

ACTIMAGINE

VX Middleware for NINTENDO DS

Quick start guide
version 1.5.0.0 (September 5th 2006)

Introduction

The VX Middleware for NINTENDO DS is divided into different components:

- The VX Video Codec for Windows.
- The AviToDs conversion tool.
- Video resources.
- Three sample codes.
- The libVX libraries and prototype files.

Each of these components will be described below.

History

Version 1.5.0.0 (September 5th 2006)

- . Modifications done in the three sample codes in function `VXDemo_StartSoundStream`. The process of moving unstreamed audio data to the beginning of the audio buffer before restart has been removed : it was too slow under some conditions. Audio data is now simply dumped.

- . Added description about TCM usage.

Version 1.4.0.1 (March 20th 2006)

- . Logo data to be displayed is modified and added (Refer to `libVX/docs/README/AboutLogo.pdf` for the display condition). Logo data is available in `libVX/data/logo/`.

Version 1.4.0.0 (February 15th 2006)

- . Modification of AviToDs tool to avoid crash issue caused by problematic Avi files, now outputs the following message : *"Error: non standard or corrupted AVI file, please check file format."*.
- . `-debugsound <ouputfile.wav>` option added to AviToDs tool to ouput a wav file for previewing the compressed sound without the need of a NINTENDO DS development kit.

- . Modifications done in the three sample codes :
 - . You can pause/resume video playback by using START button.
 - . You can stop video playback by using B button.
 - . You can restart video playback by using A button.
 - . You can select loop disable/enable by using SELECT button.
 - . LCD power is off/on when DS cover is closed/opened.
 - . Use of *#pragma unused* instead of dummy affectation in callbacks to avoid compiler warnings.
- . Modification of html documentation concerning VX_Malloc/VX_Free functions : now describes where these functions are called internally in libVX library.

Version 1.3.0.2 (December 05th 2005)

- . Note added in this documentation explaining in detail the use of **Keyframe Interval** parameter in VX Video Codec for Windows.

Version 1.3.0.2 (November 28th 2005)

- . Typo error in vx.html removed, was displaying *VX_CloseMovie.htm'* in *libVX Library->Function List* section instead of *VX_CloseMovie*.
- . *VX_UnpackFrameImage* and *VX_BlitFrameImage* functions have a more detailed description in the html documentation.
- . *VX_SkipFrameImage* function has been put back on: hiding its functionality in libVX was causing a delay of several frames between video and audio. A detailed description is provided in the html documentation to better understand its use.
- . Sample codes modified to use the *VX_SkipFrameImage* function.
- . Wrong use of Sub 2D graphic engine functions in the sample code *dualScreen* fixed: the first blitted frame was never displayed. Functions modified in the sample are *NitroMain*, *VXDemo_Flip* and *VXDemo_GetSubBackBuffer*.
- . Redundant *nbFrameBuffered* variable increment removed in sample code *dualScreen*: was causing a skip instead of a blit of the first frame.

Version 1.3.0.1 (October 31th 2005)

- . Added logo bit map file to the package.
 A NOTICE regarding the logo which should be displayed for a certain period of time, has
 been added on Readme-libVX-1_3_0_1.txt.

Version 1.3.0.0 (October 21st 2005)

- . ActimagineAviToDS program renamed AviToDS.
- . VX Windows Codec renamed VX Video Codec
- . VX_SkipFrameImage function removed: its functionality is now hidden into the libVX API.
- . Modifications in the sample codes:
 - . Unused function prototype declarations removed.
 - . Playback speed can be controlled by the video framerate with a periodic alarm (case when a video does not contain any audio).
 - . VX_SkipFrameImage calls removed.
- . *VX Core* now renamed *libVX*, directory and file structure have been changed.
- . Functions and types reference moved from this document to html format.

Version 1.2.0.0 (August 3rd 2005)

- .New Vx Windows codec version 1,2,0,0:
 - .Fixed a bug in the bitstream format which could cause a crash in rare cases, video bitstream format is no longer compatible with the old VX Core libraries.
 - .Quality of the compressed videos slightly improved.
 - .Renamed *Actimagine Windows Codec For Nintendo DS*.
- .VX Core libraries modified to take into account the new video bitstream format.
- . Fixed rare audio compression distortion. Quality slightly improved.

Version 1.1.0.0 (July 27th 2005)

- .*VX_UnpackFrameSound* **renamed** *VX_GetFrameNbAudioPacket* in the SDK.
- .**Added a pitch parameter** to the function *VX_BlitFrameImage*. See the following documentation for more details. Set outFrameBufferPitch to 256 for fullscreen videos.
- .New Vx Windows codec version 1,1,1,1:
 - . Fixed bug in codec interface (slider).
 - . Fixed minor bug in codec core (quality slightly improved).
- . Fixed a bug which could cause an ARM9 exception or an assertion error.
- . Now detects sound buffer underflow in sample codes: displays "Sound underflow !!!!" in debug console.
- . macro *SND_CHANNEL_LOOP_REPAET* replaced by *SND_CHANNEL_LOOP_REPEAT* in the *SND_SetupChannelPcm* call of the sample codes.

- . New sample movies provided by NINTENDO.
- . Improved documentation of ActimagineAviToDs.
- . Added stereo sound support in the SDK:
 - . *VX_GetNbAudioTrack* function added.
 - . *VX_SkipFrameSoundOnePacket* function added.
 - . ActimagineAviToDs tool now accepts stereo PCM sound data.
- . To integrate stereo sound, binary format of .vx files has changed: you must **reconvert your resources** with the ActimagineAviToDs conversion tool.
- . New sample *VX_Sample_From_File_Stereo* added.
- . Every *NNS_** function replaced by *SND_** equivalent in the sample codes.

Version 1.0.0.3 (June 1st 2005)

- . New Vx Windows codec version 1,1,1,0:
This codec fixes compatibility problems with external Windows applications as described below.
There is a new configuration interface:
 - . Enhanced Chominance checkbox is removed (always on now).
 - . Distortion parameter replaced by a more understandable Quality parameter.
 See following documentations for more details.
- . Binary format of .vx files no longer compatible with earlier versions of the SDK, you must reconvert your resources with the ActimagineAviToDs conversion tool.
- . *FS_Init()* is now called with DMA number 1 as parameter in the samples (DMA 0 is no longer usable for filesystem functions in the latest NITRO SDK).
- . Memory leaks removed in the *VX_CloseMovie* function.
- . Upper/lower case distinction of the video fourcc removed in the ActimagineAviToDS conversion tool.
- . Modifications of the Vx Windows codec have been done to circumvent the non standard way TMPGEnc application uses windows codecs.
- . Sound glitches due to heavy decode processing time fixed.
- . *VX_ReadFrame* function optimized (could take several milliseconds to execute, now takes only 200 microseconds), file reads done asynchronously when playback is done from an FS_File.
- . *VX_SkipFrameImage* function added : useful when you have a long sequence of hard to unpack frames, call it alternatively with *VX_BlitFrameImage* to save time for the

unpack of the next frame. See the following documentation and the samples for more details.

- . The maximum number of frames which can be unpacked without been blitted or skipped is now user definable. Prototype changes of functions *VX_OpenMovieFromMemory* and *VX_OpenMovieFromFile* made to allow this feature. See the following documentation and the samples for more details.

- . Phantom sound problem fixed: modifications in the sound compression routine of ActimaginaireAviToDS program have been done. Remove old .cbk files if any.

- . AfterEffects problem with Vx Windows Codec: modifications of the Vx Codec has been done to circumvent an AfterEffects bug. The codec now works with AfterEffects 5.5 and 6.5.

Due to AfterEffects non standard way to use Windows codecs, batch processing is not yet possible. VirtualDub batch processing works perfectly though.

Version 1.0.0.2 (April 21st 2005)

- . Problem of playback looping when streaming from memory fixed.
- . New function *VX_GetIFrameInfo* added in the SDK.

Version 1.0.0.1 (March 14th 2005)

- . new video codec: much faster compression, faster playback.
- . new audio codec: selectable quality, faster playback.
- . samples reworked for optimized playback:
 - . asynchronous video buffer flipping.
 - . decompressed frames buffering: smoother playback of difficult frames, no more sound glitches.
- . new SDK functions :
 - . *VX_GetNbIFrame*.
 - . *VX_JumpToIframe*.

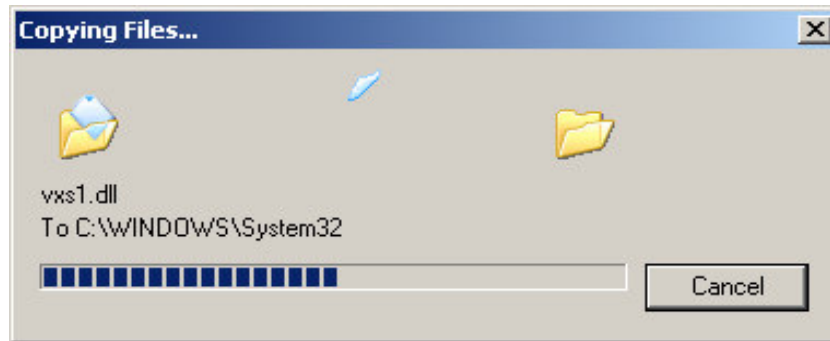
Version 1.0.0.0 (February 8th 2005)

- . first release.

VX Video Codec for Windows

Codec installation

- 1- Go to the *libVX/tools/bin* directory.
- 2- Right click on **vxs1.inf** and select Install. This installs the VX codec.



Video editing tool

We recommend using the freeware VirtualDub as a video tool to compress your videos. You can download it at <http://www.virtualdub.org>.

You can also use “off the shelf” products like Adobe Premiere® or Adobe After Effects®.

Video Preparation and Processing

Video preparation and processing are very important. They will contribute to 50% of the final video quality, the remaining 50% coming from the video codec itself.

Rule #1: Use the best video source you can get

Rule #2: Resolution of the video sources should be big, at least 640x480

Many processes (like contrast/gamma) will benefit from the size of the video to convey less errors (after the final downsize), because they are applied to more pixels beforehand.

Rule #3: Delay the final downsizing to DS resolution as late as possible in your processing pipeline

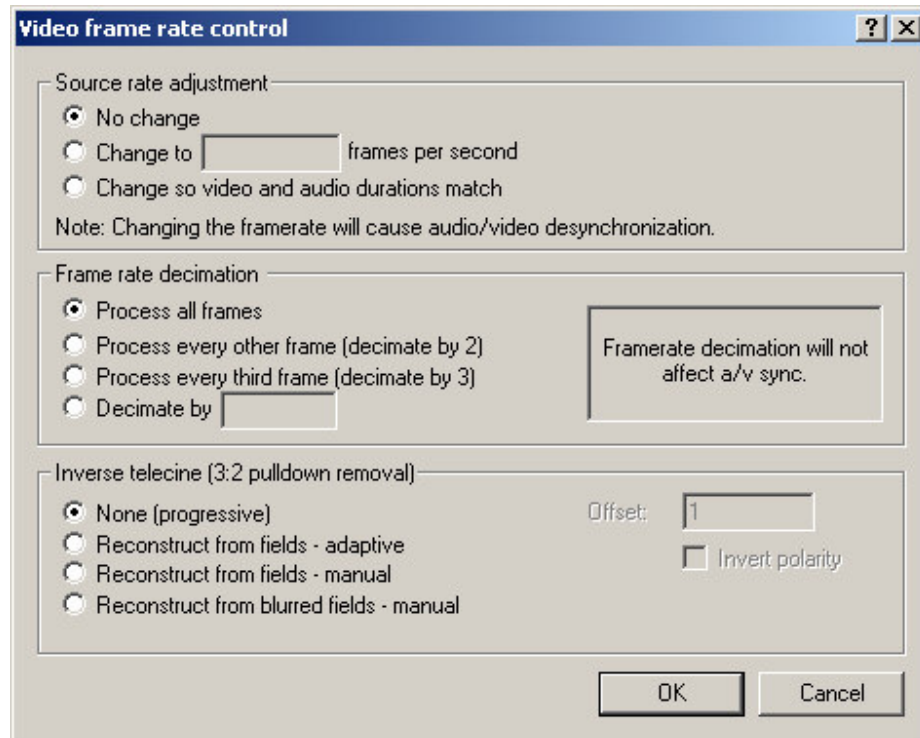
Rule #4: Source video should be uncompressed or not compressed much

Rule #5: be careful of the source video frame rate

Typical videos captured from TV are 30 fps. However, the original frame rate (cinema) is often 24 fps. In that case you should convert the movie back to 24 fps with an “inverse telecine” method to ensure better quality. This kind of method is available in VirtualDub in the *Video/Frame rate* menu.

Rule #6: When lowering frame rate, only use “decimation” methods

See the *Video/Frame rate* menu in VirtualDub.



Rule #7: in the final video, keep as much as possible the same aspect ratios as the source video (in general 10% difference is ok)

Your source videos will probably have an aspect ratio of 4/3 (TV), 16/9 (cinema) or 2.35 (cinema also). DS aspect ratio is 4/3 (256/192). To convert from source aspect ratio to DS aspect ratio, you will have to crop your video (remove a part of the image) or add black bars.

Rule #8: downsize using a “Precise bicubic” (or better) method

You can choose the target resolution of the videos with the following restrictions:

- Final resolution must not exceed 256x192 for DS.
- Width and height of the target resolution must be multiples of 16.

Again, do not forget to preserve the original aspect ratio.

Rule #9: when storing intermediate movies, use a lossless video codec

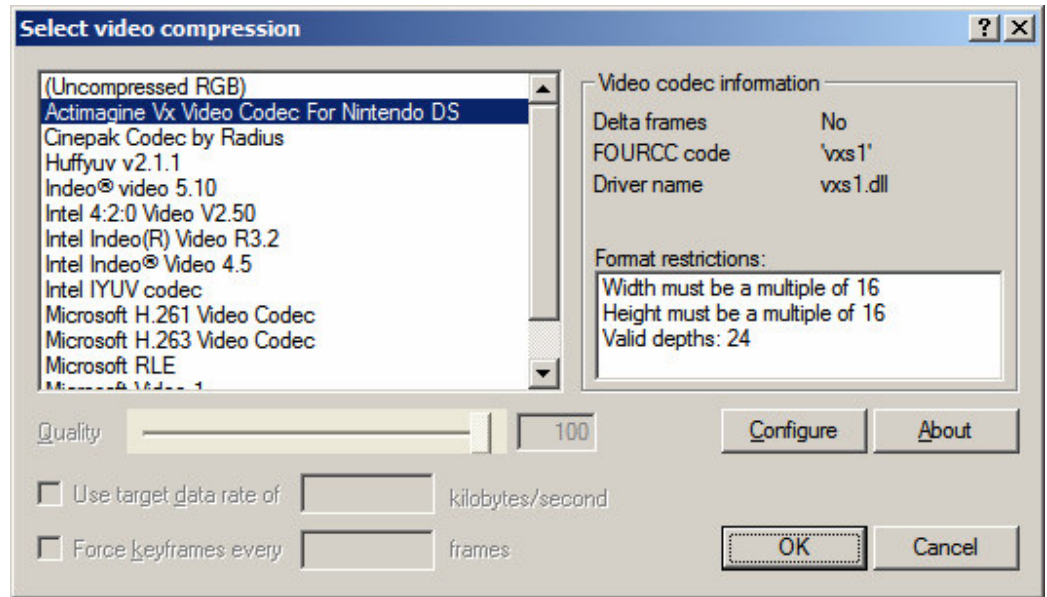
We recommend using Huffiyuv for storing intermediate files.

See <http://neuron2.net/www.math.berkeley.edu/benrg/huffyuv.html>.

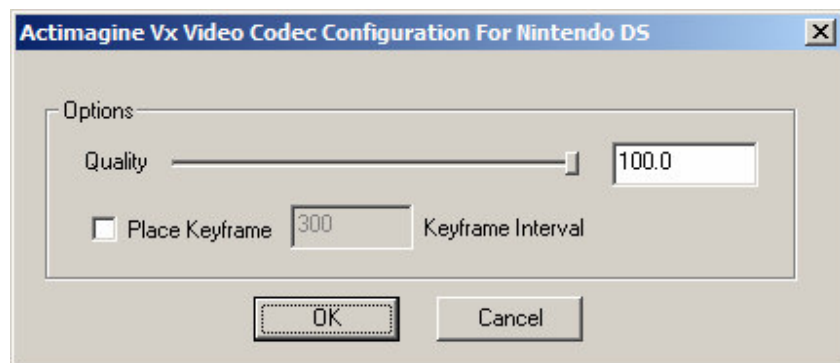
Video Compression

Once the video is prepared, compressing it is very straightforward.

1- In VirtualDub, open the *Video* → *Compression* menu and choose **Actimagine VX Video Codec For NINTENDO DS**



3- Click on the *Configure* button



3- Set the parameter values:

- . **Quality** manages image quality, artifacts and data-rate, valid values are between 0 and 100 inclusive.

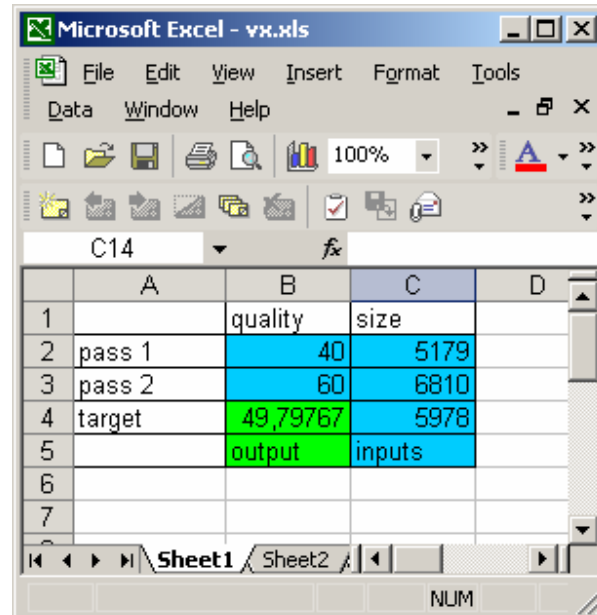
High Quality gives high quality and high data-rate.

Low Quality gives low quality and low data-rate.

- . **Place keyframe** places keyframes automatically every **Keyframe Interval** frames approximately. If it is not selected, the video will have only one keyframe (the first frame). If you need FastForward/Rewind functionality, a maximum interval of 300 frames is recommended (approximately one keyframe every 12 seconds at 24 fps). Valid values of **Keyframe Interval** parameter are between 10 and 500 inclusive.

You can use the excel file **vx.xls** (located in directory **libVX/tools/etc**) which helps you to choose the correct Quality parameter to achieve the desired target size. Just compress 2 times your video with different qualities, then fill in the blue cells in the excel file with

the two qualities and size values to obtain the target quality needed to have the desired file size (it uses linear interpolation).



The screenshot shows a Microsoft Excel window titled 'Microsoft Excel - vx.xls'. The spreadsheet has columns A, B, and C. Row 1 contains 'quality' in B1 and 'size' in C1. Row 2 contains 'pass 1' in A2, '40' in B2, and '5179' in C2. Row 3 contains 'pass 2' in A3, '60' in B3, and '6810' in C3. Row 4 contains 'target' in A4, '49,79767' in B4, and '5978' in C4. Row 5 contains 'output' in B5 and 'inputs' in C5. The status bar at the bottom shows 'NUM'.

	A	B	C	D
1		quality	size	
2	pass 1	40	5179	
3	pass 2	60	6810	
4	target	49,79767	5978	
5		output	inputs	
6				
7				

4- Close the dialog boxes and launch the compression process.

Note:

The **Keyframe Interval** parameter of the VX Video Codec for Windows is just an indication to know approximately when it should place key-frames. This parameter is used by the codec to look for a scene-cut around the 'theoretically' perfect position and place the key-frame there because putting it elsewhere would result in unpleasant visual effects.

If you want precise control of the placement of key-frames (for example to be able to play a movie from a given set of points), you should cut your source movies into parts. You can do this with VirtualDub using 'Edit -> Set Selection start/end' and save each part using 'File -> Save as AVI'.

Each part should begin at a point where the user wants to begin the playback.

Then you compress each part with the VX Video Codec for Windows.

This way, once compressed, you will be sure that there is a key-frame at all important points (at the beginning of each part) and there is no need to add additional key-frames (you can leave **Place Keyframe** unchecked in the codec options).

If you want to have a single movie file, you can merge the compressed movies.

You can do this with VirtualDub using 'File -> Open', then 'File -> Append' (to add all parts), set 'Video -> Direct Stream Copy' (to keep the video compressed) and finally 'File -> Save as AVI'.

There is one limitation to merge movie files : you must use the same compression **Quality** parameter for all parts to be merged.

Sound Preparation

- Audio format in the final AVI file must be **uncompressed 16 bit mono or stereo**.
- Audio should be **normalized** at 100% so it can use the whole dynamic range of the DS.

AviToDs conversion program

AviToDs.exe is a console program used to convert an input avi file containing VX compressed video frames to a binary file suitable for the libVX libraries.
It is located in the directory *libVX/tools/bin*.
This tool compresses the sound during conversion.

Syntax: AviToDs <commands>

<commands> can be:

- in <inputfile.avi>
- out <outputfile.vx>
- codebook: always rebuild the audio codebook
- nocodebook : never rebuild the audio codebook
- quality: audio quality factor, default value is 128, value must be positive
- debugsound <outputfile.wav>

A typically good trade-off between sound quality and data-rate is to use 32.768Hz sound and leave the default *quality* parameter (128). This should give a sound data-rate around 32kbit/s.

By comparison, an average video data-rate for a full-screen (256x192) high quality (30fps) video can typically be around 400kbit/s (this depends highly on the video).

You should keep in mind the video data-rate when tuning audio quality as audio generally doesn't use much of the available space. On the other hand, low data-rate video can make audio data-rate more significant.

If you are not satisfied with the sound quality or data-rate, you can adjust both sampling frequency and the *quality* parameter.

The *quality* parameter ranges from 0 to 1000. Resulting data-rates then ranges from 0.6 to 1.25 bit/sample. Low *quality* gives low data-rates; high *quality* gives high data-rates. Good trade-offs between quality and data-rate can be found around the default value (128).

If the sound quality is not good enough at 16kHz with a high *quality* parameter, we recommend increasing sampling frequency at 32kHz (but it will also increase data-rate accordingly). In the same way, if data-rate is too high, even with a low *quality* parameter, we recommend decreasing sampling frequency.

A codebook (.cbk) file is an "audio analysis" file generated the first time a video is converted. If the same file is converted another time, the codebook is not re-calculated (which saves time).

The *codebook* option forces the re-calculation of the codebook, even if the video file has not changed.

The *nocodebook* option is useful when you convert a video file which changes many times with always the same audio inside (for example you test different video compression parameters). If you use the *nocodebook* option, the codebook will not be recalculated every time, it will be done only at the first time, which saves time.

The *debugsound* option allows you to output the resulting compressed sound in WAV format : useful when you want to have a quick preview of the quality of the compressed sound without the need of a NINTENDO DS development kit.

VX videos

You will find in the *libVX/build/demos/libVX/data* directory the resources used to make the VX movies for NINTENDO DS:

- *scr1_audio_mono.avi*, *scr1_audio_stereo.avi*, *scrU_audio_mono.avi*, *scrD.avi* : the avi sources (already compressed using the VX Video Codec for Windows).
- *scr1_audio_mono.vx*, *scr1_audio_stereo.vx*, *scrU_audio_mono.vx*, *scrD.vx* : the resources obtained by using the AviToDs program.

These resources are used in the sample code described below.

Sample codes

You will find in the *libVX/build/demos/libVX* directory three different sample codes:

- *fromFile* is a basic sample code showing how to use the libVX libraries to play a simple video file with mono sound.
- *dualScreen* is a basic sample code showing how to use the libVX libraries to play two different video files (with mono sound) on the upper and lower screens.
- *fromFileStereo* is a basic sample code showing how to use the libVX libraries to play a simple video file with stereo sound.

These three samples are well documented. You are invited to have a look at them to learn how to use the libVX libraries.

libVX libraries and prototype files

Prototype files:

- '*vx.h*' in directory *libVX/include/nitro*.
- '*libVX.h*' in directory *libVX/include/nitro/vx*.

Library files:

- three versions of file '*libVX.a*' :
 - in directory *libVX/lib/ARM9-TS/Debug* for Debug builds.
 - in directory *libVX/lib/ARM9-TS/Release* for Release builds.
 - in directory *libVX/lib/ARM9-TS/Rom* for Rom builds.

Using the library:

Include the file 'vx.h' in your source code and link with one the three versions of the 'libVX.a' library.

For further details, see the html documentation of libVX and have a look at the sample codes.

About TCM usage

libVX utilizes these sizes of TCM(Tightly-Coupled Memory) when playing back movies.

- 26784 bytes of ITCM
- 4064 bytes of DTCM

ITCM portion of libVX is set as autoload , it cannot be overlaid.

You can do whatever you want with the ITCM portion allocated by libVX when not playing videos as long as you save it before erasing it and restore it before reusing libVX.

Here is an exhaustive list of libVX functions who are calling ITCM located code :

- 'VX_UnpackFrameImage' function calls ITCM located 'VxUnpack' internal function.
- 'VX_BlitFrameImage' function calls ITCM located 'YuvToArgb' internal function.
- 'VX_BlitFrameSoundOnePacket' function calls ITCM located 'SxUnpack' internal function