screen Machine, FPS60 and Movie Machine II boards.

See also: - setSplitMemGrabSize:

setMixerOn:

- setMixerOn:(BOOL)*state*

If *state* is YES the mixer of the output stage will be active, resulting in the data from the graphics card being mixed with the data from the live video. Live video will be displayed where the graphics are black, and no live video where the graphics are white. Useful if graphics should be visible without clipping. If *state* is set to NO, the live video data takes precedence, meaning that the graphics data is discarded at the corresponding coordinates. Useful if the video should cover the graphics, without having to change window order.

Note: When the mixer is turned off, the cursor will not be visible in those places where the video is displayed.

See also: - isMixed

setMosaicWidth:

- setMosaicWidth:(short)*width*

Set mosaicking scale to *width*. Mosaicking causes *width* pixels to be summed and displayed as a single pixel. Effectively reduces the resolution of the displayed image. Only works on output stage; has no affect on image in memory. Do not use if video is flipped using the **setFlipped:** method.

See also: - mosaicWidth, - isFlipped, - setFlipped:

setNoiseFilter:

- setNoiseFilter:(float)*val*

Sets value of input noise filter to *val*. The Range is from 0.0 to 1.0 in steps of 0.25. The default is 0.0.

See also: - noiseFilter:

setPll:

- setPll:(unsigned short)*val*

Sets the phased locked loop of the SM to *val*. Valid range is from 0-1024. Realistic values range from 500-800. This value controls the width and horizontal scale of the live video picture. It acts solely on the output stage of the SM; it has no effect on the digitized images. Setting to extreme values can cause the picture to collapse (just stripes of distorted video on screen). On a system with a resolution above 800x600 the horizontal scale of the input has to be halved by setting **setHorizontalScaled:** to YES, to achieve the correct aspect ratio of the live video on screen. This is usually done in the Configure.app which stores it values in the Instance0.table in

./usr/Devices/SMDriver.config. See appendix for example data.

See also: - pll

setPosterization:

- setPosterization:(float)*val*

Sets posterization of input image according to *val*. The range is from 0.0 to 1.0. 0.0 disables posterization. A larger value reduces the amount of distinct shades of color.

See also: - posterization

setPreFilter:

- setPreFilter:(BOOL)*state*

If *state* is YES, the input source is prefiltered, otherwise NO.

See also: - isPreFiltered

setRedGain:

- setRedGain:(float)*val*

Changes the red component in the output image according to *val*. The range is from 0.0 to 1.0, with 0.5 being neutral. A smaller value means less red and a larger value more red in the picture. Has no affect on the image in memory.

See also: - redGain, - blueGain, -greenGain, - setBlueGain:, -setGreenGain:

setSaturation:

- setSaturation:(float)*val*

Sets saturation of output image according to *val*. The range is from 0.0 to 1.0 in steps of 0.016, with 0.5 being neutral. A smaller value means a less saturated and a larger a more saturated picture. Has no affect on the image in memory.

See also: - saturation

setSharpness:

- setSharpness:(float)*val*

Sets sharpness of output image according to *val*. The range is from 0.0 to 1.0, with 0.5 being neutral. A smaller value reduces the sharpness and a larger increases the sharpness of the picture. Has no affect on the image in memory.

See also: - sharpness

setSplitMemGrabSize:

- setSplitMemGrabSize:(NXSize)*grabSize*

Sets size of the following read images from the VDOboard to *grabSize*. Only works in SM_SPLIT_MEM mode. *grabSize* should not exceed the available lower memory, reserved for the images to be read, on the board.

See also: - readImage, - readImageSelection:

setStill:

- setStill:(BOOL)*state*

If *state* is YES, the live video is halted. If the split memory mode is active, a frame is written into the lower part of memory before halting the live picture. If *state* is NO, the picture returns to the live mode. Results in a one frame offset between picture in lower and live picture (upper memory) if split memory mode is active. Setting to YES disables the writing of video data into memory. This means that the controls on the input side have no effect while set YES. These are all the filters and luma and chroma intensity.

See also: - isStill

setSystem:

- setSystem:(unsigned char)*val*

Sets system of input decoder to *val*. Possible values are SM_SYSTEM_PAL, SM_SYSTEM_-SECAM and SM_SYSTEM_NTSC. This also affects the current videoframe, which is reset to the maximum resolution according to the system. These values are 756x567 for PAL and SE-CAM and 640x480 for NTSC.

See also: - system

setVCRTimebase:

- setVCRTimebase:(BOOL)*state*

If *state* is YES, timebase correction is applied. Useful if mechanical video sources like video recorders are connected. If *state* is NO, timebase correction is disabled.

See also: - isVCRTimebase

setVerticalOffset:

- setVerticalOffset:(unsigned short)*val*

Sets vertical offset of VDOboard coordinate system to coordinate system of the graphics sub-system to *val*. Used to align the two coordinate systems. Usually set by the Configure.app, the value is stored in the Instance0.table and read by the control class at initialization and reset.

See also: - verticalOffset

setVideoFrame:

- setVideoFrame:(NXRect)*theRect*

Set the video frame to *theRect*. The video frame is the size and position at which the video image from the input decoder gets transferred into memory. The video frame is usually set by the **set-System:** method to the currently active system and its size. The offset is usually set to skip invalid or noisy data at the beginning of a field.

See also: - getVideoFrame:

setVideoInput:

- setVideoInput:(unsigned char)*val*

Sets active input source for the VDOboard to *val*. Possible values are non negative integers starting with 0. The range of values depend on the VDOboard you are using. For Screen Machine and SM_INPUT_SVHS the special connector cable has to be used, with luma connected to the black input and chroma to the red input. If an audio device is connected, it also automatically activates the according audio channel.

See also: - videoInput, - numberOfInputs, - inputType:

setVideoOn:

- setVideoOn:(BOOL)*state*

If *state* is YES, a live video will be displayed on screen. If *state* is NO, the live video will be disabled and removed from the screen. Whether an actual picture is displayed depends on if there is a valid video signal at the selected input, and if the quality adjustments are set correctly. The overlay has to be adjusted with the system function for a correct output as well. Setting video to off does noticeable disable the writing of the video data into memory; use the **setStill:** method to prevent new data from being written. It is possible to grab video without it being displayed on screen.

See also: - isVideoOn

setWindowFrame:

- setWindowFrame:(NXRect)*theRect*

Sets window frame to *theRect*. The window frame is the size and position of the live video image on screen. The minimum size is 16x1; maximum size is the current screen size. The window frame affects the size of the image in memory if it is smaller than the video frame.

See also: - getWindowFrame:

setZoomFrame:

- setZoomFrame:(NXRect)*theRect*

Sets zoom frame to *theRect*. The zoom frame is the portion of the video in memory of the VDOboard that is displayed on screen in the currently active video frame. The zoom frame is specified in percent from 0.0 to 100.0, and therefore independent of the current window and video frame.

See also: - getZoomFrame:

sharpness

- (float)**sharpness**

Returns current value of sharpness.

See also: - setSharpness:

smNum

- (short)**smNum**

Returns number of the VDOboard used by the SMControl class instance. Returns -1 if no VDO-board is assigned.

system

- (unsigned char)**system**

Returns currently set system, one of SM_SYSTEM_PAL, SM_SYSTEM_SECAM or SM_-SYSTEM_NTSC

See also: - setSystem:

verticalOffset

- (unsigned short)**verticalOffset**

Returns the vertical offset between the VDOboard coordinate system and the screen coordinate system.

See also: -setVerticalOffset:

videoInput

- (unsigned char)**videoInput**

Returns the currently active video input, a positive integer starting with 0. The range of values depend on the VDOboard you are using.

See also: - setVideoInput:, - numberOfInputs, - inputType:

writeImage:

- writeImage:(NXImage *)*anImage*

Displays *anImage* in the view of the VDOboard. The image is scaled to the current view dimensions. Used to preview images in 24 bit, independent of graphics card. *anImage* has to contain a SMYUVImageRep or SMFLMImageRep representation.

See also: - readImage, - readImageSelction:, SMYUVImageRep**,** SMFLMImageRep

MMControl

Inherits From: **SMControl**

Declared In: **<smkit/MMControl.h>**

Class Description

This class extends the SMControl class for those VDOboards that support video out. In particular this is the MovieMachine PRO, MovieMachine II, and to a limited amount the FPS60. The MovieMachine PRO supports video out with one background channel that is of fixed size and has to have the same video system as the input source (the input itself can be different). The additional foreground channel is scalable, supports loading of images and with the optional CODY board, playback of live video form harddisk . The Movie Machine II is identical to the Movie Machine PRO, except that it has a better quality (4:2:2) and supports alpha compositing of the foreground and background channel. The FPS60 is different in that it has no foreground channel.

Instance Variables

Method Types

- setVideoOut:
- setVideoOutInput:
- setVideoOutInvert:
- setVideoOutPattern:
- setVideoOutScale:
- setVideoOutSystem:
- setVideoOutXofs:
- setVideoOutYofs:
- videoOutInput
- videoOutSystem
- videoOutXofs
- videoOutYofs
- writeImage:

Instance Methods

setVideoOut:

- setVideoOut:(BOOL)*state*

If *state* is YES then the overlay will be disabled and displayed on the video out channel instead. If *state* is set to NO the foreground channel is disabled on video out and displayed in the overlay again. The mixer settings (**- setMixerOn:**) has no effect on video out.

See also: - setVideoOn: (SMControl)

setVideoOutInput:

- setVideoOutInput:(short)*nInp*

Selects the input for the background channel on video out. Possible values are VIDEO_OUT_INPUT_A, VIDEO_OUT_INPUT_B, VIDEO_OUT_INPUT_TV or VIDEO_OUT_INPUT_NONE

See also: -videoOutInput, - setVideoInput: (SMControl)

setVideoOutInvert:

- setVideoOutInvert:(BOOL)*bInv*

If *bInv* is set to YES, the vertical axis of the live picture will be flipped at the input stage. This results in the live picture and the image in memory being upside down. If *bInv* is NO, the image will remain unchanged.

See also: - setFlipped (SMControl)

setVideoOutScaled:

- setVideoOutScaled:(BOOL)*state*

Controls how the foreground channel is displayed when the input and outputs systems differ(e.g input video system is NTSC and the output system is PAL) . This is useful when converting between video standards. If *state* is set to YES the forground channel is scaled to fit the out system (i.e PAL is shrunk if output is NTSC and NTSC is enlarged if output is PAL)

See also: - setVideoOutSystem, - setSystem: (SMControl)

setVideoOutSystem:

- setVideoOutSystem:(short)*nSys*

Sets video system of output background channel. Possible values are SM_SYSTEM_PAL, SM_SYSTEM_SECAM and SM_SYSTEM_NTSC.

See also: - videoOutSystem, - setSystem (SMControl)

setVideoOutXofs:

- **setVideoOutXofs:**(short)*nXof*

Moves the foreground channel on video out relative to background channel. This is used to adjust the foreground channel to the background channel, and is normally done in the driver configuration.

See also: - videoOutXofs, - setHorizontalOffset (SMControl)

setVideoOutYofs:

- **setVideoOutYofs:**(short)*nYof*

Moves the foreground channel on video out relative to background channel. This is used to adjust the forground channel to the bagrouind channel, and is normally done in the driver configuration.

See also: - videoOutYofs, - setVerticalOffset (SMControl)

videoOutInput

- (short)**videoOutInput**

Returns the currently active video out input, one of VIDEO_OUT_INPUT_A, VIDEO_OUT_INPUT_B, VIDEO_OUT_INPUT_TV or VIDEO_OUT_INPUT_NONE.

See also: - setVideoOutInput, -videoInput (SMControl)

videoOutSystem

- (short)**videoOutSystem**

Returns currently set video out system, one of SM_SYSTEM_PAL, SM_SYSTEM_SECAM or SM_SYSTEM_NTSC.

See also: - setVideoOutSystem, - system (SMControl)

videoOutXofs

- (short)**videoOutXofs**

Returns the horizontal offset between the foreground channel on video out relative to background channel.

See also: - setVideoOutXofs, - verticalOffset (SMControl)

videoOutYofs

- (short)**videoOutYofs**

Returns the vertical offset between the foreground channel on video out relative to background channel.

See also: - setVideoOutYofs, - horizontalOffset (SMControl)

writeImage:

- writeImage:(NXImage*)*aImage*

Writes *aImage* into the foreground channel if the MMControl class is set to video out. The maximum image size supported is 704x576 for PAL and 640x480 for NTSC. Larger images will be cropped, smaller images will be centered. *aImage* has to contain a MMFLMImageRep representation. If not set to video out see **writeImage:** (SMControl).

See also: writeImage: (SMControl), (MMFLMImageRep)

SMControlAudio

Category of: **SMControl**

Declared in: **smkit/SMControl.h**

Category Description

These methods control the optional audio hardware that can be connected to or be on board a VDOboard. This is either the optional Audio-on-Bracket, a TV-Tuner or the integrated audio of the Movie Machines. Use the **hasAudio:** call to determine if an audio device is connected and functional.

Instance Methods

audioInput

- (unsigned char)**audioInput**

Returns active audio input, SM_AUDIO_INPUT_BLACK, SM_AUDIO_INPUT_RED or SM_AUDIO_INPUT_YELLOW.

See also: - setAudioInput:

balance

- (float)**balance**

Returns current audio balance, a value between 0.0 and 1.0, with 0.5 as the default (equal balance).

See also: - setAudioBalance:

bass

- (float)**bass**

Returns current value of audio bass, a value between 0.0 and 1.0, with 0.5 as the default.

See also: - setAudioBass:

fader

- (float)**fader**

Returns current value of audio fader, a float between 0.0 and 1.0, with 0.5 as the default (equal output front and back).

See also: - setAudioFader:

hasAudio

- (BOOL)**hasAudio**

Returns **YES** if an audio device is connected to the VDOboard, otherwise **NO**.

isAudioMute

- (BOOL)**isAudioMute**

Returns **YES** if audio is muted, otherwise **NO**.

See also: -setAudioMute:

setAudioBalance:

- setAudioBalance:(float)*val*

Sets balance of audio to *val*. Range is from 0.0 to 1.0, with 0.5 being equal volume at left and right, 0.0 results in the right muted, and 1.0 in the left muted.

See also: -balance:

setAudioBass:

- setAudioBass:(float)*val*

Sets bass of audio to *val*. Range is from 0.0 to 1.0, with 0.5 being neutral. 0.0 results in less bass and 1.0 in increased bass in the output audio.

See also: -bass:

setAudioFader:

- setAudioFader:(float)*val*

Sets audio fader (balance between front and back outputs) to *val*. Range is 0.0 to 1.0 with 0.5 being neutral (equal distribution between front and back). 0.0 is all to the front, and 1.0 all to the back output.

See also: -fader:

setAudioInput:

- setAudioInput:(unsigned char)*input*

Set audio input to be active to *input*, SM_AUDIO_INPUT_BLACK, SM_AUDIO_INPUT_RED or SM_AUDIO_INPUT_YELLOW.

See also: -audioInput:

setAudioMute:

- setAudioMute:(BOOL)*state*

If *state* is YES, the audio output is muted, otherwise the audio is played at the current set volume.

See also: -audioMute:

setAudioTreble:

- setAudioTreble:(float)val

Sets treble of audio to *val*. Range is from 0.0 to 1.0, with 0.5 being neutral. 0.0 results in less treble and 1.0 in increased treble in the output audio.

See also: -treble:

setAudioVolume:

- setAudioVolume:(float)*val*

Sets audio volume to *val*. Range is from 0.0 to 1.0.

See also: -volume:

treble

- (float)**treble**

Returns current value of audio treble, a float between 0.0 and 1.0, with 0.5 as the default.

See also: - setAudioTreble:

volume

- (float)**volume**

Returns current value of volume, a float between 0.0 and 1.0, with 0.5 as default.

See also: - setAudioVolume:

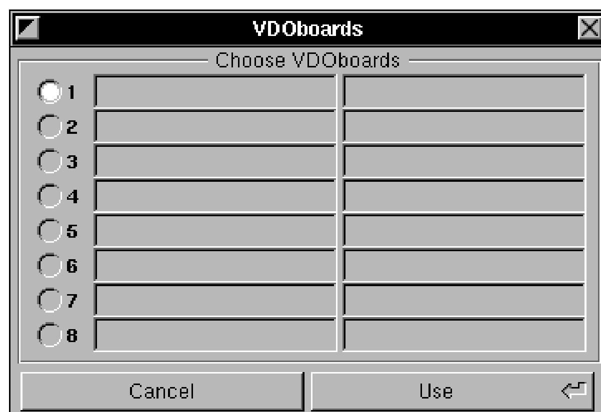
SMControlChooserController

Inherits From: **Object**

Declared In: **smkit/SMControlChooserController.h**

Class Description

The SMControlChooserController is a simple way to let the user choose which VDOboard to use if multiple are present in the system. It loads the file SMMultiChoose.nib. An example nib file is supplied with the class, but it is possible to create a custom one. The custom nib file has to have the name SMMultiChoose.nib, and the following outlets have to be connected: idRadios, idStatusMatrix, idPanel.
Further the cancel: and choose: actions methods should be called.



Instance Variables

id idRadios;
id idStatusMatrix;
id idPanel;
short choice;

idRadios	A Matrix of 4 elements reflecting the choice of the user. The cells should have tags 0 through 3. The class determines the users choice by querying the matrix with the call [[idRadios selectedCell] tag]. Multiple selections should not be allowed.
idStatusMatrix	A Matrix of 4 text cells with tags 0-3. The text cells are set with the current status of the VDOboard in the system.
idPanel	The panel or window the matrices reside in. This pan-

	el is run in a modal loop by the **chooseSM:status:-pids:** method.
choice	Internal variable that holds the users choice

Method Types

- cancel:
- choose:
- chooseSM:status:pids

Instance Methods

cancel:

- cancel:sender

Call when user has aborted choice. This action method is called by the Cancel button in the supplied nib file.

choose:

- choose:sender

Call when user has made choice. This action method is called by the Choose button in the supplied nib file.

chooseSM:status:pids:

- (short)**chooseSM:**(short)*numSM* **status:**(short *)*statuses* **pids:**(int *)*pids*

Method invoked by VDOboards **newWithSelectionAndBoards:withFeatures:** method when more than one VDOboard is present in the system. It is passed *statuses* an array of *numSM* shorts that contain the usage count of the respective VDOboard and an array of *numSM* ints that contains PID of the last application that has registered itself to use the respective VDOboard. Also passed is *numSM*, which is the number of VDOboards present in the system. This method is responsible for loading the nib file and running the panel in a modal loop until the user has made a choice (by calling the **choose:** method) or cancelled the process (by calling the **cancel:** method).

Returns number of the chosen VDOboard or -1 if the process was cancelled.

See Also: newWithSelectionAndBoards:withFeatures: (VDOboards)

SMWindow

Inherits From: **Window : Responder : Object**

Declared In: **smkit/SMWindow.h**

Class Description

The SMWindow is intended to be an "easy to use" object to display a video in its miniwindow, without having to write a single line of additional code to keep the "look and feel" of a NEXTSTEP application.
The SMWindow always behaves like a traditional Window no matter if a SMView is registered or not, despite its miniwindow representation. The default setting for a SMWindow is to display the live video or a grabbed image, if the video is set still, in the miniwindow, when the SMWindow is miniaturized. If more than one SMViews or subclass from it are registered, the video displayed is the active video of the first SMControl object the SMWindow received when registering a video view. For this reason, any settings concerning the image of the miniwindow are ignored if the default setting of the SMWindow is kept. If you want to remove the video display in the miniwindow, use the **showVideoInMiniWindow:** method. Sending a NO in this method switches the miniwindow to a normal representation using the image set for the miniwindow.
Important information:
A SMWindow can not have a backingType NX_BUFFERED. The default setting for the SMWindows backingType is NX_RETAINED. This implies more considerations for the developer to redraw the content when needed, but was necessary because the Interceptor in the current release does not support buffered windows.

Instance Variables

```
List *viewList;
List *smcList;
int nViews;
int nSMCs;
struct _smwFlags {
        unsigned char hasVideo:1;
        unsigned char movingInSMWindow:1;
        unsigned char movingInVideoView:1;
        unsigned char showVideoInMiniWindow:1;
        unsigned char reserved:4;
} smwFlags;
```

viewList	A list object containing the registered SMViews
smcList	A list object containing the SMControl objects belonging to the SMViews
nViews	The number of registered SMViews
nSMCs	The number of different SMControl objects control-

	ling the SMViews
smwFlags.hasVideo	True if at least one SMView has been registered
smwFlags.movingInSMWindow	True if dragging the SMWindow from anywhere is enabled (default: NO)
smwFlags.movingInVideoView	True if dragging the SMWindow by clicking in the SMView is enabled (default: NO)
smwFlags.showVideoInMiniWindow	True if the video should be played in the MiniWindow when the SMWindow is miniaturized (default: YES)

Method Types

Managing the video	- registerVideoView: - unregisterVideoView: - videoViews - hasVideo - showVideoInMiniWindow: - showVideoInMiniWindow
Managing the dragging	- setMovingInSMWindow: - enabledMovingInSMWindow - setMovingInVideoView: - enabledMovingInVideoView

Instance Methods

enabledMovingInSMWindow:

- (BOOL)**enabledMovingInSMWindow**

Returns **YES** if the SMWindow can be moved from anywhere within its content area, not only from the title bar. Otherwise **NO** (default).

See also: - setMovingInSMWindow:

enabledMovingInVideoView:

- (BOOL)**enabledMovingInVideoView**

Returns **YES** if the SMWindow can be moved from within a SMView. Otherwise **NO** (default).

See also: - setMovingInVideoView

hasVideo

- (BOOL)**hasVideo**

Returns **YES** if at least one SMView is present or **NO** if not.
An SMView is present if it has previously been registered by using **registerVideoView:**, and is absent if there is none or it was unregistered by using **unregisterVideoView:**.

See also: - registerVideoView, - unregisterVideoView: - videoVideos

registerVideoView:

- registerVideoView:*aVideoView*

Registers *aVideoView* which has to be a SMView or subclass of SMView. This method does not insert the SMView into the view hierarchy of the SMWindow; it simply notifies the SMWindow about the presence of a (or another) SMView. After registering the SMView, the SMWindow will manage everything concerning the video display: switching the video off while miniaturizing, setting the correct frame for the video display and so on.

See also: - unregisterVideoView, - videoViews - hasVideo

setMovingInSMWindow:

- setMovingInSMWindow:(BOOL)*enabled*

Determines whether the SMWindow can be dragged from anywhere within its content area, depending on *enabled*.

See also: - enabledMovingInSMWindow

setMovingInVideoView:

- setMovingInVideoView:(BOOL)*enabled*

Determines whether or not the SMWindow can be dragged from within its SMView, depending on *enabled*. If dragging from within the SMWindows content area is disabled this setting has no effect.

See also: - enabledMovingInVideoView

showVideoInMiniWindow

- (BOOL)**showVideoInMiniWindow**

Returns **YES** (default) if the video should be displayed in the miniwindow when the SMWindow is miniaturized. Otherwise **NO**.

See also: - showVideoInMiniWindow:

showVideoInMiniWindow:

- showVideoInMiniWindow:(BOOL)*doIt*

Determines whether or not the video should be displayed in the miniwindow when the SMWindow is miniaturized depending on *doIt*.

See also: - showVideoInMiniWindow

unregisterVideoView:

- unregisterVideoView:*aVideoView*

Unregisters *aVideoView* which has been previously set by **registerVideoView:** to inform the SMWindow that it no longer has to manage *aVideoView*. This method does not remove the SMView from the view hierarchy of the SMWindow.

See also: - registerVideoView, - videoViews - hasVideo

videoViews

- (List *)**videoViews**

Returns the list object containing the ids of the SMViews previously registered using **registerVideoView:**.

See also: - registerVideoView, - unregisterVideoView: - hasVideo

SMView

Inherits From: **View : Responder : Object**

Declared In: **smkit/SMView.h**

Class Description

SMView is a view class allowing scaled live video to be displayed in a window and should be used in conjunction with a SMWindow to ensure proper handling. It is possible to use the SM-View in a standard window, but view handling might not be as smooth, and when moved partially offscreen, unexpected results can occur. A SMView has to be connected to a SMControl class in order to be able to display and control the video. It is primarily for handling the size and geometric appearance of the video. It also makes printing possible, as well as clipping (masking of views obscuring live video). A SMView can be part of a scroll view. For easy use there is the SMPaletteView which has action methods to control almost all of the SMControls methods, and it can be used in InterfaceBuilder.

The SMView class controls those aspects of the video that have to do with its geometric appearance on screen, and only those. In particular those are in terms of SMControl class methods, the window and zoom frame, as well as the **setStill:** method. In order to avoid conflicts with the SMControl class, the following methods of the SMControl class should not be used when using a SMView class.

setWindowFrame	(automatically controlled by SMView)
setZoomFrame	(automatically controlled by SMView)
setStill	(use view's **start:** and **stop:** methods instead)
setVideoOn	(use view's **start:** and **stop:** methods instead)

See the SMViewDragging category for drag&drop of images in the SMview class.

Note: Printing
When the **printPSCode:** method is invoked it usually brings up a PrintPanel, and only when the user continues the **drawSelf::** method will be sent to the view. This means the point in time at which the video is grabbed for printing is relatively undetermined. So it is suggested that the view is stopped (with the **stop:**) method before invoking **printPSCode:**. One way to accomplish this is to subclass the **printPSCode:** method and stop the video there.

Note: Clipping
The following methods reset the clipping when invoked. If several of these methods are called in succession (like when initializing the view) it is suggested that the clipping is suspended with the suspendClipping: method. Use suspendClipping: rather the **setClippingOn:** since **setClippingOn:** also resets the clips. Resetting clips involves a flush of the VDOboards memory, which is a time consuming process.
Don't use the clipping methods provided by the SMControl class. They only affect the clipping mask of the VDOboard not the state of the view. Effects like the view blanking, or that the live

video does not follow the window around can happen if the clipping of the VDOboard has been turned off without the view being aware of it.

Note: Windows
All windows that contain a SMView that are of type NX_BUFFERED will behave like a window of type NX_RETAINED. The usual precautions, and drawing optimizations for retained windows apply.

Instance Variables

```
id smControl;
struct _flags {
        unsigned char scrollers:1;
        unsigned char fixedratio:1;
        unsigned char fixedsize:1;
        unsigned char clipping:1;
        unsigned char animated:1;
        unsigned char grabonstop:1;
        unsigned char zoom:1;
        unsigned char stopped:1;
        unsigned char dragfrom:1;
        unsigned char dragto:1;
        unsigned char isReallyObscured:1;
        unsigned char wasOffScreen:1;
        unsigned char willResize:1;
        unsigned char pad:3;
} flags;
NXRect windowFrame;
NXRect zoomFrame;
NXSize screenSize;
NXImage *idGrabbedImage;
char tmpfilename[MAXPATHLEN+1];
int windowNum;
```

smControl	The SMControl class, whose video the view is displaying
flags.scrollers	YES if view should use scrollers on size mismatch
flags.fixedratio	YES if video height to width ratio should always be 0.75
flags.fixedsize	YES if video should not resize with view
flags.clipping	YES if view should mask out windows obscuring the view
flags.animated	YES if dragging and resizing of view should resize the video at the same time
flags.grabonstop	YES if view should grab video and display the grabbed image instead of the still video
flags.zoom	YES if view is not completely visible and video should be adjusted accordingly
flags.stopped	YES if video is still
windowFrame	Size of visible part of view displaying video
zoomFrame	Size of zoom rectangle of full video size
screenSize	Size of current screen

idGrabbedImage	The grabbed image

Method Types

Initializing and freeing SMView objects	- initFrame:control:
	- free
Modifying control	- setControl:
	- control
	- hasControl
	- controlWillFree:
Changing behaviour	- setFixedRatio:
	- isFixedRatio
	- setScaledVideo:
	- isScaledVideo;
	- setVideoSize:
	- getVideoSize:
	- setAnimatedMovement:
	- isAnimatedMovement
	- setGrabOnStop
	- doesGrabOnStop
	- start:sender
	- stop:sender
Acquiring an image from view	- grab
Clipping	- setClippingOn:
	- isClippingOn
	- suspendClipping:
	- resetClips- redrawClips
Dragging (In SMViewDragging Category)	- canDragFrom
	- canDragTo
	- setDragFrom:
	- setDragTo:

Instance Methods

control

- control

Returns id of SMControl class the view is currently connected to.

See also: - setControl:, initFrame:control:

controlWillFree:

- controlWillFree:*sender*

Sent by SMControl to every view in its list when it receives a free message. When the view looses a control it also looses its live video representation. Can be used to give feedback to the user.

See also: - setControl:, initFrame:control:

doesGrabOnStop

- (BOOL)**doesGrabOnStop**

Returns **YES** if an image is grabbed and displayed when video is stopped, otherwise **NO**.

See also: - setControl:, initFrame:control:

getVideoSize:

- getVideoSize:(NXSize *)*aSize*

Returns size of video in *aSize*. This is the size of the video acquisition rectangle, not the size as displayed on screen. Useful if the view is set to fixed size and the displayed video size should be changed.

See also: - setVideoSize:, - setScaledVideo:, - isScaledVideo

grab

- (NXImage *)**grab**

The current frame or field is captured in the size of the view. Returns an id of a newly allocated NXImage, containing a SMYUV- or SMFLMImageRep with the captured data. The receiving object should free the image.

See also: - setGrabOnStop:, - doesGrabOnStop

initFrame:control

- initFrame:(NXRect *)*aRect* **control:***aControl*

Initializes SMView to size specified by *aRect* and sets control to *aControl*. Returns self.

See also: - initFrame: (View)

isClippingOn

- (BOOL)**isClippingOn**

Returns **YES** if windows obscuring the view are masked out. Otherwise returns **NO**. Do not confuse with the views **setClipping:** method.

See also: - setClippingOn:

hasControl

- (BOOL)**hasControl**

Returns **YES** if view has current control (i.e live video), otherwise returns **NO**. If this method returns **NO**, it does not mean that the SMView instance does not have a valid instance of a SM-Control class. It is possible for more than one SMView to be connected to the same SMControl (like the live video in the SMMiniWindow). This method can be used to determine which of these views is currently displaying the live video (since only one can at a given time). This method will also return **NO** if no SMControl is present.

isScaledVideo

- (BOOL)**isScaledVideo**

Returns **YES** if video is scaled to view dimensions. Otherwise returns **NO**. When **NO**, video size is determined by **setVideoSize:**, and the visible portion by visible rectangle of the view.

See also: - setVideoSize:, - setScaledVideo:, - getVideoSize

redrawClips

- redrawClips

Redraws all clips. This means that all areas obscuring the live video view will be masked out, preventing the live video from 'ghosting' ontop of the obscuring windows. This method will not remove areas that have accidentally been masked out although there is no window obscuring the view. Use **resetClips** instead if you want to get rid of wrongly clipped areas. For performance reasons it it usally prefreable to call **suspendClipping:** with YES prior to this call, so the automatic clipping mechanism doesn't conflict with this manual method. Don't forget to reeanable clipping with a **suspendClipping:** with NO afterwards.

```
[view suspendClipping:YES];
[view redrawClips];
[view suspendClipping:NO];
```

See also: - setClippingOn:, - redrawClips:, - suspendClipping:

resetClips

- resetClips

Resets clipps. This is the 'expensive' version, it will clear the entire video view prior to redrawing the clips. For a 'cheaper' version use **redrawClips**. For performance reasons it it usally prefreable to call **suspendClipping:** with YES prior to this call, so the automatic clipping mechanism doesn't conflict with this manual method. Don't forget to reeanable clipping with a **suspendClipping:** NO afterwards.

```
[view suspendClipping:YES];
[view resetClips];
[view suspendClipping:NO];
```

See also: - setClippingOn:, - redrawClips:, - suspendClipping:

setClippingOn:

- setClippingOn:(BOOL)*state*

Turns clipping on when *state* is YES. This method should be used to use clipping, not the method in the control class, since it doesn't change the view behaviour.

See also: - isClippingOn, - resetClips, - redrawClips

setControl:

- setControl:*anObject*

Sets the SMControl class, which the view belongs to, to *anObject*. Reinitializes the VDOboard associated with the control class to the dimensions of the view.

See also: - hasControl, - initFrame:control:

setGrabOnStop

- setGrabOnStop:(BOOL)*state*

Controls whether a grabbed image is displayed, instead of the still video. Set *state* to YES for image, NO for still video. Has to be set to YES to enable dragging images from view.

See also: - doesGrabOnStop:, - grab, - setDragFrom:

setScaledVideo:

- setScaledVideo:(BOOL)*state*

Controls if video is always scaled to view dimensions. If *state* is set to YES, video is scaled to view dimensions. If set to NO, video is cropped when the view is smaller than videosize and the views maximum size is constrained to videosize. The videosize is set with the **setVideoSize:**, and can be determined with the **getVideoSize:** method.

See also: - isScaledVideo, - setVideoSize:, - getVideoSize

setVideoSize:

- (BOOL)**setVideoSize**:(NXSize)*aSize*

Sets size of video when the **setScaledVideo:** is set to NO. If *aSize* is smaller than the current view bounds, the view will not be completely filled with video. If *aSize* is larger than the current view bounds only part of the video can be seen. Use a scrollview to make everything visible.

See also: - isScaledVideo, - setScaledVideo:, - getVideoSize

start:

- start:*sender*

Turns video on and live. Removes grabbed image from view if present. Use instead of SMControls **setStill:** and **setVideoOn:** methods.

See also: - stop

stop:

- stop:*sender*

Sets video to still. If set to **setGrabOnStop:**, displays grabbed image in view instead of still video.Use instead of SMControls **setStill:** and **setVideoOn:** methods.

See also: - start:, - doesGrabOnStop, - setGrabOnStop:

suspendClipping:

- suspendClipping:(BOOL)*state*

If *state* is YES all clipping activity is suspended until this method is called with a *state* of NO. Use this method to temporarily disable the clipping mechanism without the performance hit. Especially useful when calling more than one method that resets clipping. (See class description for details).

See also: - setClippingOn:, - isClippingOn

SMPaletteView

Inherits From: **SMView : View : Responder : Object**

Declared In: **smkit/SMPaletteView.h**

Class Description

The SMPaletteView has been designed to build simple applications using InterfaceBuilder. Most of its methods work by connecting Controls (e.g. Buttons, Matrices …) to it. The really new thing is that you can set defaults for the video input and the frame type that will be taken, when the SMPaletteView gets a **start:** message. If the VDOboard is connected to an additional audio feature, default values for the volume and the mute state are also set when receiving the **start:** message. If one of these values is changed through methods of the SMPaletteView, they will be updated as default values. The reason behind this behaviour is that you can set different default values for different SMPaletteViews in a Window, which will be switched when another SMPaletteView is clicked with the mouse to become active.
For further descriptions see the corresponding methods of the SMControl class.

Instance Variables

unsigned char defInput;
unsigned char defFields;
float defVolume;
BOOL defAudioMute;

defInput	One of four possible video input values the VDO-board offers
defFields	One of the possible three inputFrameTypes
defVolume	A default volume setting for the optional audio feature
defAudioMute	The default state of the optional audio feature whether audio is on or not

Method Types

Managing defaults	- setDefaultAudioMute: - defaultAudioMute: - setDefaultFields: - defaultFields - setDefaultInput: - defaultInput - setDefaultVolume: - defaultVolume

Action methods

- saveImageAs:
- setAudioBalance:
- setAudioBass:
- setAudioFader:
- setAudioMute:
- setAudioTreble:
- setAudioVolume:
- setBlueGain:
- setBrightness:
- setChromaIntensity:
- setChromaInvert:
- setColorOn:
- setContrast:
- setFlipped:
- setGreenGain:
- setHue:
- setLumaIntensity:
- setLumaInvert:
- setLumaSharpness:
- setMixerOn:
- setRedGain:
- setSaturation:
- setSharpness:
- setStill:
- setVideoFields:
- setVideoInput:
- setVideoOnOff:

Instance Methods

defaultAudioMute

- (BOOL)**defaultAudioMute**

Returns **YES** (default) if the optional audio feature is mute when the SMPaletteView gets a start: message. Otherwise this method returns **NO**.

See also: - setDefaultAudioMute:

defaultFields

- (unsigned char)**defaultFields**

Returns SM_FRAME_ODD, SM_FRAME_EVEN or SM_FRAME_BOTH (default) as default for the frameInputType for this SMPaletteView.

See also: - setDefaultFields:

defaultInput

- (unsigned char)**defaultInput**

Returns a psitive integer (starting with 0) as the default video input for this SMPaletteView. The range of values depend on the VDOboard you are using.

See also: - setDefaultInput:, - numberOfInputs (SMControl)**, - inputType:** (SMControl)

defaultVolume

- (float)**defaultVolume**

Returns the default volume for the additional audio option for this SMPaletteView. The default value if not changed is 0.3.

See also: - setDefaultVolume

saveImageAs:

- saveImageAs:*sender*

This method causes the SMPaletteView to grab the actual live video and activates a save panel to store the newly grabbed image.

setAudioBalance:

- setAudioBalance:*sender*

sender has to be a Control object (e.g. Scroller) offering a **floatValue:** method. The range of the float value has to be between 0.0 and 1.0.

setAudioBass:

- setAudioBass:*sender*

sender has to be a Control object (e.g. Scroller) offering a **floatValue:** method. The range of the float value has to be between 0.0 and 1.0.

setAudioFader:

- setAudioFader:*sender*

sender has to be a Control object (e.g. Scroller) offering a **floatValue:** method. The range of the float value has to be between 0.0 and 1.0.

setAudioMute:

- setAudioMute: *sender*

sender has to be a Control object (e.g. Button in toggle mode) offering a **state:** method. The returned value for state has to be YES (selected) or NO (not selected).

setAudioTreble:

- setAudioTreble:*sender*

sender has to be a Control object (e.g. Scroller) offering a **floatValue:** method. The range of the float value has to be between 0.0 and 1.0.

setAudioVolume:

- setAudioVolume:*sender*

sender has to be a Control object (e.g. Scroller) offering a **floatValue:** method. The range of the float value has to be between 0.0 and 1.0.

setBlueGain:

- setBlueGain:*sender*

sender has to be a Control object (e.g. Scroller) offering a **floatValue:** method. The range of the float value has to be between 0.0 and 1.0.

setBrightness:

- setBrightness:*sender*

sender has to be a Control object (e.g. Scroller) offering a **floatValue:** method. The range of the float value has to be between 0.0 and 1.0.

setChromaIntensity:

- setChromaIntensity:*sender*

sender has to be a Control object (e.g. Scroller) offering a **floatValue:** method. The range of the float value has to be between 0.0 and 1.0.

setChromaInvert:

- setChromaInvert:*sender*

sender has to be a Control object (e.g. Button in toggle mode) offering a **state:** method. The returned value for state has to be YES (selected) or NO (not selected).

setColorOn:

- setColorOn:*sender*

sender has to be a Control object (e.g. Button in toggle mode) offering a **state:** method. The returned value for state has to be YES (selected) or NO (not selected).

setContrast:

- setContrast:*sender*

sender has to be a Control object (e.g. Scroller) offering a **floatValue:** method. The range of the float value has to be between 0.0 and 1.0.

setDefaultAudioMute:

- setDefaultAudioMute:(BOOL)*audioMute*

This method has effect only if an optional audio feature is connected to the VDOboard. Through this method, the mute state of the audio feature can be set. Sending NO (default) keeps the audio feature turned off when the SMPaletteView receives a **start:** message, YES will turn audio on.

See also: - defaultAudioMute

setDefaultFields:

- setDefaultFields:(int)*fields*

This method sets the **inputFrameType** for this SMPaletteView that will be set when a **start:** message is sent.
Possible values are : SM_FRAME_ODD, SM_FRAME_EVEN and SM_FRAME_BOTH.

See also: - defaultFields

setDefaultInput:

- setDefaultInput:(int)*input*

This method sets the video input for this SMPaletteView that will be set when a **start:** message is sent.
Possible values are positive integers starting with 0(default). The range of values depend on the VDOboard you are using.

See also: - defaultInput, - numberOfInputs (SMControl)**, - inputType:** (SMControl)

setDefaultVolume:

- setDefaultVolume:(float)*volume*

This method has effect only if an optional audio feature is connected to the VDOboard. Setting a default volume between 0.0 and 1.0 will set the volume of the audio feature to volume when a **start:** message is sent.

See also: - (float)defaultVolume

setFlipped:

- setFlipped:*sender*

sender has to be a Control object (e.g. Button in toggle mode) offering a **state:** method. The returned value for state has to be YES (selected) or NO (not selected).

setGreenGain:

- setGreenGain:*sender*

sender has to be a Control object (e.g. Scroller) offering a **floatValue:** method. The range of the float value has to be between 0.0 and 1.0.

setHue:

- setHue:*sender*

sender has to be a Control object (e.g. Scroller) offering a **floatValue:** method. The range of the float value has to be between 0.0 and 1.0.

setLumaIntensity:

- setLumaIntensity:*sender*

sender has to be a Control object (e.g. Scroller) offering a **floatValue:** method. The range of the float value has to be between 0.0 and 1.0.

setLumaInvert:

- setLumaInvert: *sender*

sender has to be a Control object (e.g. Button in toggle mode) offering a **state:** method. The returned value for state has to be YES (selected) or NO (not selected).

setLumaSharpness:

- setLumaSharpness: *sender*

sender has to be a Control object (e.g. Scroller) offering a **floatValue:** method. The range of the float value has to be between 0.0 and 1.0.

setMixerOn:

- setMixerOn: *sender*

sender has to be a Control object (e.g. Button in toggle mode) offering a **state:** method. The returned value for state has to be YES (selected) or NO (not selected).

setRedGain:

- setRedGain: *sender*

sender has to be a Control object (e.g. Scroller) offering a **floatValue:** method. The range of the float value has to be between 0.0 and 1.0.

setSaturation:

- setSaturation: *sender*

sender has to be a Control object (e.g. Scroller) offering a **floatValue:** method. The range of the float value has to be between 0.0 and 1.0.

setSharpness:

- setSharpness: *sender*

sender has to be a Control object (e.g. Scroller) offering a **floatValue:** method. The range of the float value has to be between 0.0 and 1.0.

setStill:

- setStill: *sender*

sender has to be a Control object (e.g. Button in toggle mode) offering a **state:** method. The returned value for state has to be YES (selected) or NO (not selected).

setVideoFields:

- setVideoFields: *sender*

sender has to be a matrix of Controls (e.g. Matrix of radiobuttons) where the tags have to be set to represent possible **inputFrameTypes** of the VDOboard. (tag value 1 = SM_FRAME_ODD, 2 = SM_FRAME_EVEN, 3 = SM_FRAME_BOTH). *sender* has to be a matrix because the mechanism used to set the value is :
[[sender selectedCell] tag].

setVideoInput:

- setVideoInput: *sender*

sender has to be a matrix of Controls (e.g. Matrix of radiobuttons) where the tags have to be set to represent possible video inputs of the VDOboard. The range of values depend on the VDO-board you are using. (e.g. Screen Machine : tag value 0 = SM_INPUT_BLACK, 1 = SM_INPUT_RED, 2 = SM_INPUT_YELLOW, 3 =SM_INPUT_SVHS). *sender* has to be a matrix because the mechanism used to set the value is :
[[sender selectedCell] tag].

setVideoOnOff:

- setVideoOnOff: *sender*

sender has to be a Control object (e.g. Button in toggle mode) offering a **state:** method. The returned value for state has to be YES (selected) or NO (not selected).

SMViewDragging

Category of: **SMView**

Declared in: **smkit/SMView.h**

Category Description

SMViewDragging is an addition to the SMView implementing the dragging protocol. It allows dragging from images out of the view into other applications. Any stopped, grabbed image can be dragged (enabled with **setGrabOnStop:**). Dragging of live video or stopped video only is not supported.
Images can also be dragged from the workspace into the view, provided they are type FLM, TIFF or EPS. This provides a convenient way to preview images in 24bit regardless of the windoserver color depth.

Instance Methods

canDragFrom

- (BOOL)**canDragFrom**

Returns **YES** if dragging a stopped grabbed image is enabled. Only works if **setGrabOnStop:** is enabled as well. When the view is stopped, the user can drag an icon out of the view representing the stopped video into any application accepting NXFilenamePboardType or NXTIFFPboardType

See also: - setDragFrom:

canDragTo

- (BOOL)**canDragTo**

Returns **YES** if dragging an image into the view is enabled. If the dragged pasteboard contains a filename ending in .FLM, .TIFF or .EPS the image will be scaled and displayed in the view.

See also: - setDragTo:

setDragFrom

- setDragFrom:(BOOL)*state*

If set to YES, dragging a stopped image form view is enabled otherwise this feature is disabled.

See also: - canDragFrom

setDragTo:

- setDragTo:(BOOL)*state*

If set to YES, dragging an image from Workspace into view is enabled otherwise this feature is disabled

SMYUVImageRep

Inherits From: **NXImageRep:Object**

Declared In: **smkit/SMYUVImageRep.h**

Class Description

The SMYUVImageRep simplifies the use of the YUV data obtained from a VDOboard. The YUV format (Y is luminance, U and V are chrominance) is commonly used in video processing and broadcasting. It makes it possible to support both black and white and color video at the same time as the color and monochrome information are stored separately. Computer images, and those in the other image representations, use the RGB color space (like TIFF uses). The SMYUVImageRep bridges the gap between the YUV and RGB color space by converting either way. Be aware that in SMYUVImageRep the settings of all gains, brightness, contrast, saturation and hue values are not taken into consideration when converting to and from YUV.
The class employs a lazy conversion strategy - data is only converted into the other format when requested (i.e when the instance is initialized with YUV data with the **initData:pixelsWide:-pixelsHigh:YUVMode:** method, the data is not converted into RGB until requested by a method like **writeTIFF:**), or to SM_YUVMODE6 if necessary during initialization. Internaly the data is hold in mode SM_YUVMODE6, therefore this mode is the best mode to read and write data. This allows faster data handling during a process like sequence grabbing.

Instance Variables

int imgMode;

imgMode	YUV mode of the image data.

Instance Constants

Constant Name	Pixel Ordering
SM_YUVMODE0	YYVUYY
SM_YUVMODE1	Y
SM_YUVMODE2	YYVU
SM_YUVMODE3	YVU
SM_YUVMODE4	YYVUYY
SM_YUVMODE6	YUYV
SM_RGBMODE	RGB (24 bit)

Method Types

Initializing a new SMYUVImageRep object	- init - initData: pixelsWide: pixelsHigh:YUVMode: - initDataFromStream: pixelsWide:pixelsHigh: YUVMode: - initFromPasteboard: - initTIFFDataFromStream:
Freeing an SMYUVImageRep	- free
Checking data types	+ canLoadFromStream:
Setting the size of the image	- setSize: - imageSizeFromMode:
Representation attributes	- bitsPerSample - hasAlpha - isOpaque - mode - numColors - setAlpha: - setBitsPerSample: - setNumColors: - setOpaque: - setPixelsHigh: - setPixelsWide: - updateDisplayData
Getting image data	- getData: withMode: - imageDataInMode6
Writing a TIFF representation of the image	- writeTIFF: - writeTIFF: usingCompression: - writeTIFF: usingCompression: andFactor:
Setting/checking compression types	+ getTIFFCompressionTypes:count: - getCompression:andFactor: - setCompression:andFactor:
Archiving	- read: - write:

Class Methods

canLoadFromStream:

+ (BOOL)**canLoadFromStream:**(NXStream *)*stream*

Tests whether the receiving class can initialize an instance of itself from *stream*. Currently this method always returns **YES**.

See also: - initDataFromStream:, - initTIFFDataFromStream:

getTIFFCompressionTypes:count:

+ (void)**getTIFFCompressionTypes:**(const int **)*list* **count:**(int *)*numTypes*

Returns, by reference, an integer array representing all available compression types that can be used when writing a TIFF image. The number of elements in *list* is represented by *numTypes*. *list* and *numTypes*, belonging to the SMYUVImageRep class; it shouldn't be freed or altered.

The following compression types are supported:

NX_TIFF_COMPRESSION_NONE	1
NX_TIFF_COMPRESSION_LZW	5
NX_TIFF_COMPRESSION_JPEG	6

Instance Methods

bitsPerSample

- (int)**bitsPerSample**

Currently always returns 8.

See also: - setBitsPerSample:

free

- free

Deallocates the SMYUVImageRep. This method will not free any image data that the object merely references, that is, raw data that was passed to it in a **initData:...** , **initDataFrom-Stream:...** or **initTIFFDataFromStream:...** message. Returns **nil**.

getCompression:andFactor:

- (void)**getCompression:**(int *)*compression* **andFactor:**(float *)*factor*

Returns, by reference, the receiver's compression type and compression factor. Use this method to get information on the compression type for the source image data. *compression* represents the compression type used on the data and corresponds to one of the values returned by the class method **getTIFFCompressionTypes:count:**. **factor** is usually a value between 0.0 and 255.0, with 0.0 representing no compression.

See also: + getTIFFCompressionTypes:..., - setCompression:andFactor:

getData: withMode:

- getData:(unsigned char*)*data* **withMode:**(int)*aMode*

Copies the image data in the mode *aMode* to the array *data*. *aMode* can be one of the modes declared in **mode**. The size of data depends on the given mode *aMode* and should be the size given by **imageSizeFromMode:**. Returns **self**.

See also: - mode, - imageSizeFromMode

hasAlpha

- (BOOL)**hasAlpha**

Currently always returns **NO**.

See also: - setAlpha:

imageDataInMode6

- (unsigned char*)**imageDataInMode6**

This method gives an direct access to the image data stored in the SMYUVImageRep. When modifying the data after displaying the image on screen you should send a **updateDisplayData** method to force the SMYUVImageRep to rerender the display data.

See also: - getData:, -updateDisplayData

imageSizeFromMode:

- (int)**imageSizeFromMode:**(int)*aMode*

Returns the number of bytes that would be required to get data for the current image in mode *aMode*. *aMode* can be one of the modes declared in **mode**.

See also: - mode, - getData:

init

- init

Initializes the receiver, a newly allocated SMYUVImageRep object. If the object can't be initialized, this method frees it and returns **nil**. Otherwise, it returns the object **self**.

See also: - initTIFFDataFromStream:..., - initDataFromStream:..., -initData:...

initData: pixelsWide: pixelsHigh:YUVMode:

- initData:(unsigned char*)*data* **pixelsWide:**(int)*width* **pixelsHigh:**(int)*height* **YUVMode:**(int)*aMode*

Initializes the receiver, a newly allocated SMYUVImageRep object, so that it can render the image as specified in *data* and the other arguments. If the object can't be initialized, this method frees it and returns **nil**. Otherwise it returns the object **self**.
data points to a buffer containing raw bitmap data. The data component values should be interwoven in a single channel ("meshed configuration"). If *data* is NULL, the SMYUVImageRep will allocate enough memory to hold YUV data (SM_YUVMODE6) for the image.
All the other arguments to this method are the same as those to **initDataFromStream:...** See that method for further descriptions.

See also: - initTIFFDataFromStream:..., - initDataFromStream:..., -initData:...

initDataFromStream: pixelsWide: pixelsHigh: YUVMode:

- initDataFromStream:(NXStream*)*theStream* **pixelsWide:**(int)*width* **pixelsHigh:**(int)*height* **YUVMode:**(int)*aMode*

Initializes the receiver, a newly allocated SMYUVImageRep object, so that it can render the image as specified in *theStream* and the other arguments. If the object can't be initialized, this method frees it and returns **nil**. Otherwise it returns the object **self**.
The data component values contained in *theStream* should be interwoven in a single channel ("meshed configuration"). *theStream* must be seekable.
Each of the other arguments informs the SMYUVImageRep object about the image. They're explained below:

- *width* and *height* specify the size of the image in pixels.
  The size in each direction must be greater than 0.

- *aMode* indicates how data values are to be interpreted.
  It should be one of the following enumerated values:

SM_YUVMODE0	YYVUYY mode
SM_YUVMODE1	Y mode
SM_YUVMODE2	YYVU mode
SM_YUVMODE3	YVU mode
SM_YUVMODE4	YYVUYY (internal mode of Movie Machine)
SM_YUVMODE6	YUYV mode (default mode)
SM_RGBMODE	24 bit RGB mode without a alpha plane

See also: - canLoadFromStream:, - initTIFFDataFromStream:

initFromPasteboard:

- initFromPasteboard:(Pasteboard *)*pasteboard*

Does nothing and returns **nil**.

See also: - initTIFFDataFromStream:..., - initDataFromStream:..., -initData:...

initTIFFDataFromStream:

- initTIFFDataFromStream:(NXStream*)*theStream*

Initializes the receiver, a newly allocated SMYUVImageRep object, with the TIFF image read from *theStream*. If the new object can't be initialized for any reason (for example, *theStream* doesn't contain TIFF data), this method frees it and returns **nil**. Otherwise it returns **self**.

See also: - initDataFromStream:..., - canLoadFromStream:

isOpaque

- (BOOL)**isOpaque**

Currently always returns **NO**.

See also: - hasAlpha, - setOpaque:

mode

- (int)**mode**

Returns the mode of the currently initialized image. The mode can be one of the following enumerated values:

SM_YUVMODE0	YYVUYY mode
SM_YUVMODE1	Y mode
SM_YUVMODE2	YYVU mode
SM_YUVMODE3	YVU mode
SM_YUVMODE4	YYVUYY
SM_YUVMODE6	YUYV mode
SM_RGBMODE	24 bit RGB mode

numColors

- (int)**numColors**

Returns the number of color components in the image. For example, the return value will be 1 for images specified by mode SM_YUVMODE1 or any other one components. It will be 3 for images specified by SM_RGBMODE, or SM_YUVMODE3 mode.

See also: - setNumColors:, - mode

read:

- read:(NXTypedStream *)*stream*

Reads the SMYUVImageRep from the typed stream *stream*. Returns **self**.

See also: - write:

setAlpha:

- setAlpha:(BOOL)*flag*

Does nothing and returns **self**.

See also: - hasAlpha:

setBitsPerSample:

- setBitsPerSample:(int)*anInt*

Does nothing and returns **self**.

See also: - bitsPerSample:

setCompression:andFactor:

- setCompression:(int)*compression* **andFactor:**(float)*factor*

Sets the receiver's compression type and compression factor. *compression* is one of the supported compression types. *factor* is a compression factor, usually between 0.0 (no compression) and 255.0 (maximum compression).

When an SMYUVImageRep is created, the instance stores the compression type and factor for the source data. When the data is subsequently saved, **writeTIFF:** tries to use the stored compression type and factor. Use this method to change the compression type and factor. Returns **self**.

See also: + getTiffCompressionTypes:count:, - getCompression:andFactor:

setNumColors:

- setNumColors:(int)anInt

Does nothing and returns **self**.

See also: - numColors:, - mode

setOpaque:

- setOpaque:(BOOL)*flag*

Does nothing and returns **self**.

See also: - isOpaque:

setPixelsHigh:

- setPixelsHigh:(int)*anInt*

Does nothing and returns **self**.

See also: - pixelsHigh (NXImageRep)**,**

setPixelsWide:

- setPixelsWide:(int)*anInt*

Does nothing and returns **self**.

See also: - pixelsWide (NXImageRep)

setSize:

- setSize:(const NXSize *)*aSize*

Does nothing and returns **self**.

See also: - getSize (NXImageRep)

updateDisplayData:

- updateDisplayData

Forces the SMYUVImageRep to rerender its display image data. This is only necessary if you change the YUV data using the **imageDataInMode6** method. Returns **self**.

See also: - imageDataInMode6

write:

- write:(NXTypedStream *)*stream*

Writes the SMYUVImageRep to the typed stream *stream*. Returns **self**.

See also: - read:

writeTIFF:

- writeTIFF:(NXStream *)*stream*

Writes a TIFF representation of the image to *stream*. This method invokes **writeTIFF:usingCompression:andFactor:** using the stored compression type and factor retrieved from the initial image data, or is changed using **setCompression:andFactor:**. If the stored compression type isn't supported for writing TIFF data
(e.g., NX_TIFF_COMPRESSION_NEXT), the stored compression is changed to NX_TIFF_COMPRESSION_NONE and the compression factor to 0.0 before invoking **writeTIFF:usingCompression:andFactor:**.

See also: - getCompression:andFactor:, - setCompression:andFactor:

writeTIFF: usingCompression:

- writeTIFF:(NXStream *)*stream* **usingCompression:**(int)*compression*

Writes a TIFF representation of the image to *stream*, compressing the data according to the compression scheme. If *compression* is NX_TIFF_COMPRESSION_JPEG, the default compression factor is used. This and the other compression constants are listed under the following method.

See also: - writeTIFF:usingCompression:andFactor:

writeTIFF: usingCompression: andFactor:

- writeTIFF:(NXStream *)*stream* **usingCompression:**(int)*compression*
andFactor:(float)*factor*

Writes a TIFF representation of the image to *stream*. If the stream isn't currently positioned at location 0, this method assumes that it contains another TIFF image. It will try to append the TIFF representation it writes to that image. To do this, it must read the header of the image already in the stream. Therefore, the stream must be opened with NX_READWRITE permission.

The second argument, *compression*, indicates the compression scheme to use. It should be one of the following constants:

NX_TIFF_COMPRESSION_NONE	No compression
NX_TIFF_COMPRESSION_LZW	LZW compression
NX_TIFF_COMPRESSION_JPEG	JPEG compression

The third argument, *factor*, is used in the JPEG scheme to determine the degree of compression. If factor is 0.0, the default compression factor of 10.0 will be used. Otherwise, factor should fall within the range 1.0 - 255.0, with higher values yielding greater compression but also greater information losses.

See also: - writeTIFF:usingCompression:

SMFLMImageRep

Inherits From: **SMYUVImageRep:NXImageRep:Object**

Declared In: **smkit/SMFLMImageRep.h**

Class Description

The SMFLMImageRep is a class to provide compatibility with the FLM data format, which is used by the DOS and Windows version of the VDOboard's software. It writes and reads the YUV data, including a special header. FLM data files have a .flm extension. The FLM format is the ///FASTest way of storing the YUV data from the VDOboard and no conversion between color spaces has to be made.

Different from the SMYUVImageRep the SMFLMImageRep can store the values for brightness, contrast, saturation, red, green and blue gain. These values take effect when converting data from the YUV color space to RGB or reloading the image back into the VDOboard. Note that (as described in the SMControl class) higher values for SMImageHead.nRed, SMImageHead.nGreen, SMImageHead.nBlue greater than 32 (SMControl : redGain, blueGain, greenGain greater than 1.0) may result in strange image effects when converting to RGB. This is due to the higher output dynamics of the VDOboard.

When initializing the rep with a **initData:**, **initDataStream:** or a **initFromPasteboard:** and the pasteboard contains FLM data (SMFLMPboardType) the stReadHeader struct is set and you can get information about the image with **getHeader:**. You should not change the stReadHeader struct when subclassing the SMFLMImageRep, but you can change the header for write operations with **setWriteHeader:**. This Header is also used when rendering the data on screen.

In the initialization of a SMFLMImageRep the values of the header and the optional description text of a FLM-image are copied to the corresponding “write”-variables. You need not to do so if you don't want to change anything.

The text handled with **text:**, **writeText:** and **setWriteText:** is optional and gives you the possibility to store with the raw image data a few more information about the image, the date of creation, the creator and so on.

To get a fast preview of FLM images, you can add an icon of the image to the image data. To do so you have to **setIconEnabled:** to YES before using the **writeFLM:withMode:** method. The icon is optional and not necessary to display the image through a VDOboard. However, with **getIcon:** you can get an idea of the contents of a FLM file before writing it into a VDOboard or rendering it on screen. The icon is written in the YUV mode SM_YUVMODE0 to use as less space as possible.

To ensure fast working, converting, and rendering of the FLM data you should only use the SM_YUVMODE6 mode, because, like the SMYUVImageRep, the SMFLMImageRep class employs a kind of lazy conversion strategy. That means that the data is internally held in the SM_YUVMODE6 format and is only converted to other formats when requested. Converting to other formats can reduce data length, but in most cases with a lack of quality and in every case a lack of speed.

Instance Variables

```
struct_SMImageHead {
        char sFLMId[5];
        char cTextEnd;
        long lIconOffs;
        short nImageWdt;
        short nImageHgt;
        char nYUVMode;
        short nYUVLeng;
        char cYBits;
        char cUBits;
        char cVBits;
        char cCompMod;
        char cOldComp;
        short nScrMod;
        short nIconLenght;
        long lTextOffs;
        short nTextLen;
        short nContr;
        short nBright;
        short nSatur;
        short nHue;
        short nRed;
        short nGreen;
        short nBlue;
        char sRes[18];
} SMImageHead;
NXAtom SMFLMPboardType;
SMImageHead stWriteHeader;
SMImageHead stReadHeader;
char *sReadImageText;
char *sWriteImageText;
BOOL bIconEnabled;
NXStream *iconStream;
NXSize iconSize;
```

SMImageHead.sFLMId[5]	Id of FLM files
SMImageHead.cTextEnd	Reserved
SMImageHead.lIconOffs	Offset of icon data
SMImageHead.nImageWdt	Width of the image in pixels
SMImageHead.nImageHgt	Height of the image in pixels
SMImageHead.nYUVMode	YUV mode of the image data
SMImageHead.nYUVLeng	Width of the image in bytes
SMImageHead.cYBits	No. of bits of each Y component. (Currently always 8)
SMImageHead.cUBits	No. of bits of each U component. (Currently always 8)
SMImageHead.cVBits	No. of bits of each V component. (Currently always 8)

SMImageHead.cCompMod	Compression mode of the image data. (Currently always 0)
SMImageHead.cOldComp	Reserved
SMImageHead.nScrMod	Reserved
SMImageHead.nIconLenght	Length of icon data (if 0 no icon available)
SMImageHead.lTextOffs	Offset of text data
SMImageHead.nTextLen	Length of text data (0 if no text available).
SMImageHead.nContr	Contrast value of image when grabbed.
SMImageHead.nBright	Brightness value of image when grabbed.
SMImageHead.nSatur	Saturation value of image when grabbed.
SMImageHead.nHue	Hue value of image when grabbed.
SMImageHead.nRed	Red gain of image when grabbed.
SMImageHead.nGreen	Green gain of image when grabbed.
SMImageHead.nBlue	Blue gain of image when grabbed.
SMImageHead.sRes[18]	Reserved
SMFLMPboardType	FLM Pasteboard
stWriteHeader	FLM-Header for write operations.
stReadHeader	Read FLM-Header when initializing.
sReadImageText	Read image text when initializing.
sWriteImageText	Image text for write operations.
bIconEnabled	Write FLM with icon.
iconStream	Icon stream
iconSize	Size of the icon.

Method Types

Initializing a new SMFLMImageRep object	- init - initData: - initFromStream: - initFromPasteboard:
Freeing an SMFLMImageRep	- free
Checking data types	+ canLoadFromStream: + imageUnfilteredFileTypes + imageUnfilteredPasteboardTypes
Getting information about the image	- getHeader: - getIcon: - text - hasIcon
Modifying the write attributes	- getWriteHeader: - writeText - iconEnabled - setIconEnabled: - setWriteHeader: - setWriteText:
Producing a FLM representation of an image	- writeFLM: withMode:

Archiving	- read: - write:

Class Methods

canLoadFromStream:

+ (BOOL)**canLoadFromStream:**(NXStream *)*stream*

Tests whether the SMFLMImageRep class can initialize an instance of itself from *stream*.

See also: - initFromStream:

imageUnfilteredFileTypes

+ (const char *const *)**imageUnfilteredFileTypes**

Returns a null-terminated array of strings representing all file types (extensions) supported by the SMFLMImageRep. Supported types are: “tiff” and “flm”. Invoked by NXImage's **imageRepForFileType:** method to find the NXImageRep subclass capable of handling files with a particular extension.

imageUnfilteredPasteboardTypes

+ (const NXAtom *)**imageUnfilteredPasteboardTypes**

Returns a null-terminated array representing all pasteboard types supported by the SMFLMImageRep. Supported types are :

NXTIFFPboardType
SMFLMPboardType
NXFilenamePboardType

Invoked by NXImage's **imageRepForPasteboardType:** method to find the NXImageRep subclass capable of handling pasteboards containing FLM.

Instance Methods

free

- free

Deallocates the SMFLMImageRep. This method will not free any image data that the object merely references, such as raw data that was passed to it in a **initData:...** , **initFromStream:...** , **initTIFFDataFromStream:...**, **initDataFromStream:...** or **initTIFFDataFromStream:...** message. Returns **nil**.

getHeader:

- getHeader:(SMImageHead *)*aHeader*

Copies the header of the image to the structure referred to by *aHeader*, and returns **self**. The returned header is the header of the image initialized in a **initData:...**, **initFromStream:...** or **initFromPasteboard:...** message. For all the other init-methods the returned header contains default values.

See also: - getWriteHeader:, - setWriteHeader:

getIcon:

- getIcon:(NXImage *)*anImage*

Copies the icon in the FLM raw data into *anImage* and returns **self**. *anImage* should be allocated and initialized. If no icon is available (e.g. the FLM data contains no icon or the SMFLMImageRep is not initialized in a **initFromPasteboard:...**, **initData:...** or **initFromStream:...** message) this method returns **nil**.

See also: - hasIcon

getWriteHeader:

- getWriteHeader:(SMImageHead)*aHeader*

Copies the header of the image to the structure referred to by *aHeader*, and returns **self**. The returned header is the header that will be written in the **writeFLM:...** method. If the SMFLMImageRep is initialized in a **initFromPasteboard:...**, **initFromStream:...** or **initData:...** message, the write header is set equal to the read header.

See also: - setWriteHeader:, - getHeader:

hasIcon

- (BOOL)**hasIcon**

Returns **YES** if there is an icon in the FLM raw data available. You can extract the icon with the **getIcon:** method.

See also: - getIcon:

iconEnabled

- (BOOL)**iconEnabled**

Returns whether the SMFLMImageRep adds an icon at the end of the FLM data using the **writeFLM:...** method. If the SMFLMImageRep is initialized in an **initFromPasteboard:...**, **initFromStream:...** or **initData:...** message and contains an icon, iconEnabled is set to **YES**.

See also - setIconEnabled:

init

- init

Initializes the receiver, a newly allocated SMFLMImageRep object. If the object can't be initialized, this method frees it and returns **nil**. Otherwise, it returns the object **self**.

See also: - initData:, - initFromStream:, - initFromPasteboard:

initData:

- initData:(unsigned char *)*data*

Initializes the receiver, a newly allocated SMFLMImageRep object, so that it can render the image specified in *data*. If the object can't be initialized, this method frees it and returns **nil**. Otherwise, it returns the object **self**.

See also: - init:, - initFromStream:, - initFromPasteboard:

initFromStream:

- initFromStream:(NXStream *)*theStream*

Initializes the receiver, a newly allocated SMFLMImageRep object, so that it can render the image specified in *theStream*. If the object can't be initialized, this method frees it and returns **nil**. Otherwise, it returns the object **self**.

See also: - init:, - initData:, - initFromPasteboard:

initFromPasteboard:

- initFromPasteboard:(Pasteboard *)*thePasteboard*

Initializes the receiver, a newly allocated SMFLMImageRep object, so that it can render the image with data from the given pasteboard *thePasteboard*. If the object can't be initialized, this method frees it and returns **nil**. Otherwise, it returns the object **self**.

See also: - init:, - initData:, - initFromStream:

read:

- read:(NXTypedStream *)*stream*

Reads the SMFLMImageRep from the typed stream *stream*. Returns **self**.

See also: - write:

setIconEnabled:

- setIconEnabled:(BOOL)*flag*

Informs the SMFLMImageRep if an icon should be added to the FLM data when using the **writeFLM:...** method. If *flag* is YES an icon is added.

See also: - iconEnabled:

setWriteHeader:

- setWriteHeader:(SMImageHead)*aHeader*

Informs the SMFLMImageRep that the header *aHeader* should be used when writing the FLM data using the **writeFLM:** method. Negative values in the struct **SMImageHead** are ignored.

Warning : The **setWriteHeader:** is overridden by all init methods.

The only values of *aHeader* used yet are : nContr, nBright, nSatur, nHue, nRed, nGreen and nBlue.

See also: - getWriteHeader:, - getHeader:

setWriteText:

- setWriteText:(char *)*aText*

Informs the SMFLMImageRep that the string aText should be added to the FLM data when using the **writeFLM:...** method.

Warning : The **setWriteText:** is overridden by all init methods.

See also: - writeText, - text

text

- (char *)**text**

Returns the text in the FLM raw data. If no text is available (e.g. the FLM data contains no text or the SMFLMImageRep is not initialized in a **initFromPasteboard:...**, **initFromStream:...** or **initData:...** message) this method returns **NULL**.

See also: - writeText, - setWriteText:

write:

- write:(NXTypedStream *)*stream*

Writes the SMFLMImageRep to the typed stream *stream*. Returns **self**.

See also: - read:

writeFLM: withMode:

- writeFLM:(NXStream *)*theStream* **withMode:**(int)*aMode*

Writes a FLM representation of the image to stream *theStream*. *aMode* is the YUV mode of the FLM representation and should be one of the following enumerated values (also described in SMYUVImageRep) :

SM_YUVMODE0	YYVUYY mode
SM_YUVMODE1	Y mode
SM_YUVMODE2	YYVU mode
SM_YUVMODE3	YVU mode
SM_YUVMODE4	YYVUYY mode (internal mode of Movie Machine)
SM_YUVMODE6	YUYV mode (internal mode of Screen Machine)

The header which is used can be modified with the **setWriteHeader:...** method.

See also: - setWriteHeader:, - setWriteText:, - setIconEnabled:

writeText

- (char *)**writeText**

Returns the text that will be written in the **writeFLM:...** method. If no text is available this method returns **NULL**. If the SMFLMImageRep is initialized in an **initFromPasteboard:...**, **initDataStream:...** or **initData:...** message, the write text is set equal to the read text.

See also: - setWriteText:, - text

MMFLMImageRep

Inherits From: **SMFLMImageRep**

Declared In: **smkit/MMFLMImageRep.h**

Class Description

The MMFLMImageRep is a class to provide the keying feature of Movie Machine Pro/II. The Movie Machine Pro/II has the ability to display images on video out with a transparent color in it. Use the -**setKeyColorFromPoint:** method to pick a key color out of an image and activate keying by using -**setKeyingOn:** .

```
- displayImageOnVideoOut:(NXStream*)flmImageStream
{
    SMChromaSpace aSpace={0.0,0.0,0.0,0.0,0.0,0.0};
    MMFLMImageRep *myRep;
    NXImage *myImage;
    NXSize imageSize;
    NXPoint keyPoint;

    // Initialize a new MMFLMImageRep form flmImageStream
    myRep=[[MMFLMImageRep alloc] initFromStream:flmImageStream];
    if(!myRep)
    {
            return self;
    }

    // Get a valid point for keying
    [myRep getSize:&imageSize];
    keyPoint.x=imageSize.width/2.0;
    keyPoint.y=imageSize.height/2.0;

    // Pick the color of that point
    [myRep setKeyColorFromPoint:&keyPoint]

    // Enable keying
    [myRep setKeyingOn:YES];
    // Make sure that Movie Machine is set to still
    if(![mmControl isStill])
    {
            [mmControl setStill:YES];
    }

    // Set keying on Movie Machine on
    [mmControl setChromaKey:aSpace mode:1];
```

```
    // Put the rep in an image wrapper
    myImage=[[NXImage alloc] initSize:&imageSize];
    [myImage useRepresentation:myRep];

    // display the image
    [mmControl writeImage:myImage];
    // Frees also rep inside
    [myImage free];
    return self;
}
```

You needn't to use this class just for handling images for Movie Machine Pro/II. Use this class only if you want to get control of the keying feature. In all other cases the SMFLMImageRep class is all you need.
Note : Be aware that not every YUV color has a correlating RGB color and the other way round. To be sure to key the right color use the **-setYUVKeyColor:** or the **-setKeyColorFromPoint:** method.

Instance Variables

```
SMColor smKeyColor;
NXColor nxKeyColor;
BOOL useKeyColor;
BOOL mm_data_should_update;
typedef struct _SMColor {
        unsigned char Y;
        char U;
        char V;
} SMColor;
```

smKeyColor	Currently set YUV-color for keying in Movie Machine.
nxKeyColor	Currently set RGB-color for keying on screen.
useKeyColor	MMFLMImageRep should use key color.

Method Types

- rbgKeyColor
- setKeyColorFromPoint:
- setKeyingOn:
- setRGBKeyColor:
- setYUVKeyColor:
- usesKeying
- yuvKeyColor

Function Types

SMConvertRGBToYUV()
SMConvertYUVToRGB()

Instance Methods

rbgKeyColor

- (NXColor)**rbgKeyColor**

Returns the currently set RGB key color. If no color was set Black is returned. Use **usesKeying** to evaluate whether the MMFLMImageRep uses this color for keying.

See also: -setYUVKeyColor:, -setRGBKeyColor:, -usesKeying

setKeyColorFromPoint:

- setKeyColorFromPoint:(NXPoint*)*aPoint*

Sets the YUV and the RGB key color to the color *aPoint* has. *aPoint* has to be in the image region. If *aPoint* is not in the image region keying is set off and **nil** is returned. Otherwise returns **self**. Use this method as your preferred method to set key colors.

See also: -setYUVKeyColor:, -setRGBKeyColor:, -rgbKeyColor, -yuvKeyColor

setKeyingOn:

- setKeyingOn:(BOOL)*aBool*

If *aBool* is set to YES color keying is activated. Returns **self**.

See also: -usesKeying

setRGBKeyColor:

- setRGBKeyColor:(NXColor*)*aColor*

Sets the RGB key color. *aColor* is used to set the correct alpha channel in screen representation. Furthermore the according values for the YUV key color are calculated. Due to fact of conversion loss problems it may happen that a correct RGB color could not be found in the YUV color space. For that reason it is useful to set the YUV key color separately. Returns **self**.

See also: -setYUVKeyColor:, -rgbKeyColor

setYUVKeyColor:

- setYUVKeyColor:(SMColor*)*aColor*

Sets the YUV key color. *aColor* is used to set a transparent color when displaying an image on video out. The values for the according RGB key color aren't set. Returns **self**.

See also: setRGBKeyColor:, -setKeyColorFromPoint:, -yuvKeyColor

usesKeying

- (BOOL)**usesKeying**

Returns **YES** if the MMFLMImageRep uses key colors otherwise **NO** is returned.

yuvKeyColor

- (SMColor)**yuvKeyColor**

Returns the currently set YUV key color. If no color was set Black is returned. Use **usesKeying** to evaluate whether the MMFLMImageRep uses this color for keying.

See also: -setYUVKeyColor:, -setRGBKeyColor:, -usesKeying

Function Calls

SMConvertRGBToYUV

SMColor **SMConvertRGBToYUV(**NXColor *rgb***)**

Converts the NXColor *rgb* to SMColor and returns it.

See also: SMConvertYUVToRGB

SMConvertYUVToRGB

NXColor **SMConvertYUVToRGB(**SMColor *yuv***)**

Converts the SMColor *yuv* to NXColor and returns it.

See also: SMConvertRGBToYUV

SMTVControl

Inherits From: **Object**

Declared In: **smkit/SMTVControl.h**

Class Description

SMTVControl is an object that controls the tuner and the stereo device in the optional TV-Tuner of ScreenMachineII, or the build in tuner of the other VDOboards. With this class you can tune channels by setting frequencies and norms, set and request status of the stereo chip, and search for TV channels. Furthermore, you can store TV channels into memory, save them to and restore them from disc.
The **initFromControl:** method is the designated initializer for the SMTVControl class.
Note: Never use the **init** method - an initialized SMControl class is needed for most of the methods included in this object :

```
- setupNewTVControlForSMControl:sender
{
    unsigned short prog_nr;

    // check if an SMControl class was received
    if(![sender isKindOfClassNamed:"SMControl"])
    {
            return nil;
    }
    // initialize a new instance of SMTVControl
    tvControl=[[SMTVControl alloc] initFromControl:sender];

    if(tvControl)
    {
            // read file .stations in the users home directory
            if([tvControl readProgs:""])
            {
                    // we found a table
                    // now search the first valid channel
                    prog_nr=[tvControl nextProgFor:0 searchUp:YES];
                    // set this channel
                    [tvControl setProg:prog_nr];
            }
            return self;
    }
    return nil;
}
```

Searching channels

The best way to search for TV channels is to use the **afcSearch:withSystem:andNorm:** method. To work, this method needs an AFC-table given in the **setAFCTable:** methods. If the already existing tables do not contain enough information, you can add some values or create your own tables. For further information on AFC-tables see the class description of **AFCTables** and **AFCTableList**. If, for some reason, you can't work with the given tables, you can use the **search:withSystem:andNorm:** method. Be aware that this method takes much more time and it may not give proper results if not configured correctly for your country. You can configure the search with the **setSearch:withWidth: andWeight:** method. A third and very difficult way to search channels is to write your own search algorithm. See the **statusVT** method for information.

Storing channels

There are three ways to use the program table within this class. First, you can use the **setAFCTable: andPresetWithNorm:andSystem:** method which presets all channels with the information given by the **AFCTable**. The second way is to set every channel manually with the **storeProg:withFrequency:system:andNorm:** method. Finally, you can read a table stored somewhere on your disk by using **readProgs:**. You should preset the values from an **AFCTable** - the user then has the chance to change these settings at any time and save them on the harddisk. For an example see the MMTV or the VDOtv application.

Instance Variables

id theSMTVView;

theSMTVView	VideoText view (only interesting for VideoText).

Method Types

- afcSearch:withSystem:andNorm:
- frequency
- initFromControl:
+ isTunerConnectedAtControl:
- nextProgFor:searchUp:
- norm
- progFrequency:
- progName:
- progNorm:
- progSystem:
- progVisible:
- readProgs:
- search:withSystem:andNorm:
- setAFCTable:
- setAFCTable:andPresetWithNorm:andSystem:
- setChannel:withSystem:andNorm:
- setMonoStereo:
- setProg:

- setSearch:withWidth:andWeight:
- setVisible:theProg:
- statusAudio
- statusTV
- statusVT
- storeProg:withFrequency:system:andNorm:
- storeProg:withFrequency:system
  norm:andName:
- tunerType
- writeProgs:

Class Methods

isTunerConnectedAtControl:

+ (BOOL)**isTunerConnectedAtControl:**(id)*aControl*

Returns **YES** if a TV-Tuner is connected to *aControl*. *aControl* has to be an initialized instance of SMControl/MMControl class.

See also: -initFromControl:

Instance Methods

afcSearch:withSystem:andNorm:

- (int)**afcSearch:**(int)*frequenz* **withSystem:**(int)*system* **andNorm:**(int)*norm*

Searches the next valid TV channel on all AFC-frequencies starting at frequenz with *system* and *norm*. The frequency has to be in a range from 4825 to 85525 (48.25 - 855.25 MHz). For values of system see **system** (SMControl), for values of norm see **norm**. If no error occurred the found frequency is returned, otherwise the error is returned. For a further description of all errors see **Errors** at the end of this chapter. Before using this method an AFC-table has to be set. In every non-error case the returned frequency is set.

See also: -setAFCTable:, -setAFCTable:andPresetWithNorm:andSystem:, -search:withSystem:andNorm:

frequency

- (int)**frequency**

Returns the currently set frequency of the TV-Tuner. The frequency is in a range from 4825 to 85525 (48.25 - 855.25 MHz).

See also: -progFrequency:, setChannel:withSystem:andNorm:

initFromControl:

- initFromControl:(id)*aControl*

Initializes and returns the receiver, a new SMTVControl instance with a set frequency of 48.25 MHz and a norm of SMTV_NORM_BG. *aControl* has to be an initialized instance of SMControl class. Before using **afcSearch:withSystem:andNorm:** you have to set a AFC-Table with the **setAFCTable:** method.

See also: -isTunerConnectedAtControl:

nextProgFor:searchUp:

- (int)**nextProgFor:**(int)*aProg* **searchUp:**(BOOL)*aBool*

Returns the next/previous (*aBool* equals YES/NO) visible channel starting at *aProg*. If no channel is set visible, or *aProg* is the only visible channel, *aProg* is returned. In case of overflow (larger 99/lower 0), the search starts over at the beginning/end of the program table.

See also: -setVisible:theProg, -progVisible:

norm

- (int)**norm**

Returns the currently set norm of the TV-Tuner. Possible return values (depending on the type of the TV-Tuner) are :

SMTV_NORM_M	M
SMTV_NORM_BG	B/G
SMTV_NORM_L	L
SMTV_NORM_Li	L'
SMTV_NORM_I	I

The norm of a TV system describes the frequency distance between picture and sound carrier. Using wrong norms will result in no sound with a good video picture or good sound with no picture. If you are not sure about the standard or the norm used in your country ask your local television/radio dealer or simply experiment until the correct one is found.

See also: -progNorm:, setChannel:withSystem:andNorm:

progFrequency:

- (int)**progFrequency:**(int)*prognr*

Returns the frequency of the stored channel *prognr*. The frequency is in a range from 4825 to 85525 (48.25 - 855.25 MHz). A return value of -1 means that no frequency is currently set for *prognr*. All other negative return values indicate that an error occurred. For a further description of all errors see **Errors** at the end of this chapter.

See also: -storeProg:withFrequency:system:andNorm:, -frequency

progName:

- (char*)**progName:**(int)prognr

Returns the name of the stored channel *prognr*. The name can contain a maximum count of 5 characters. If no name is currently set for *prognr* an empty string is returned. A return value of **NULL** indicates that an error occurred.

See also: -storeProg:withFrequency:system:andNorm:andName:

progNorm:

- (int)**progNorm:**(int)prognr

Returns the norm of the stored channel *prognr*. For the possible return values see **norm**. A return value of -1 means that no norm is currently set for *prognr*. All other negative return values indicate that an error occurred. For a further description of all errors see **Errors** at the end of this chapter.

See also: -storeProg:withFrequency:system:andNorm:, -norm

progSystem:

- (int)**progSystem:**(int)*prognr*

Returns the system of the stored channel *prognr*. For the possible return values see **system** (SMControl). A return value of -1 means that no system is currently set for *prognr*. All other negative return values indicate that an error occurred. For a further description of all errors see **Errors** at the end of this chapter.

See also: -storeProg:withFrequency:system:andNorm:, -system (SMControl)

progVisible:

- (BOOL)**progVisible:**(int)*prognr*

Returns **YES** if the stored channel *prognr* is set to visible. You needn't use this method when switching channels with the **nextProgFor:searchUp:** method. By default (channel *prognr* hasn't been stored yet) **NO** is returned.

See also: -setVisible:theProg:, -storeProg:withFrequency:system:andNorm:

readProgs:

- (int)**readProgs:**(char*)*dateiname*

Reads an already stored program table from harddisk. *dateiname* should contain the complete filename of the program table you want to read. If *dateiname* is an empty string (""), or could not be read from harddisk, this method tries to open the file '.stations' in the users home directory. A return value of 0 means that an error occurred and no table was read.

See also: -writeProgs:, -storeProg:withFrequency:system:andNorm:

search:withSystem:andNorm:

- (int)**search:**(int)*frequenz* **withSystem:**(int)*system* **andNorm:**(int)*norm*

Searches the next valid TV channel starting at *frequenz* with system *system* and norm *norm*. The frequency has to be in a range from 4825 to 85525 (48.25 - 855.25 MHz). For values of *system* see **system** (SMControl), for values of *norm* see **norm**. If no error occurred the found frequency is returned, otherwise the error is returned. For a further description of all errors see **Errors** at the end of this chapter. If the returned frequency equals *frequenz*, is larger than frequenz plus *deltafrequenz* (**setSearch:::**),or is larger than 85525, no valid channel has been found. In every non-error case the returned frequency is set.

Note : This method does not start over at the beginning of the frequency spectrum. You have to do so by your own.
In comparison to **afcSearch:::**,this is a distinctly slower method to find valid TV channels because all frequencies are scanned. You should only use this method if you don't have an AFC-table for your country or if the channels your searching for are not based on the AFC-frequencies of your country.

Warning : This method can't be interrupted and it can take quite a long time to scan all frequencies.

See also: -setSearch:withWidth:andWeight:, -afcSearch:withSystem:andNorm:

setAFCTable:

- setAFCTable:(id)*aTable*

Sets an AFC-table used for the **afcSearch:::** method. *aTable* has to be an initialized instance of SMTV_AFC_Table class. *aTable* will not be freed when freeing the SMTVControl class. Returns **self**.

See also: -afcSearch:withSystem:andNorm:, -setAFCTable:andPresetWithNorm:andSystem:

setAFCTable:andPresetWithNorm:andSystem:

- **setAFCTable:**(id)*aTable* **andPresetWithNorm:**(int)*aNorm* **andSystem:**(int)*aSystem*

Sets an AFC-table used for the **afcSearch:::** method. *aTable* has to be an initialized instance of SMTV_AFC_Table class. In comparison to the function above, this method presets the program table with the values found in the AFC-table. In the AFC-table there is no information concerning the standard or the norm to use. The norm *norm* and system *system* will be set for all available frequencies. The available frequencies are set to visible. *aTable* will not be freed when freeing the SMTVControl class. Returns *self*.

Warning : Only use this method before making changes to the program table. All values of the program table may be destroyed.

See also: -afcSearch:withSystem:andNorm:, -setAFCTable:, setVisible:theProg:

setChannel:withSystem:andNorm:

- (int)**setChannel:**(int)*frequenz* **withSystem:**(int)*system* **andNorm:**(int)*norm*

With this method you can set the *frequenz* (frequency), the system and the norm directly. The frequency has to be in a range from 4825 to 85525 (48.25 - 855.25 MHz). For values of *system* see **system** (SMControl), for values of *norm* see **norm**. If no error occurred SMTV_OK is returned, otherwise the error is returned. For a further description of all errors see **Errors** at the end of this chapter.

See also: -frequency, -norm, -system (SMControl)

setMonoStereo:

- (int)**setMonoStereo:**(int)*monster*

Sets the stereo chip on the TV-Tuner board. This option is not available for all boards. Possible values for *monster* are :

SMTV_MONO	Sets to mono sound.
SMTV_STEREO	Sets to stereo sound if possible.
SMTV_A_CHANNEL	Sets to left channel (only possible if twochannel is detected).
SMTV_B_CHANNEL	Sets to left channel (only possible if twochannel is detected).

On success SMTV_OK else an error is returned. For a further description of all errors see **Errors** at the end of this chapter.

Note: The Movie Machine /Pro has only a mono channel in the NTSC version.

See also: -statusAudio

setProg:

- (int)**setProg:**(int)*prognr*

Sets the channel that is stored in the program table at position *prognr*. Be aware that not only the frequency but also the currently set system and norm may change. If no error occurred SMTV_OK otherwise the error is returned. For a further description of all errors see **Errors** at the end of this chapter.

See also: -progFrequency:, -progNorm:, -progSystem:, -progName:, -readProgs:, -storeProg:withFrequency:system:andNorm:

setSearch:withWidth:andWeight:

- (int)**setSearch:**(int)*deltafrequenz* **withWidth:**(int)*breite* **andWeight:**(int)*gewichtung*

This methods sets the limit and the adjustments for the **search:::** method. The default values for *breite* and *gewichtung* (see below) are experienced for german television channels and may not work in your country. You needn't set values for using **search:::** the first time.

deltafrequenz has a range from 0 to 80000 (default 80000 relates to 800MHz) and describes a range of frequency in which should be searched for a valid channel. Keep in mind that a valid channel could only be detected properly if you scan all over to the end of this channel. The best picture quality of a channel you get somewhere in between the visible start and end frequency (see also *breite* and *gewichtung*). Use value of 0 to reset to default.

breite is the minimum range of frequency (in MHz * 100) a channel is detected as a valid TV channel. If you set this minimum to low (default is 150 equals 1.5MHz) you may detect frequencies where you 'can't find' a real TV channel. Use a value of 0 to set to the default.

gewichtung is a percentage value between 0 and 100 (default 75) and represents the frequency that is set and returned by **search:::** in between the found visible start and end frequency. So a value of 100 means to use the frequency at the end of the found channel range. Use a value of 0 to set to the default.

Example: The **search:::** method detects a valid TV signal between 50000 and 50200 (500 - 502 MHz). Your start frequency was 45000 (450 MHz) and your deltafrequenz has been set to 10000 (100 MHz). So the end of your range hasn't been reached. The width of the channel (*breite*) is larger then 150 (50200 - 50000 = 200) which means that a valid channel is detected. Because of the weight (*gewichtung*) of 75 a frequency of 50150 (501.5MHz) is set and returned.

In case of error the error is returned (For a further description of all errors see **Errors** at the end of this chapter) otherwise SMTV_OK.

See also: -search:withSystem:andNorm:

setVisible:theProg:

- (int)**setVisible:**(BOOL)*aBool* **theProg:**(int)*prognr*

If *aBool* is Yes, sets the stored channel *prognr* visible. This method gives you the opportunity to decide wether a stored channel should be switched to when using the **nextProgFor:searchUp:** method. In case of error the error is returned (For a further description of all errors see **Errors** at the end of this chapter) otherwise SMTV_OK.

See also: - progVisible, - nextProgFor:searchUp:, - storeProg:withFrequency:system:andNorm:

statusAudio

- (int)**statusAudio**

Returns the status of the stereo chip. A negative return value means that an error occurred. For a further description of all errors see **Errors** at the end of this chapter. Possible non negative return values are :

SMTV_MONO	Mono sound signal detected.
SMTV_STEREO	Stereo sound signal detected.
SMTV_TWOCHANNEL	Twochannel sound signal detected.
SMTV_NOIDENT	The signal couldn't be detected properly.

It takes the stereo chip a maximum time of about half a second after changing frequency to detect a proper audio signal. If waiting less time this method can return a wrong value (in the most cases not SMTV_NOIDENT).

See also: -setMonoStereo:

statusTV

- (int)**statusTV**

Returns the status of the TV-Tuner. If a valid TV channel frequency is already set a fine tuning can be done using this method. The lower three bits of the return value indicate :

< 2	Set frequency is to high.
= 2	Set frequency is ok.
> 2	Set frequency is to low.

You needn't use this method after using the search methods described above in this chapter. The search methods already do that. This method may be only useful to you if you develop your own search algorithm.

See also: -statusVT

statusVT

- (int)**statusVT**

Returns the status of the VideoText chip. A set Bit 0 (SMTV_TV_QUALITY_BIT) indicates that on the current frequency a valid video signal was detected. That means that this frequency has a valid picture information. A set Bit 1 (SMTV_VT_QUALITY_BIT) indicates a valid Video Text signal. Video Text is not available in most countries. Even if your TV-Tuner has no Video Text Chip on board you can use this method to determinate the quality of the currently set frequency.

You needn't use this method after using the search methods described above in this chapter. The search methods already do that. This method may be only useful to you if you develop your own search algorithm.

See also: -statusTV

storeProg:withFrequency:system:andNorm:

- (int)**storeProg:**(int)*prognr* **withFrequency:**(int)*frequenz* **system:**(int)*system* **andNorm:**(int)*norm*

Stores the frequency *frequenz*, the system *system* and the norm *norm* in the program table at position *prognr*. Positions from 0 to 99 are available. The range of the other parameters are described in the methods mentioned below (see also). Like the method **setVisible:theProg:** this method sets the position *prognr* to visible.

In case of error the error is returned (For a further description of all errors see **Errors** at the end of this chapter) otherwise SMTV_OK.

Note : Storage of a program channel means only to store it into the memory of the computer. If you want to store the program table to harddisk you have to use **writeProgs:** if you loaded your table with the **readProgs:** method.

See also: -storeProg:withFrequency:system:norm:andName:,-progFrequency:, -progNorm:, -progSystem:

storeProg:withFrequency:system:norm:andName:

- (int)**storeProg:**(int)*prognr* **withFrequency:**(int)*frequenz* **system:**(int)*system* **norm:**(int)*norm* **andName:**(char*)*aName*

Stores the frequency *frequenz* the system *system* the norm *norm* and the name *aName* in the program table at position *prognr*. Positions from 0 to 99 are available. *aName* should only be a short form of the channel name because only the first 5 characters are taken for storage. The range of the other parameters are described in the methods mentioned below (see also). Like the method **setVisible:theProg:** this method sets the position *prognr* to visible.

In case of error the error is returned (For a further description of all errors see **Errors** at the end of this chapter) otherwise SMTV_OK.

Note : Storage of a program channel means only to store it into the memory of the computer. If you want to store the program table to harddisk you have to use **writeProgs:** if you loaded your table with the **readProgs:** method.

See also: -storeProg:withFrequency:system:norm:,-progFrequency:, -progNorm:, -progSystem:, -progName:

tunerType

- (int)**tunerType**

Returns the type of the tuner. The different tuner types differ on the norm they support. Currently two types of tuners are available :

SMTV_GR_MODUL Supported norms : M, B/G, L
SMTV_UK_MODUL Supported norms : B/G, L, L', I
For a further description of norm see **norm**.

Note on the specific tuners of the different VDOboards:
The optional Tuner for the Screen Machine II is a multinorm tuner. It supports NTSC, PAL and SECAM systems.
The onboard tuners of the other VDOboards are single norm tuners, and either support NTSC,PAL or SECAM, depending on where you bought the board.

See also: +isTunerConnectedAtControl:

writeProgs:

- (int)**writeProgs:**(char*)*dateiname*

Writes an already stored program table from memory to harddisk. *dateiname* should contain the complete filename of the program table you want to write. If *dateiname* is an empty string ("") this methods tries to write the file '.stations' in the users home directory. A return value of 0 means that an error occurred and no table was written.

You needn't save every change of your program table to harddisk. Usually it is enough to save the program table when quitting your application.

See also: -readProgs:, -storeProg:withFrequency:system:andNorm:

Errors

SMTV_OK	No Error occurred.
SMTV_ERROR	General Error.
SMTV_I2CERROR	Bus Error (try again and check cables and connections).
SMTV_FREQUENCYERROR	Wrong frequency range.
SMTV_SYSTEMERROR	Wrong system value.
SMTV_NORMERROR	Wrong norm value.
SMTV_PROGRAMERROR	Wrong program channel number given.
SMTV_WEIGHTERROR	Wrong value for weight.
SMTV_MONOSTEREOERROR	Wrong value for the stereo chip given.

SMTV_AFC_Table

Inherits From: **Object**

Declared In: **smkit/SMTV_AFC_Table.h**

Class Description

The SMTV_AFC_Table object gives you the opportunity to handle all entries in an AFC-table. You can get all frequencies of a county an the corresponding channel descriptors stored in an AFC-Table file. The **initTableFromFile:** method is the designated initializer for the SMTV_AFC_Table class. Normally you needn't initialize this class by your own. This is done by an initialized **SMTV_AFC_TableList**. You can get a specific table with a **getAFCTableWithName:**(SMTV_AFC_TableList) call. The argument for the call has to be one of the names returned in a **tableNamesForCountry:**(SMTV_AFC_TableList) method.

Creating your own AFC-Table

If you want to create your own AFC-Table here's a list of the supported arguments :

%%!AFC-Table *ver*	Has to be in the first line of an AFC-Table. *ver* is an optional argument to keep track of table versions.
%% Name *aName*	Name of the AFC-Table. *aName* has to be the same as the name placed in the Countries.afcTable file (See SMTV_AFC_TableList for further description). *aName* has to be quoted.
%% Entries *num*	*num* has to be the number of entries in the AFC-Table.
%% Begin	Between a Begin and End statement all table entries are placed. A table entry is a [TAB] separated pair of channel descriptor and channel frequency. The channel descriptor is the name of a channel. The frequency is in MHz. The entries should be sorted by frequency.
%% End	Marks the end of the table entries.

All other arguments followed by %% are ignored.

Instance Variables

int numEntries;
char tableName[1024];

numEntries	Number of frequencies that have been found.
tableName[1024]	Name of the AFC-table.

Method Types

- channelNameForFreq:
- channelNameForVal:
- count
- freqForVal:
- initTableFromFile:
- nextValForFreq:
- tableName

Instance Methods

channelNameForFreq:

- (char*)**channelNameForFreq:**(int)*aFreq*

Returns the channel Descriptor for the frequency *aFreq*. If there is no known channel at the frequency *aFreq*, "?" is returned. The search range for this method is –3 MHz. The frequency has to be in a range from 4825 to 85525 (48.25 - 855.25 MHz).

See also: -channelNameForVal:

channelNameForVal:

- (char*)**channelNameForVal:**(int)*aVal*

Returns the channel Descriptor for the table entry *aVal*. If aVal is not in the table "" is returned.

See also: -channelNameForFreq:

count

- (unsigned)**count**

Returns the number of entries in the AFC-Table. The first entry is number 0.

See also: -initTableFromFile:

freqForVal:

- (int)**freqForVal:**(int)*aVal*

Return the frequency of table entry *aVal*. If aVal is not in the table -1 is returned.

See also: -nextValForFreq:

initTableFromFile:

- initTableFromFile:(const char*)*filename*

This method is the designated initializer for the SMTV_AFC_Table class. *filename* has to be the full path to an AFC-Table file. If the file can't be read or no valid entries are found the class will be freed and **nil** will be returned. Otherwise **self** is returned.

See also: -initForPath: (SMTV_AFC_TableList)

nextValForFreq:

- (int)**nextValForFreq:**(int)*aFreq*

Returns the next table entry whose frequency is larger than *aFreq*. If no larger frequency was found 0 is returned.

See also: -freqForVal:

tableName

- (char*)**tableName**

Returns the name of the AFC-Table stored in the AFC-Table file.

See also: -initTableFromFile:

SMTV_AFC_TableList

Inherits From: **List**

Declared In: **smkit/SMTV_AFC_TableList.h**

Class Description

The SMTV_AFC_TableList object gives you the oportunerty to handle all the countries and all the AFC-table corresponding to this countries. The **initForPath:** method is the designated initializer for the SMTV_AFC_TableList class. The argument for this method is the path where all the tables should be. This class contains a list of all AFC-tables and has methods to handle them (**tableNamesForCountry:**, **getAFCTableWithName:**). So you do not have to know anything about the SMTV_AFC_Table class to use the **afcSearch:** methods of SMTVControl class.

Creating your own AFC-Tables

If you want to use your own AFC-table (see SMTV_AFC_Table) you have to append the name of your table to the names already written at the end of the line of your country name ('Countries.afcTable' file). In the 'Countries.afcTable' file you can also add a new country but be aware that you have to increase the count of entries (%% Entries). The format of the 'Countries.afcTable' file and also of the other *.afcTable files are as we hope quite easy and self explanatory. The values of norm and system are explained in SMControl (**system**) and SMTVControl (**norm**).
To get an idea what you can do with this class examine the SMTV or the VDOtv application.

Instance Variables

int numEntries;

numEntries	Number of countries that have been found.

Method Types

- countries
- getAFCTableWithName:
- initForPath:
- normForCountry:
- nrOfCountries
- systemForCountry:
- tableNamesForCountry:

Instance Methods

countries

- (char**)**countries**

Returns a NULL-terminated list of country names found in the 'Countries.afcTable' file.

See also: -nrOfCountries

getAFCTableWithName:

- getAFCTableWithName:(const char*)*aName*

Returns an initialized SMTV_AFC_Table class named *aName*. If the table named *aName* couldn't be found **nil** is returned.

Warning : Do not free the returned table. It will be freed when freeing the SMTV_AFC_TableList class.

See also: -tableName (SMTV_AFC_Table)**, -initTableFromFile:** (SMTV_AFC_Table)

initForPath:

- initForPath:(const char*)*my_path*

This method is the designated initializer for the SMTV_AFC_TableList class. *my_path* should contain the full path to the AFC-table files and the 'Countries.afcTable' file. If either no tables or the no 'Countries.afcTable' files are found the class will be freed and **nil** will be returned. Otherwise **self** is returned.

See also: -initTableFromFile: (SMTV_AFC_Table)

normForCountry:

- (int)**normForCountry:**(const char*)*aCountry*

Returns the norm that is usually in use in the country *aCountry*. If *aCountry* wasn't found -1 is returned. For a description of all possible norms see the description of SMTVControl class.

See also: -norm (SMTVControl)**, -countries**

nrOfCountries

- (int)**nrOfCountries**

Returns the number of countries that have been found in the 'Countries.afcTable' file.

See also: -countries

systemForCountry:

- (int)**systemForCountry:**(const char*)*aCountry*

Returns the system that is usually in use in the country *aCountry*. If *aCountry* wasn't found -1 is returned. For a description of all possible systems see the description of SMControl class.

See also: system (SMControl)**, -countries**

tableNamesForCountry:

- (char**)**tableNamesForCountry:**(const char*)*aCountry*

Returns a NULL-terminated list of all the AFC-table names that are usually in use in the country *aCountry*. Every country has at least one corresponding AFC-table. But some have more (e.g. the USA has USA and USA Cable).

See also: -tableName (SMTV_AFC_Table)**, -countries**

Chapter 3

SMPalette

InterfaceBuilder

The InterfaceBuilder is a development tool where the most used objects of user interfaces are provided as instantiated objects. The available objects are grouped by their range of use and are linked into InterfaceBuilder through so called palettes. These objects can be used by dragging them from the palettes window into the appropriate working area of an InterfaceBuilder document and simply dropping them. An object added to a user interface is fully instantiated, which means they have their own set of object variables. For this reason, settings made for these objects are saved through the object and restored when the user interface is loaded by an application.

An object is able to have outlets and actions. Outlets are object variables of type <id> and allow the object to establish connections to other objects. Actions are object-methods which can be called directly or by events of other objects.

After the initialization of an object the value of every outlet is normally NULL, so there has to be a possibility to make other objects known to an object by assigning values to its outlets. To do this, simply select an object and drag the mouse over the target object while pressing the control key and the left mouse button. Release the buttons over the target object and InterfaceBuilder will offer the possibility to make the target object an outlet of the source object through the Connection Inspector. InterfaceBuilder is drawing a line (connection), between the object where you started to drag the mouse and the object found under the current mouse location to visualize the connection you are setting. Outlets can only be assigned if the source object is not a subclass of Control, because these objects do not normally have outlets but are able to call methods in other objects. To trigger an action, set a connection between two objects as described above, where the starting object is a subclass of Control, e.g. a Button. You are then able to set the action method to be called in the activated Connection Inspector.

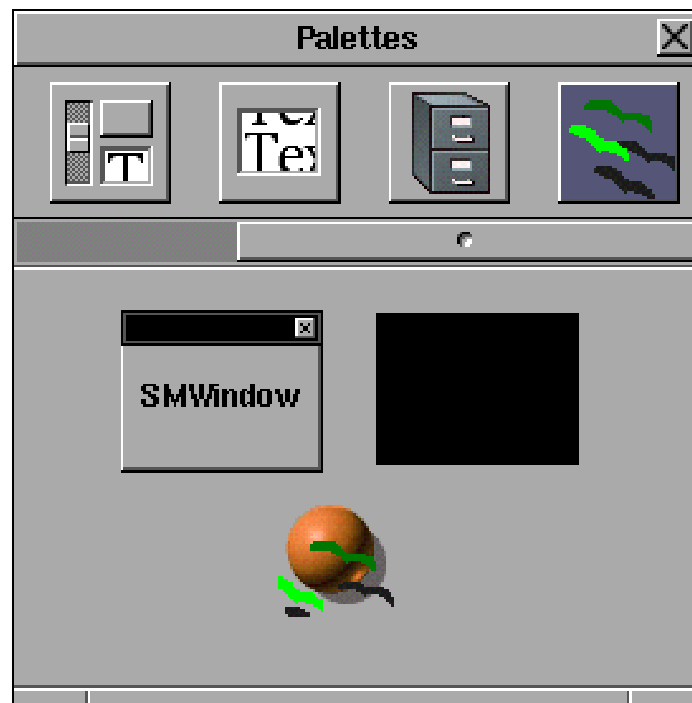
Through objects provided in InterfaceBuilder and objects made known to InterfaceBuilder you are able to give the designed user interface functionality just by dragging and dropping objects and by 'drawing' connections between objects.

The outlets and action methods set in an InterfaceBuilder session are saved in special objects embedded in the saved InterfaceBuilder files and are restored after these files are loaded in an application.

Palettes

Objects having related characteristics or specialized objects derived from the same class are grouped and provided in so called palettes shown in the InterfaceBuilders palettes window. If you are activating InterfaceBuilder for the first time there are four palettes available to the user.

In InterfaceBuilder's Tool Menu you will find a submenu entry called Load Palette which offers the possibility to add palettes to the palettes window. Adding palettes to InterfaceBuilder offers new sets of objects to the user and allows an extended and more detailed, easy to use way of designing user interfaces with InterfaceBuilder. The ability to add a palette to InterfaceBuilder was used to provide SMPalette, a set of three customized objects to build simple applications using VDOboards.



The SMPalette provides three basic objects SMControl, SMWindow and SMPaletteView (a subclass of SMView). These three objects are the minimum requirement to build a simple VDOboard application.

The SMWindow is a specialized window class designed to manage VDOboard videos through SMPaletteViews. A special window class is needed to manage the visibility of the live video while the window is moved, rearranged or hidden.

The SMControl class is the fundamental object to control the VDOboard settings and to manage the views assigned to one VDOboard.

The SMPaletteView is a subclass of SMView where action methods for InterfaceBuilder use were added.

The action methods of the SMPaletteView are the following:

Method:	**accepted Value:**
Mode	
- setVideoOnOff:sender	(BOOL) [sender state]
- setStill:sender;	(BOOL) [sender state]
- setColorOn:sender;	(BOOL) [sender state]
- setMixerOn:sender	(BOOL) [sender state]
- setChromaInvert:sender;	(BOOL) [sender state]
- setLumaInvert:sender;	(BOOL) [sender state]
- setFlipped:sender;	(BOOL) [sender state]
- setVideoInput:sender;	(unsigned char) [[sender selectedCell] tag]
- setVideoFields:sender;	(unsigned char) [[sender selectedCell] tag]
Quality	
- setLumaIntensity:sender;	(float) [sender floatValue]
- setChromaIntensity:sender;	(float) [sender floatValue]
- setRedGain:sender;	(float) [sender floatValue]
- setGreenGain:sender;	(float) [sender floatValue]
- setBlueGain:sender;	(float) [sender floatValue]
- setBrightness:sender;	(float) [sender floatValue]
- setSaturation:sender;	(float) [sender floatValue]
- setHue:sender;	(float) [sender floatValue]
- setContrast:sender;	(float) [sender floatValue]
- setSharpness:sender;	(float) [sender floatValue]
- setLumaSharpness:sender;	(float) [sender floatValue]
Audio	
- setAudioMute:sender;	(BOOL) [sender state]
- setAudioVolume:sender;	(float) [sender floatValue]
- setAudioBalance:sender;	(float) [sender floatValue]
- setAudioFader:sender;	(float) [sender floatValue]
- setAudioTreble:sender;	(float) [sender floatValue]
- setAudioBass:sender;	(float) [sender floatValue]
Image	
- saveImageAs:sender;	

All these methods are made available through the SMControl class known to the SMPalette-View.

If the column Value taken: says (BOOL) [sender state], the control object connected to this action has to be a Button or any other Control which can be in the state representing on and off. If the column Value taken: says (float) [sender floatValue], the control object connected to this action has to be a Control offering the action floatValue which represents the state of the control

in a continuous range (e.g. the Slider provided by InterfaceBuilder). The range of the control object should be set to a minimum of 0.0 and a maximum of 1.0.
If the column Value taken: says (unsigned char) [[sender selectedCell] tag], the control object connected to this action has to be a Matrix of radio buttons. The buttons in this matrix should have continuous tags.

```
e.g. setVideoInput:tags from 0 to 3
```

A tag value of 0 represents SM_INPUT_BLACK, 1 = SM_INPUT_RED, 2 = SM_INPUT_YELLOW and a tag value of 3 stands for SM_INPUT_SVHS.

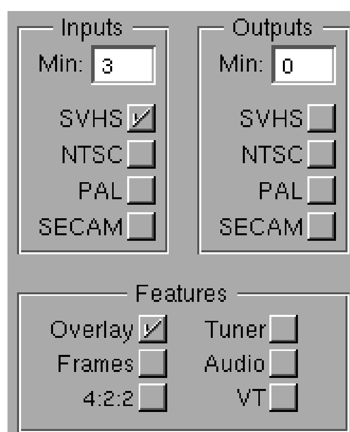
```
e.g. setVideoFields: tags from 1 to 3
```

A tag value of 1 represents SM_FRAME_ODD, 2 = SM_FRAME_EVEN, and 3 = SM_FRAME_BOTH.
The method saveImageAs: does not use its argument sender. When called, this method allows you to save the current frame as a TIFF-image. This method can be called through any control (e.g. Button, MenuItem …).

Features

SMControl

The SMControl has a customized inspector to indicate what type of board should be used. Enabling a switch means the board has to support this feature. Disabling it means 'I don't care about this feature'. If the selected features don't match the installed board types on startup time of your application, a panel appears to indicate that no suitable board was found.



The different features are described below :

Inputs : Select the number and types of inputs the desired board has to have.
If you don't care just select no types and set the number of inputs to 0.

Outputs : Select the number and types of outputs the desired board has to have.
If you don't care just select no types and set the number of outputs to 0.

The desired board has to have the Features :

- Overlay It can display live overlay . Currently all boards support this feature.
- Frames The board has enough memory do display both video fields at once (1 MB).
- 4:2:2 YUV-Mode 6 is supported, otherwise 4:1:1.
- Tuner It has an on board TV-Tuner or an external TV-Tuner is connected.
- Audio It supports audio features.
- VT It has an on board or external TV-Tuner with a VideoText chip on it.

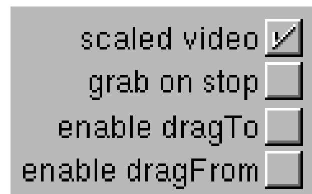
Check this tabel to find out about the features of the boards.

	Overlay	Full Frames	YUV 4 2 2	YUV 4 1 1	Tuner	Audio	VT	Input	Output	S-VHS
Screen Machine II	X	X	X		O	O	O	3		X
Screen Machine II Rev. 2	X	X	X		O	O	O	3		X
Movie Machine	X			X	X	X (tv out)	X	2 + 1		
Movie Machine PRO	X			X	X	X (tv out)		2 + 1	X	
Movie Machine II	X		X		X	X (tv out)	X	2 + 2	X	X (in & out)
FPS 60	X		X			O (mpeg)		2 + 2		X

X = supported O = optional

Note:
Through SMPalette you are only able to make the SMPaletteView do something, but there is no possibility to get a direct response. This implies that you are not able to query the SMPalette-View about the states or values which you can set using the SMPaletteViews action methods. For this you have to be sure about the options installed or the methods you are using.
For example a button connected to the setAudioMute:sender method in the state of being selected would cause the SMPaletteView turning the sound on through the SMControl class. However sound can only be turned on if there is a sound option installed to your VDOboard.

SMPaletteView



The SMPaletteView has a customized inspector to indicate and change its behaviour.

- scaled video If the SMPaletteView is resized, the video will be adjusted.
- grab on stop The last frame of the video will be grabbed and displayed if the video display is stopped.
- enable dragTo enables dragging of .flm and .tiff files into the SMPaletteView. The dragged files will be displayed through the VDOboard.
- enable dragFrom Enables dragging of stopped images out of the SMPaletteView if grab on stop has been set.

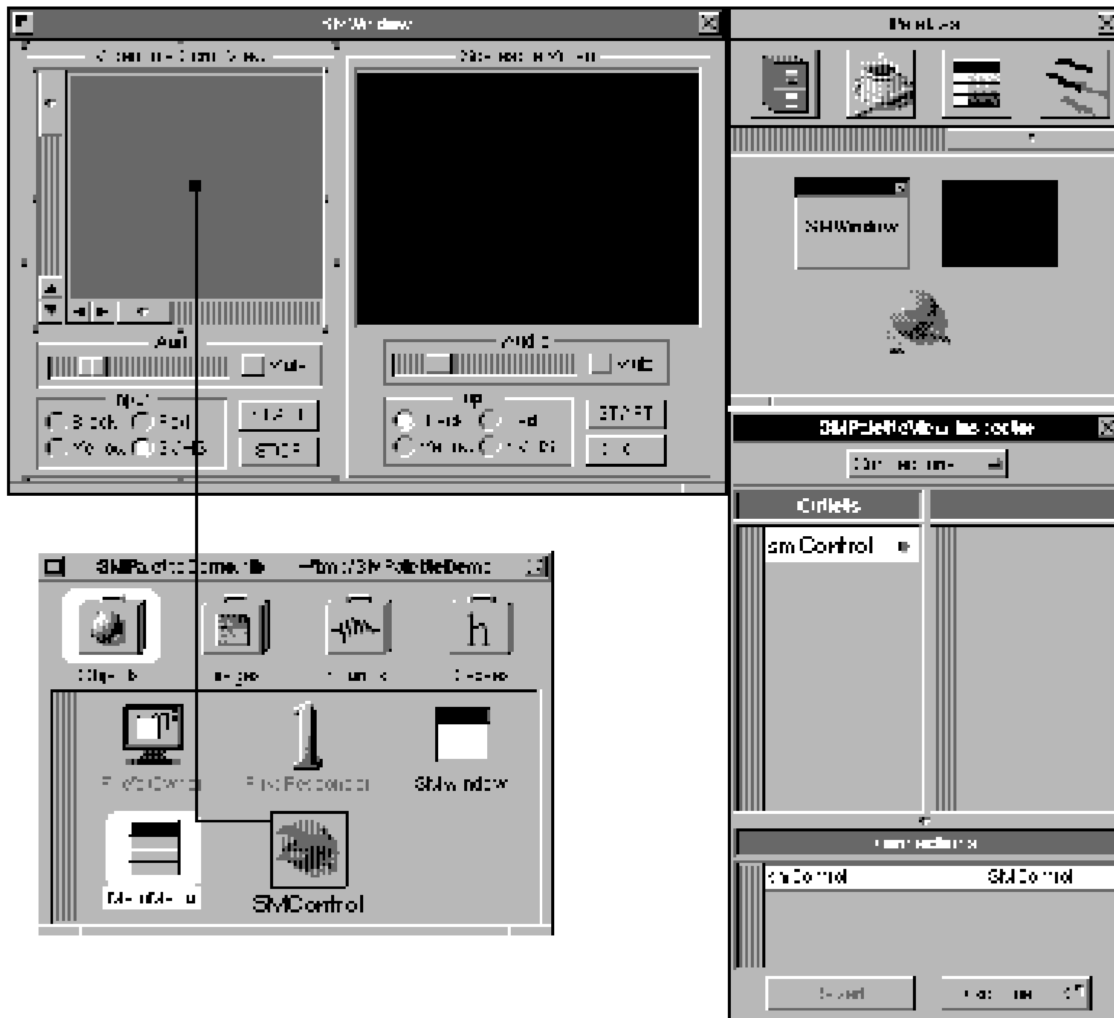
Default handling:

- Video Input allows the user to select a default video input for this SMPaletteView.
- VideoFields allows a default setting of the input frame type used by the SMControl object to display live video in this SMPaletteView.
- Audio mute indicates whether audio will be turned on or off for this SMPaletteView.
- Volume allows setting of a default volume for this SMPaletteView.

The audio defaults will only take effect if an audio option is installed.
Settings of the default handling will be set when the SMPaletteView receives a start: message or if the SMPaletteView is activated again when you are using multiple SMPaletteViews. Every action method changing these values will update the corresponding default setting used while the application is running.

Building an Application

To build a simple VDOboard application you need at least one of every object provided in the SMPalette. There is no special order required to add these objects to your InterfaceBuilder document. For the following example it is just the way we are using the objects.



- First we need a SMWindow

There are two ways to get a SMWindow. The first way is to change the class of a window to SMWindow in the Window Inspector and the second way is to drag the SMWindow symbol from the SMPalette into your working area, exactly the same way you would open a new Window from the standard palettes provided by InterfaceBuilder.

- The next step is to add a SMPaletteView into the SMWindow

To get a SMPaletteView, drag the symbol from the SMPalette into the content area of a SMWindow. InterfaceBuilder won't prevent you from dragging a SMPaletteView into a normal

Window, so be careful to change the class of the Window to SMWindow, otherwise the two objects won't cooperate.

- The last object needed is a SMControl object.

A SMControl object has to be dragged into the file window of the InterfaceBuilder document you are working on. You should not use more SMControl objects than VDOboards available on your computer.

- Create connections

If you have a SMWindow including a SMPaletteView and a SMControl object, you have to set one connection to enable the new objects to communicate and to cooperate. Connect the SMPaletteView to the SMControl object and make the SMControl object the smControl outlet of the SMPaletteView.

Minimum needed connections:

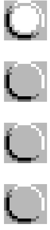
    SMPaletteView smControl: ----> SMControl

If these connections are made, the SMWindow is able to control the video displayed in the SMPaletteView through the SMControl known by the SMPaletteView.

- Start Button

If you add a button to your interface which is connected to the start: action of the SMPaletteView and a video source is connected to the "black input" of the VDOboard you could go into InterfaceBuilders test mode and press the button to start the video display.
Obviously it is not much work to create a simple application using a VDOboard with the classes provided by the SMPalette.

(See the examples for more information on building applications using InterfaceBuilder)



MAKE

IT

VIDEO

Chapter 4

Appendix

Variable Types and Constants

SMChromaSpace

DECLARED IN **smkit/SMControl.h**

SYNOPSIS

```
typedef struct _SMChromaSpace {
    float Umin;
    float Umax;
    float Vmin;
    float Vmax;
    float Ymin;
    float Ymax;
} SMChromaSpace;
```

DESCRIPTION

SMChromaspace is used to describe a color space for chroma and luma keying. Each of the colorspaces components limits can be specified. The range is between 0.0 and 1.0.

SMColor

DECLARED IN **smkit/SMFLMImageRep.h**

SYNOPSIS

```
typedef struct _SMColor{
    unsigned char Y;
    char U;
    char V;
} SMColor;
```

DESCRIPTION

SMColor is used to describe a color for keying images on video out.

Television system

DECLARED IN **smkit/SMControl.h**

SYNOPSIS

```
System            Value
SM_SYSTEM_NTSC    0
SM_SYSTEM_PAL     1
SM_SYSTEM_SECAM   2
```

DESCRIPTION

These constants define the television system used to decode the video signal.

Frame type

DECLARED IN **smkit/SMControl.h**

SYNOPSIS

```
Fields            Value
SM_FRAME_ODD      1
SM_FRAME_EVEN     2
SM_FRAME_BOTH     3
```

DESCRIPTION

These constants define the number of fields a digitized frame consists of.

Video Input

DECLARED IN **smkit/SMControl.h**

SYNOPSIS

```
Input             Value
SM_INPUT_BLACK    0
SM_INPUT_RED      1
SM_INPUT_YELLOW   2
SM_INPUT_SVHS     3
```

DESCRIPTION

These constants define the input of the VDOboard from which the video is digitized.

Video-Out Input

DECLARED IN **smkit/MMControl.h**

SYNOPSIS

```
Input                      Value
VIDEO_OUT_INPUT_NONE       0
VIDEO_OUT_INPUT_IMAGE      1
VIDEO_OUT_INPUT_A          2
VIDEO_OUT_INPUT_B          3
VIDEO_OUT_INPUT_TV         3
```

DESCRIPTION

These constants define the video out input of theVDOboards that support video out.

Memory mode

DECLARED IN **smkit/SMControl.h**

SYNOPSIS

```
Mode              Value
SM_MEM_NORM       1
SM_MEM_SPLIT      2
```

DESCRIPTION

These constants define the memory mode the VDOboard digitizes the images in.

Type of image representation

DECLARED IN **smkit/SMControl.h**

SYNOPSIS

```
ImageRep          Value
SM_IMAGE_YUV      1
SM_IMAGE_FLM      2
```

DESCRIPTION

These constants define the image representation used when calling the readImage: method.

Mode for YUV and FLM Imagereps

DECLARED IN **smkit/SMYUVImageRep.h**

SYNOPSIS

```
Mode            Value
SM_YUVMODE0     0
SM_YUVMODE1     1
SM_YUVMODE2     2
SM_YUVMODE3     3
SM_YUVMODE4     4
SM_YUVMODE6     6
SM_RGBMODE      8
```

DESCRIPTION

These constants define the mode in which the data is present in the YUV and FLM image representations.

Audio Input

DECLARED IN **smkit/SMControl.h**

SYNOPSIS

```
Input                    Value
SM_AUDIO_INPUT_BLACK     0
SM_AUDIO_INPUT_RED       1
SM_AUDIO_INPUT_YELLOW    2
SM_AUDIO_INPUT_SVHS      0
```

DESCRIPTION

These constants define the input of the optional audio-on-bracket or TV-tuner used .

Audio Signal Type

DECLARED IN **smkit/SMTVControl.h**

SYNOPSIS

```
Input                      Value
SMTV_MONO                  2
SMTV_STEREO                3
SMTV_A_CHANNEL             4
SMTV_B_CHANNEL             5
SMTV_TWOCHANNEL            6
SMTV_NOIDENT               7
```

DESCRIPTION

These constants define the audio signal type of the TV-tuner used .

Television norm (audio standard)

DECLARED IN **smkit/SMTVControl.h**

SYNOPSIS

```
System           Value
SMTV_NORM_M      0
SMTV_NORM_BG     1
SMTV_NORM_L      2
SMTV_NORM_Li     3
SMTV_NORM_I      4
```

DESCRIPTION

These constants define the television norm used to decode the audio signal.

Tuner Type

DECLARED IN **smkit/SMTVControl.h**

SYNOPSIS

```
System           Value
SMTV_GR_MODUL    0
SMTV_UK_MODUL    1
```

DESCRIPTION

These constants define the TV-tuner type.

Tuner Return Values

DECLARED IN **smkit/SMTVControl.h**

SYNOPSIS

```
System                    Value
SMTV_OK                   1
SMTV_ERROR                0
SMTV_I2CERROR             -1
SMTV_FREQUENCYERROR       -2
SMTV_SYSTEMERROR          -3
SMTV_NORMERROR            -4
SMTV_PROGRAMERROR         -5
SMTV_WEIGHTERROR          -6
SMTV_MONOSTEREOERROR      -7
```

DESCRIPTION

These constants define the possible return values of the SMTVContol class.

Board types

DECLARED IN **smkit/VDOboards.h**

SYNOPSIS

```
Board                     Value
VMCBOARD_MMPRO            0x00000001
VMCBOARD_SMII8            0x00000002
VMCBOARD_MM               0x00000008
VMCBOARD_MMII             0x00000010
VMCBOARD_FPS60            0x00000020
VMCBOARD_ALL_BOARDS       0x000000FF
```

DESCRIPTION

Specifies the VDOboard type.

Board features

DECLARED IN **smkit/VDOboards.h**

SYNOPSIS

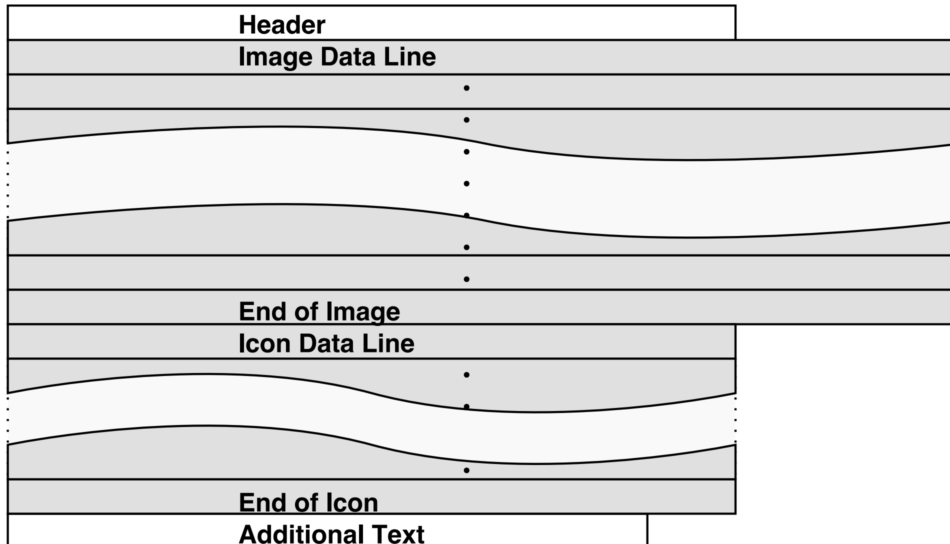
```
System                    Value
VMCBOARD_INPUTS           0x00000700
VMCBOARD_INPUT_SVHS       0x00000800
VMCBOARD_INPUT_NTSC       0x00001000
VMCBOARD_INPUT_PAL        0x00002000
VMCBOARD_INPUT_SECAM      0x00004000

VMCBOARD_OUTPUTS          0x00070000
VMCBOARD_OUTPUT_SVHS      0x00080000
VMCBOARD_OUTPUT_NTSC      0x00100000
VMCBOARD_OUTPUT_PAL       0x00200000
VMCBOARD_OUTPUT_SECAM     0x00400000

VMCBOARD_AUDIO            0x01000000
VMCBOARD_TUNER            0x02000000
VMCBOARD_VT               0x04000000
VMCBOARD_OVERLAY          0x08000000
VMCBOARD_422              0x10000000
VMCBOARD_SQP              0x20000000
VMCBOARD_FRAMES           0x40000000
VMCBOARD_LEILA            0x80000000
```

DESCRIPTION

These constants define the features of the individual VDOboards. The first and second set describe the number of inputs or outputs the board has, as well what characteristics the in or outouts have. e.g if they support NTSC or PAL. The third set describes the general features of a board.

The FLM file format

File Structure



Every FLM file consist of the following main elements :

- header
- YUV image data with different ordering
- icon of the image
- additional text info

Header
The header is described in the previos figure.

Image Data
The image data consists of a certain number of rows depending on the YUV mode and the same number of lines like the original video image. The modes and rows are described in the next figure.

Icon
Each file also has an icon of its image at the end. The Icon always has same format and size. (60 pixels height, 80 pixels width, YUV mode 0)

Text
Flm files may contain additional text of any length. The offset and length are set in the header.

File Header

Header of the FLM File Format

Nativ PC byte ordering.
64 Bytes, the first five always contains the string "SM1.0".

This figure shows :
- offset
- meaning
- example hex data
- example interpretation

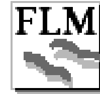
Offset	Meaning	Hex	Interpretation
0	FLM Image Header [LSB]	53	S
1	•	4D	M
2	•	31	1
3	•	2E	.
4	FLM Image Header [MSB]	30	0
5	Text End	1A	
6	Icon Offset [LSB]	80	
7	•	A0	
8	•	01	
9	Icon Offset [MSB]	00	
10	Image Width [LSB]	20	
11	Image Width [MSB]	01	
12	Image Height [LSB]	B9	
13	Image Height [MSB]	00	
14	YUV Mode	06	
15	YUV Length [LSB]	40	
16	YUV Length [MSB]	02	
17	Y Bits	08	
18	U Bits	08	
19	V Bits	08	
20	Compression Mode	00	
21	Old Compression	00	
22	Source Mode [LSB]	01	
23	Source Mode [MSB]	00	
24	Icon Length [LSB]	00	
25	Icon Length [MSB]	00	
26	Text Offset [LSB]	00	
27	•	00	
28	•	00	
29	Text Offset [MSB]	00	
30	Text Length [LSB]	00	
31	Text Length [MSB]	00	
32	Contrast [LSB]	21	
33	Contrast [MSB]	00	
34	Brightness [LSB]	24	
35	Brightness [MSB]	00	
36	Saturation [LSB]	24	
37	Saturation [MSB]	00	
38	Hue [LSB]	3F	
39	Hue [MSB]	00	
40	Red [LSB]	1F	
41	Red [MSB]	00	
42	Green [LSB]	1F	
43	Green [MSB]	00	
44	Blue [LSB]	1F	
45	Blue [MSB]	00	
46	Reserved [LSB]	00	
47	•	00	
		00	
		00	
62	•	00	
63	Reseverd	00	

Header

Field	Value
Text:	SM1.0
Text Ende:	26
Icon Off:	106624
Image Width:	288
Image Height:	185
YUV Mode:	6
YUV Lenght:	576
Y Bits:	8
U Bits:	8
V Bits:	8
Compression Mode:	0
Old Compression:	0
Src Mode:	1
Icon Length:	0
Text Off:	0
Text Length:	0
Contrast:	33
Brightness:	36
Saturation:	36
Hue:	63
Rot:	31
Grün:	31
Blau:	31
Reserv.:	

Data Format

FLM Image Data Format



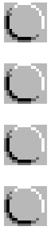
The YUV Ordering
A FBAS signal (Video and TV) consists of two "channels", one for the luminance Y and one for the two chrominance Ur and Vb signals. The green component is calculated out of the other two chrominance information. This was made because the human eye has more receptors for the luminance than for color.
Most video systems use the YUV format, so Screen Machine does. Therefore the fastes way to save an image is to store it in the original YUV format.

Layout		Mode
1 2 3 4	Video Pixel	**SM_YUVMODE0** YYVUYY (4 pixels; 4*Y,1*V,1*U)
Y_1 Y_2 V_{1234} U_{1234} Y_3 Y_4	File Bytes	Rarely used. mode factor : 1.5
RGBRGBRGBRGB	RGB Image Pixel	
1	Video Pixel	**SM_YUVMODE1** Y (1 pixel; 1*Y)
Y	File Bytes	Shortes mode, greyscale image. mode factor : 1.0
RGB	RGB Image Pixel	
1 2	Video Pixel	**SM_YUVMODE2** YYVU (2 pixels; 2*Y,1*V,1*U)
Y_1 Y_2 V_{12} U_{12}	File Bytes	Same like mode 6, but different byte ordering.
RGBRGB	RGB Image Pixel	mode factor : 2.0
1	Video Pixel	**SM_YUVMODE3** YVU (1 pixel; 1*Y,1*U,1*V)
Y V U	File Bytes	Largest but exactest format. mode factor : 3.0
RGB	RGB Image Pixel	
1 2	Video Pixel	**SM_YUVMODE6** YUYV (2 pixels; 2*Y,1*V,1*U)
Y_1 U_{12} Y_2 V_{12}	File Bytes	Screen Machine internal format. Commonly used.
RGBRGB	RGB Image Pixel	mode factor : 2.0

YUV Image Data Lengt : (video pixels x mode factor) x lines = bytes

FLM file length :

Header	64	bytes
Image	vp x mf x l	bytes
Text	text length	bytes
+ Icon	7 680	bytes
= absolute	sum	bytes



The YUV Color Model

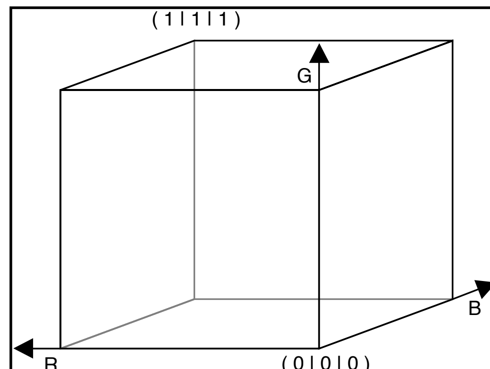
Explanation and Mapping

The YUV format predominates in the TV technology. As a result, the brightness information and color information are separated. To save bandwidth and also, therefore, data amount, the color information will be transmitted with less depth in relation to the brightness information. The reason for this is that the human eye can distinguish slighter variations in brightness than it can in color. Screen Machine also uses this format by fully digitizing the brightness for every picture point, but including more points per value in the color information. In this context, the format is 4:2:2.

The YUV color space is more “extensive” as that from computers which employ RGB color space. This means that color values can only be shown in approximation. The color space is therefore not completely conveyed, which is, for example, distinct in the chroma keying. While the visible RGB color space is a cube in the Cartesian coordinate system, the YUV (in English also the YIC) color space becomes a convex polyhedron.

$$R = Y + 1.370705\ V$$
$$G = Y - 0.698001\ V - 0.337633\ U$$
$$B = Y + 1.732446\ U$$

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 & 1.37 & 0 \\ 1 & -0.698 & -0.338 \\ 1 & 0 & 1.732 \end{bmatrix} \times \begin{bmatrix} Y \\ U \\ V \end{bmatrix}$$



Index

A

afcSearch:withSystem:andNorm: 107
Audio Input 137
Audio Signal Type 138
audioInput 59

B

balance 59
bandpassFilter 36
bass 59
bitsPerSample 86
blueGain 36
Board 139
Board features 140
Board types 139
boardAvailable: 26
boardsPresentOfType:withFeatures: 27
brightness 36

C

cancel: 63
canDragFrom 83
canDragTo 83
canLoadFromStream: 86, 96
channelNameForFreq: 118
channelNameForVal: 118
choose: 63
chooseSM:status:pids: 63
chromaIntensity 37
closeAllBoards 28
contrast 37
control 71
controlWillFree: 71
count 118
countries 121

D

defaultAudioMute 76
defaultFields 76
defaultInput 77
defaultVolume 77
doesGrabOnStop 71

E

enabledMovingInSMWindow: 65
enabledMovingInVideoView: 65

F

fader 60
Fields 135
FLM Data Format 143
FLM file format 141
FLM Header 142
Frame type 135
free 37, 86, 96
freqForVal: 119
frequency 107

G

getAFCTableWithName: 121
getCompression:andFactor: 87
getData: withMode: 87
getFeaturesOfBoard: 28
getHeader: 97
getIcon: 97
getImageSize: 37
getKernelVersion: minor: 37, 29
getSplitModeGrabSize: 38
getTIFFCompressionTypes:count: 86
getVideoFrame: 38

getVideoSize: 71
getWindowFrame: 38
getWriteHeader: 97
getZoomFrame: 38
grab 71
greenGain 38

H

hasAlpha 87
hasAudio 60
hasControl 72
hasIcon 97
hasVideo 66
horizontalOffset 38
hue 39

I

iconEnabled 97
image representation 136
imageDataInMode6 87
ImageRep 136
imageSizeFromMode: 87
imageType 39
imageUnfilteredFileTypes 96
imageUnfilteredPasteboardTypes 96
init 39, 88, 98
initData: 98
initData: pixelsWide: pixelsHigh: YUVMode: 88
initDataFromStream: pixelsWide: pixelsHigh: YUVMode: 88
initForPath: 121
initFrame:control 72
initFromControl: 108
initFromPasteboard: 89, 98
initFromStream: 98
initTableFromFile: 119
initTIFFDataFromStream: 89
Input 135, 136, 137, 138
inputFilter 39
inputFrameType 39
inputType 40
isAudioMute 60
isChromaInverted 40
isClippingOn 72
isColor 40
isFlipped 40
isHorizontalScaled 40
isInterlaced 41
isLumaInverted 41
isMixed 41
isOpaque 89
isPreFiltered 41
isScaledVideo 72
isStill 41
isTunerConnectedAtControl: 107
isVCRTimebase 41
isVideoOn 42

L

lumaIntensity 42

M

Memory mode136
memoryMode 42
Mode 136,137
mode 89
mosaicWidth 42

N

newForBoardNum: 29
newForBoardNum:fromZone: 29
newWithSelectionAndBoards :withFeatures: 29

newWithSelectionAndBoards:
withFeatures:fromZone: 29
nextProgFor:searchUp: 108
nextValForFreq: 119
noiseFilter 42, 43
norm 108
normForCountry: 121
nrOfCountries 121, 42
numberOfOutputs 43
numColors 89

O

outputType: 43

P

pll 43
posterization 43
progFrequency: 108
progName: 109
progNorm: 109
progSystem: 109
progVisible: 109

R

rbgKeyColor 103
read: 90
read: 98
readImage 44
readImageSelection: 44
readProgs: 110
redGain 44
redrawClips 72
registerVideoView: 66
reset 44
resetClips 73

S

saturation 44
saveImageAs: 77
search:withSystem:andNorm: 110
setAFCTable: 110
setAFCTable:andPresetWithNorm:
andSystem: 111
setAlpha: 90
setAudioBalance: 60, 77
setAudioBass: 60, 77
setAudioFader: 60, 77
setAudioInput: 61
setAudioMute: 78
setAudioTreble: 61, 78
setAudioVolume: 61, 78
setBandpassFilter: 45
setBitsPerSample: 90
setBlueGain: 45, 78
setBrightness: 45, 78
setChannel:withSystem:andNorm: 111
setChromaIntensity: 45, 78
setChromaInvert: 45, 78
setClippingOn: 73
setColorOn: 46, 79
setCompression:andFactor: 90
setContrast: 46, 79
setControl: 73
setDefaultAudioMute: 79
setDefaultFields: 79
setDefaultInput: 79
setDefaultVolume: 80
setDragFrom 83
setDragTo: 83
setFlipped: 46, 80
setGrabOnStop 73
setGreenGain: 46, 80
setHorizontalOffset: 47
setHorizontalScaled: 47

setHue: 47, 80
setIconEnabled: 99
setImageType: 47
setInputFilter: 47
setInputFrameType: 48
setInterlaced: 48
setKeyColorFromPoint: 103
setKeyingOn: 103
setLumaIntensity: 48, 80
setLumaInvert: 48, 80
setLumaSharpness: 81
setMemoryMode: 49
setMixerOn: 49, 81
setMonoStereo: 111
setMosaicWidth: 49
setMovingInSMWindow: 66
setMovingInVideoView: 66
setNoiseFilter: 50
setNumColors: 90
setOpaque: 91
setPixelsHigh: 91
setPixelsWide: 91
setPll: 50
setPosterization: 50
setPreFilter: 50
setProg: 112
setRedGain: 50, 81
setRGBKeyColor: 103
setSaturation: 51, 81
setScaledVideo: 74
setSearch:withWidth:andWeight: 112
setSharpness: 51, 81
setSize: 91
setSplitMemGrabSize: 51
setStill: 51, 81
setSystem: 52
setVCRTimebase: 52
setVerticalOffset: 52
setVideoFields: 82
setVideoFrame: 52
setVideoInput: 53, 82
setVideoOn: 53
setVideoOnOff: 82
setVideoOut: 56
setVideoOutInput: 56
setVideoOutInvert: 56
setVideoOutScaled: 56
setVideoOutSystem: 56
setVideoOutXofs: 57
setVideoOutYofs: 57
setVideoSize: 74
setVisible:theProg: 113
setWindowFrame: 53
setWriteHeader: 99
setWriteText: 99
setYUVKeyColor: 104
setZoomFrame: 53
sharpness 54
showVideoInMiniWindow 66, 67
SMChromaSpace 134
SMColor 134
SMConvertRGBToYUV 104
SMConvertYUVToRGB 104
smNum 54
start: 74
statusAudio 113
statusTV 113
statusVT 114
stop: 74
storeProg:withFrequency:system:
andNorm: 114
storeProg:withFrequency:system:
norm:andName: 114
suspendClipping: 74
System 135,138, 139, 140
system 54

systemForCountry: 122

T

tableName 119
tableNamesForCountry: 122
Television norm 138
Television system 135
text 99
treble 61
Tuner Return Values 139
Tuner Type 138
tunerType 115

U

unregisterVideoView: 67
updateDisplayData: 91
usesKeying 104

V

verticalOffset 54
Video Input 135
Video-Out Input 136
videoInput 54
videoOutInput 57
videoOutSystem 57
videoOutXofs 57
videoOutYofs 58
videoViews 67
volume 61

W

write: 91, 99
writeFLM: withMode: 100
writeImage: 54, 58
writeProgs: 115
writeText 100
writeTIFF: 92
writeTIFF: usingCompression: 92
writeTIFF: usingCompression: andFactor:92

Y

YUV Color Model144
YUV Modes 137
yuvKeyColor 104