



Programming Software

SDK – Software Development Kit

Version 2.7

User Guide

SDK – Software Development Kit User Guide

Introduction

Overview

Installing the SDK

Debugging

Appendix A – DVSIInfo Program

Appendix B – DVSConf Program

Appendix C – svram Program

Index

1

2

3

4

A

B

C

I

User Guide Version 2.0 for SDK – Software Development Kit Version 2.7

Copyright © 2006 by DVS Digital Video Systems GmbH, Hanover. All rights reserved.

The manuals as well as the soft- and/or hardware described here and all their constituent parts are protected by copyright. Without the express permission of DVS Digital Video Systems GmbH any form of use which goes beyond the narrow bounds prescribed by copyright legislation is prohibited and liable to prosecution.

This particularly applies to duplication, copying, translation, processing, evaluation, publishing, and storing and/or processing in an electronic system.

Specifications and data may change without notice. We offer no guarantee that this documentation is correct and/or complete. In no event shall DVS Digital Video Systems GmbH be liable for any damages whatsoever (including without limitation any special, indirect, or consequential damages, and damages resulting from loss of use, data, or profits, or business interruption) arising out of the use of or inability to use the hardware, software and/or manual materials.

Those parts of this documentation that describe optional software or hardware features usually contain a corresponding note. Anyway, a lack of this note does not mean any commitment from DVS Digital Video Systems GmbH.

Adobe and Acrobat Reader are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States and/or other countries. CLIPSTER is a registered trademark of DVS Digital Video Systems GmbH. Linux is a registered trademark of Linus Torvalds. Microsoft, MS DOS, Windows, and Windows NT are registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. UNIX is a registered trademark in the United States and other countries, licensed exclusively through X/Open Company, Ltd. Red Hat is either a registered trademark or trademark of Red Hat, Inc. in the United States and other countries.

Any other product names mentioned in this documentation may be trademarks or registered trademarks of their respective owners and as such are subject to the usual statutory provisions.



Headquarters:

DVS Digital Video Systems GmbH
Krepenstr. 8
30165 Hannover
GERMANY

Phone: +49-511-67807-0
Fax: +49-511-630070
E-mail: info@dvs.de
Internet: <http://www.dvs.de>

Support:

Phone: +49-511-67807-25
Fax: +49-511-67807-31
E-mail: support@dvs.de

For the Americas:**U.S. Headquarters:**

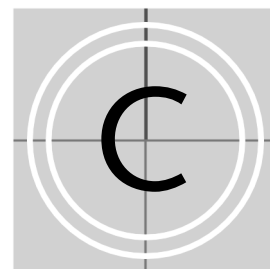
DVS Digital Video, Inc.
300 East Magnolia Boulevard, Suite 102
Burbank, CA 91502
USA

Phone: +1-818-846-3600
Fax: +1-818-846-3648
E-mail: info@dvsus.com
Internet: <http://www.dvsus.com>

Support:

E-mail: support@dvsus.com

Contents



1	Introduction	1-1
1.1	Overview	1-2
1.2	Target Group	1-2
1.3	Conventions Used in this User Guide	1-3
1.4	Requirements	1-5
2	Overview	2-1
2.1	The SDK – Software Development Kit	2-2
2.2	Overview of Folders and Files	2-4
2.2.1	Folder 'development'	2-4
2.2.2	Folder 'documentation'	2-6
2.2.3	Folder '<operating system>'	2-6
2.3	The DVS PCI Video Board Driver	2-10
2.3.1	Debug Driver vs. Release Driver	2-10
2.3.2	The Driver under Linux	2-10
2.3.3	The Driver under Windows	2-13
2.3.4	Customized Video Rasters	2-13
2.4	The DVS Header Files	2-14
2.4.1	Video C Library ('dvs_clib.h')	2-14
2.4.2	Error Code Header ('dvs_errors.h')	2-15
2.4.3	FIFO API ('dvs_fifo.h')	2-15
2.4.4	File Converter C Library ('dvs_fm.h')	2-16
2.4.5	Setup Header ('dvs_setup.h')	2-17
2.4.6	Thread Header ('dvs_thread.h')	2-17
2.4.7	Version Header ('dvs_version.h')	2-17
2.5	The Example Projects	2-18
2.5.1	RAM-recorder Example Programs	2-18
2.5.2	FIFO API Example Programs	2-19

3	Installing the SDK	3-1
3.1	Installation under Linux	3-2
3.1.1	Installing the SDK	3-2
3.1.2	Creating Device Nodes	3-2
3.1.3	Post-compiling the Driver	3-3
3.1.4	Loading the Driver	3-4
3.2	Installation under Windows	3-5
3.2.1	Installing the SDK	3-5
3.2.2	Loading the Driver	3-5
3.3	Updating an Existing SDK	3-8
3.3.1	Updating the SDK	3-8
3.3.2	Updating the Driver	3-8
4	Debugging	4-1
4.1	How to Obtain Debug Information	4-1
4.2	How to Obtain DVS Information Logs	4-2
4.2.1	Gathering Diagnostic Information under Linux	4-2
4.2.2	Gathering Diagnostic Information under Windows	4-2
4.3	How to Obtain a Streamer Mode Record	4-4
A	Appendix A – DVSIInfo Program	A-1
A.1	DVSIInfo under UNIX/Linux	A-2
A.1.1	Basic Usage	A-2
A.1.2	Options of DVSIInfo	A-2
A.2	DVSIInfo under Windows	A-5
A.2.1	Basic Usage	A-5
A.2.2	The User Interface	A-5
B	Appendix B – DVSConf Program	B-1
B.1	Basic Usage	B-2
B.1.1	Starting the Program	B-2
B.1.2	Exiting the Program	B-2

B.2	The User Interface	B-4
B.2.1	The Tab 'Driver'	B-4
B.2.2	The Tab 'Settings'	B-8
B.2.3	The Tab 'Server'	B-9
B.2.4	The Tab 'Card'	B-11
C	Appendix C – svram Program	C-1
C.1	Using the svram Program	C-2
C.1.1	svram versus sv	C-2
C.1.2	Operation Modes	C-2
C.1.3	Obtaining Help	C-3
C.1.4	Setting the Environment Variable	C-3
C.2	Command Reference	C-4
I	Index	I-1

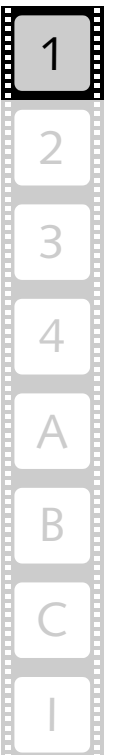
Introduction



This document provides information about the software development kit (SDK) by DVS. The SDK can be used together with the OEM products manufactured by DVS. It is a software package that – once installed properly – provides a complete development and runtime environment, including not only an SDK but helpful tools and drivers as well.

The DVS OEM products are designed for companies that develop their own digital video and audio I/O solutions. As PCI video boards they constitute the heart of your digital video computer system's hardware where they can be seamlessly integrated. The SDK can be used to build the software application which will access the PCI video board and control its features. It is delivered together with some tools for hardware setup and diagnostics, such as the DVSInfo program. Furthermore, there are PCI video board drivers included. To run the DVS OEM product properly a driver has to be loaded before accessing the PCI video board which can be done with the tools for the hardware setup. The video board driver then controls the board and thus the in- and output of video, audio and control signals.

The SDK by DVS is compatible among the DVS OEM products, meaning your code can be used with other DVS PCI video boards as well.



1.1 Overview

This user guide informs you about the general handling of the SDK as well as about the tools and drivers delivered with it.

The chapters in this user guide contain the following information:

Chapter 1	Begins with a short introduction to the SDK, followed by a note regarding the audience this manual is written for and an explanation of the conventions used in this manual. Furthermore, it lists the system requirements to run the DVS SDK.
Chapter 2	This chapter provides an overview of the SDK as well as about the tools and drivers delivered with it. All folders and files of the SDK are described.
Chapter 3	Describes the installation of the SDK and explains how to update it in case a new version is available.
Chapter 4	Explains how to debug an application built with the SDK.
Appendix A	Provides details about one of the tools delivered with the SDK, i.e. it explains the DVSTInfo program.
Appendix B	Provides details about one of the tools delivered with the SDK, i.e. it explains the DVSTConf program.
Appendix C	Provides details about one of the example programs delivered with the SDK, i.e. it describes several features of the svram program.
Index	This chapter facilitates the search for specific terms.

1.2 Target Group

To use this manual and the SDK you should have experience in software development and knowledge in the field of digital video/audio in general, including knowledge about the handling and the internal structure of a digital video system.

1.3 Conventions Used in this User Guide

The following typographical conventions will be used in this documentation:

- Texts preceded by this symbol describe activities that you must perform in the order indicated.
- Texts preceded by this symbol are parts of a list.



Texts preceded by this symbol are general notes intended to facilitate work and help avoid errors.



You must pay particular attention to text that follows this symbol to avoid errors.

" " Texts enclosed by quotation marks are references to other manuals, guides, chapters, or sections.

Tab/Menu

Standard text in italic and bold indicates either a tab name, a menu name or options in a menu list

BUTTON

Standard text in small caps and bold indicates push buttons

Item

Standard text in bold only stands for other labelled items of the user interface

Directory/File

Directory structure or file; when specifying paths, this document follows the Linux/UNIX conventions, i.e. a slash (/) will be used; under Windows it has to be replaced with a backslash (\)

Command

In the standard text flow a regular typeface of a command indicates commands, variables, or parameters; it may also indicate a file syntax or contents of a file; when used in conjunction with the command in **bold**, it stands for optional parameters

Command

Command, for example, at a prompt; only used when needed and then to indicate that this has to be typed in exactly as written

[Key]

An individual key or a key combination on a keyboard

Keyboard Short-cuts

To perform options or procedures with the keyboard often requires simultaneous pressing of two keys.

Example:

[Ctrl + F1]	If this is given, hold down the [Ctrl] key and press simultaneously the [F1] key.
-------------	---

Command Descriptions

In command syntax descriptions, a **bold** typeface indicates literal commands and parameters that have to be typed in as they are, whereas a `plain` typeface indicates a substitute that has to be replaced by a suitable expression (options, variables, defaults, obligatory parameters etc.).

<...>	variable; this term has to be replaced by an appropriate value
-------	--

Example:	svram goto <#frame> Replace <#frame> by a valid frame number and type e.g. <code>svram goto 20</code> .
----------	---

[...]	optional part; you don't have to specify it
-------	---

Example:	svram goto <#frame> [{ field1 , field2 , frame }] You have to specify a value for <#frame>, but may leave out the rest.
----------	---

<xxx>=y	default; if you don't specify <xxx>, it will be set to the default value y
---------	--

Example:	svram inputport { analog , sdi , info }=sdi If not specified otherwise the input port will be set to <code>sdi</code> , i.e. to select the SDI you simply have to type <code>svram inputport</code> .
----------	---

{...}	selection; select one of the given alternatives separated by comma(s)
-------	---

Example:	svram speed <#speed> [{ once , loop , shuttle }] Type e.g. <code>svram speed 0.5 shuttle</code> .
----------	---

1.4 Requirements

The SDK is available for the following operating systems:

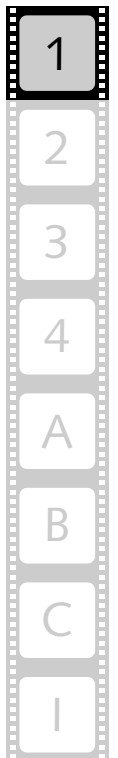
- Linux Red Hat
- Windows 2000
- Windows XP

Linux

The following minimum environment will be needed to ensure a proper operation of the SDK on a Linux system:

- GLIBC 2.3
- gcc 3.2
- stable release kernel

The SDK was developed for RHEL 3 and 4. Other distributions should work as well. In some cases slight modifications may be necessary.



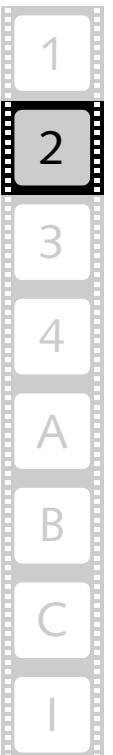
Overview



This chapter provides you with some general information about the software development kit (SDK) by DVS. It contains an overview of the SDK as well as of the tools and drivers delivered with it. Additionally, all folders/files of the SDK are detailed.



A documentation of the individual defines and functions provided by the SDK can be found in the separate “SDK – Software Development Kit” reference guide.



2.1 The SDK – Software Development Kit

The SDK by DVS provides a complete development and runtime environment to build your own video and audio I/O software solutions with the DVS PCI video board (OEM product) as a basis. During the development you will implement appropriate commands into your application to access the various features of the PCI video board. Thus you can realize your own editing or storage solutions when dealing with digital video and audio.

The SDK comes together with tools for a basic hardware setup and diagnostics, such as the DVSConf program, as well as PCI video board drivers for a control of the video board.

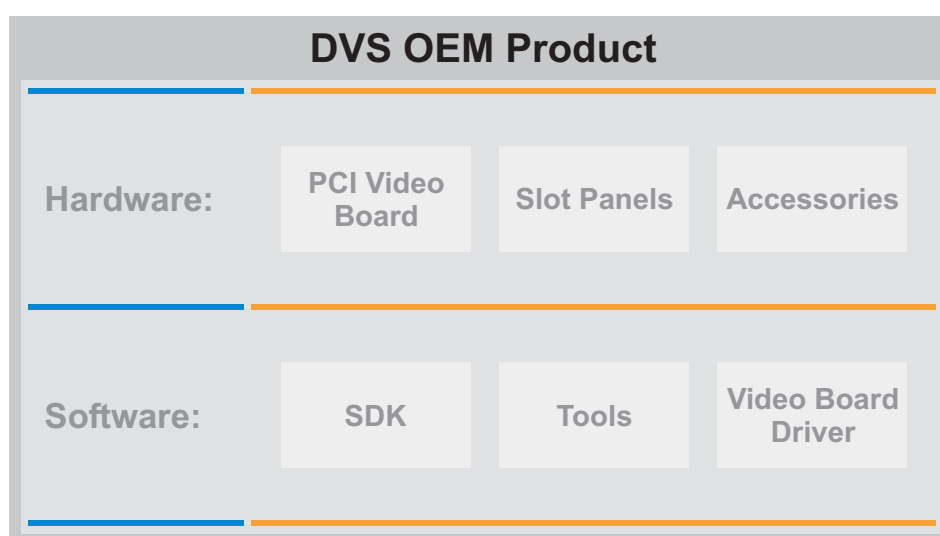


Figure 2-1: Components overview



Further information about the hardware side of the DVS OEM product can be found in the installation guide of the PCI video board.

In its parts the software package of the SDK by DVS consists of the following components:

SDK	Tools	Video Board Drivers ¹
<ul style="list-style-type: none"> – Header files – Binaries – Examples 	<ul style="list-style-type: none"> – DVSConf² – DVSInfo 	<ul style="list-style-type: none"> – Video board driver – Debug driver

1) Must be post-compiled under Linux.

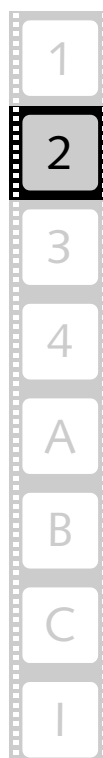
2) Windows only.

A short description of the folders and files included in the SDK can be found in section “Overview of Folders and Files” on page 2-4, whereas the tools are described in chapter “Appendix A – DVSInfo Program” on

page A-1 and chapter “Appendix B – DVSConf Program” on page B-1. Last but not least further information about the drivers can be found in section “The DVS PCI Video Board Driver” on page 2-10.

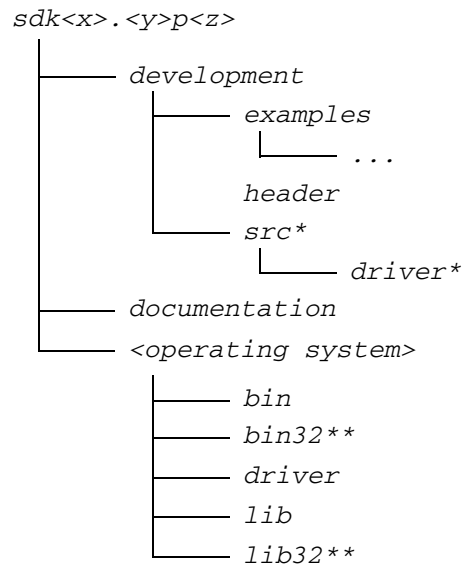


A documentation of the individual defines and functions provided by the SDK can be found in the separate “SDK – Software Development Kit” reference guide.



2.2 Overview of Folders and Files

This section provides a short description of the folders and files that you can find on your computer's hard disk after the installation of the SDK (see chapter "Installing the SDK" on page 3-1). Once the SDK is extracted, the following directory structure will be created:



* Available under Linux only.
 ** Available under 64-bit SDKs.

Figure 2-2: SDK directory structure

The main folder of the SDK *sdk<x>.<y>p<z>* contains 'readme' files with up-to-date information about the SDK release and – in the subfolders – all the files that you need to start a development with the SDK. The numbers in the name of the main folder indicate the respective SDK version, i.e. the major (<x>), minor (<y>) and patch (<z>) version levels.

2.2.1 Folder 'development'

In the folder *development* you can find files that you will need to develop software with the SDK. The contents of the development folders are the same for all operating systems. It contains the following:

- Universally applicable files (such as C headers and C sources) that are needed when developing under any operating system.
- Project files (**.dsp*), workspace files (**.dsw*), resource scripts (**.rc*), and icons (**.ico*) that are only needed when developing under Windows.
- *makefile* files that are only needed when developing under Linux.



All plain text files (like C headers, C sources, makefiles, etc.) are formatted as Windows ASCII files in the SDK for Windows, and formatted as UNIX ASCII files in the SDK for Linux. The main difference are the end-of-line characters, being <CR><LF> under Windows and <LF> under UNIX/Linux.

Subfolder 'examples'

The *examples* directory contains example projects that can be analyzed to get a better understanding of how to program with the SDK:

- The Windows workspace file *sdk.dsw* holds all example projects that can be found in the subfolders of the folder *examples* (e.g. *counter*, *demo*, *dmarect*, etc.).
- Under Linux the file *makefile* may be used to build all example programs at once.

The subfolders sorted below the *examples* folder contain the sources for the example projects, each in one folder: They contain the following files:

- C sources
- Resources such as icons, sounds, etc.
- Additional header files that are needed for the respective project in particular
- Project files for Windows (**.dsw*, **.dsp*, **.rc*) and Linux (*makefile*) that are needed to compile the project

The functions and purposes of the individual example projects are described in section “The Example Projects” on page 2-18.

Subfolder 'header'

The *header* folder contains the DVS header files. Further information about them can be found in section “The DVS Header Files” on page 2-14.

Subfolder 'src/driver'

These folders are available in SDKs for Linux only. They provide in the *driver* folder the necessary project files to finish (post-compile) the DVS PCI video board driver for your respective Linux kernel interface



(see section “The Driver under Linux” on page 2-10 as well as section “Post-compiling the Driver” on page 3-3):

File	Description
<i>Makefile</i>	Post-compile step makefile.
<i>linux.c</i>	Linux kernel interface.
<i>ps_common.h</i>	Common header.
<i>ps_debug.h</i>	Debug header.

When post-compiling the driver with these files, the pre-compiled binary files with the extension **.pre* in the folder *sdk<x>.<y>p<z>/linux/driver* (see section “Subfolder ‘driver’ under Linux” on page 2-7) will be compiled and finished to the final object files of the driver.

2.2.2 Folder ‘documentation’

The folder *documentation* contains the SDK user guide you are reading right now, the SDK reference guide providing details about defines, functions and commands that can be used under the SDK, and hardware installation guides for the different PCI video boards that can be delivered by DVS. Usually, the documentation is available in the PDF file format. To open these files you will need the Adobe Acrobat Reader which can be downloaded free of charge from the web site of Adobe (www.adobe.com/prodindex/acrobat/readstep.html).

2.2.3 Folder ‘<operating system>’

The *<operating system>* folder is named according to the operating system the SDK is developed for. For example, under Windows it will either carry the name *Win32* or *Win64* (depending on the bit architecture) whereas under Linux it would be *linux*. It contains binaries, drivers and libraries needed for a development under the SDK.

Subfolder ‘bin’

The subfolder *bin* contains programs and, in case of Windows, dynamic link libraries (DLLs) that are needed to operate the video device. The programs can be divided into two categories:

1. Tools for configuration, operation, and debugging. They are described in chapter “Appendix A – DVSIInfo Program” on page A-1 and chapter “Appendix B – DVSConf Program” on page B-1.

2. All other executable files are the example programs delivered with the SDK. They were compiled from the projects in the folder `sdk<x>.<y>p<z>/development/examples`. The functions and purposes of the examples are described in section “The Example Projects” on page 2-18.

Under an SDK for 64-bit operating systems two *bin* folders will be available: The folder *bin* contains the 64-bit versions of the above mentioned files, whereas the folder *bin32* contains the same files in 32 bit (in case you want to develop 32-bit applications on a 64-bit operating system).

Subfolder 'driver' under Linux

Under Linux the folder *driver* contains helpful script files, the PCI video board driver and additional files that are needed by the driver.



Prior to using the DVS PCI video board driver under Linux it has to be post-compiled (see section “Installation under Linux” on page 3-2).

For further details about the driver see section “The DVS PCI Video Board Driver” on page 2-10.

File	Description
<i>driver_*</i>	All files that start with this word are script files to make a handling of the driver under Linux easier. They are used to create (post-compile), load or unload the driver.
<i>dvspartner</i>	This file is the driver for Linux. It can be found either with the extension <i>*.pre</i> which is the pre-compiled version of the driver to be finished with the post-compile step, or <i>*.ko</i> which is the final usable driver.
<i>dvspartner</i>	This file is the debug driver for Linux. It can be found either with the extension <i>*.pre</i> which is the pre-compiled version of the driver to be finished with the post-compile step, or <i>*.ko</i> which is the final usable driver.
<i>mkdev</i>	This file is used to create the driver device nodes during the installation of the SDK.

1

2

3

4

A

B

C

I

File	Description
<i>*.pld</i>	The files with this extension contain the micro-code for the video hardware (pld = program-mable logic device). By default, they must reside in the same folder as the driver. It is possible to place them in another path, but then you have to specify that path in the script for the loading of the driver (<i>driver_load</i>).
<i>*.ref</i>	The files with this extension (ref = raster editor file) store video raster definitions where <i>default.ref</i> holds the standard DVS video rasters and <i>userdef.ref</i> is used to store customized video rasters.

Subfolder 'driver' under Windows

Under Windows the folder *driver* contains **.sys* files which are the driver object files (for different versions of the driver and of Windows). Most other files that can be found in this folder are additional files needed by the driver.



For further details about the driver see section “The DVS PCI Video Board Driver” on page 2-10.

File	Description
<i>clipbord.sys</i>	This file is the driver for Windows NT.
<i>clipdbg.sys</i>	This file is the debug driver for Windows NT.
<i>dvswin2k.sys</i>	This file is the driver for Windows 2000/XP.
<i>dvs2kdbg.sys</i>	This file is the debug driver for Windows 2000/XP.
<i>dvs.inf</i>	This file is the Windows 2000/XP driver installation file.
<i>*.pld</i>	The files with this extension (pld = programmable logic device) contain the microcode for the video hardware. They must reside in the same folder as the driver.
<i>*.ref</i>	The files with this extension (ref = raster editor file) store video raster definitions where <i>default.ref</i> holds the standard DVS video rasters and <i>userdef.ref</i> is used to store customized video rasters.

Subfolder 'lib'

The subfolder *lib* contains the library files that are needed for the linking process. Under an SDK for 64-bit operating systems two *lib* folders will be available: The folder *lib* contains the 64-bit versions of the above mentioned files, whereas the folder *lib32* contains the same files in 32 bit (in case you want to develop 32-bit applications on a 64-bit operating system). For further information when to use the respective library files consult section section "The DVS Header Files" on page 2-14.

Under Linux all DVS libraries are linked statically. The naming convention is *lib<name>.a*, where *<name>* specifies the library name.

Under Windows the naming convention is *<name>.lib*, where *<name>* specifies the library name.



2.3 The DVS PCI Video Board Driver

This section contains some details about the DVS PCI video board driver.

The DVS PCI video board and its driver are the central components of your video system because they provide its functionality. Without them it would not be able to display any video signals nor would the developed software components be operational. The PCI video board driver controls the video board and thus the in- and output of video signals.

At first some general information will be given, followed by notes about the driver under the different operating systems supported by the SDK. The section will be concluded with an explanation about customized video raster files and how to create them.

2.3.1 Debug Driver vs. Release Driver

The PCI video board driver exists in two versions, the debug driver and the release driver:

- The release driver is to be used for normal operation of the video device.
- The debug driver is significantly larger than the release driver because it contains a lot more plain-text error messages.

If one of the functions returns an error message from the driver, then normally an error message is printed to the drivers internal message buffer. This error message can be read out, for example, with the command `svram debugprint` or by using the DVS Video Card Info program. Further information about how to obtain debug information can be found in chapter “Debugging” on page 4-1.



Note that you do not need to use the debug libraries to use the functions of the debug driver. The release driver and debug driver are compiled from the same source code and respond to the same commands.

You can also use the `sv_debugprint()` function of the SDK to receive debug information from both the debug and the release driver and fill these into a buffer.

2.3.2 The Driver under Linux

The driver for Linux consists of the driver object files, `*.pld` files, and `*.ref` files (see also section “Subfolder ‘driver’ under Linux” on page 2-7). The `*.pld` and `*.ref` files are valid for all operating systems.

The Script Files

For easy operation the driver for Linux comes with some script files that will help you during its installation. With them you can create the driver device nodes or post-compile, load and unload the driver.



The usage of the scripts is described in detail in chapter “Installing the SDK” on page 3-1.

When loading the driver with the script *driver_load*, the driver file must usually be stored in the same directory where the **.pld* and **.ref* files are located.

There is also the possibility to load the driver without this script. For this enter the command:

```
insmod [-f] <driver> [hwpath=<path>]
```

- f Forces to load the driver module even if you are running a wrong kernel version. You can use this option if you have, for example, an SDK for kernel 2.4.2, a kernel 2.4.3 and want to load the driver nevertheless.
- <driver> Specifies the relative or absolute path and the name of the driver object file.
- <path> Specifies the location, where the **.pld* and **.ref* files are stored. If you omit this parameter, the driver tries to load these files from the default location, which is *usr/local/dvs/hw*. In case the files cannot be found, the driver will refuse to load.

To unload the driver it is best to use the script *driver_unload*. Nevertheless, you may also enter the command:

```
rmmod <driver>
```

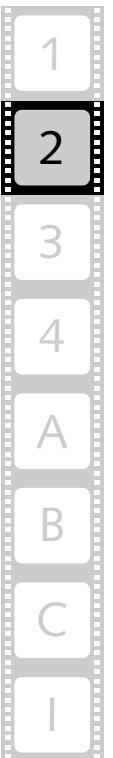
- <driver> For <driver> enter the name of the driver only, i.e. without any path information.

To create (post-compile) the driver it is recommended to use the script *driver_create*, but you may also change into the SDK directory *development/src/driver* and type in *make*. The source files will then be compiled and linked together with the precompiled driver binaries.

To rebuild the final driver files after a kernel version or configuration change it is best to use the script *driver_create* as well. In case you want to perform this step manually use *make clean* in the SDK directory *development/src/driver* to delete all intermediate files and afterwards type in *make*.

Using High Memory

In general the driver is able to handle high memory.



When using `CONFIG_HIGHMEM64G` with an `HDStationOEM` or `SDStationOEM`, the DMA is implemented using a bounce buffer copy which results in a lower data rate and higher CPU load. Centaurus and future PCI-X compliant boards can directly access all memory as well as physical memory that is addressed via PAE.

The Post-compile Driver

The post-compile driver allows to create a driver that matches perfectly to a specific kernel version and/or kernel configuration. In the SDK there is a pre-compiled driver file supplied in binary format along with a few source files containing the kernel-dependent code. The post-compile step can be easily performed on the target system and creates the final loadable driver for that special system.



Further information about the files to post-compile the driver can be found in section “Subfolder ‘src/driver’” on page 2-5, whereas the steps necessary to actually create the final driver are detailed in section “Post-compiling the Driver” on page 3-3.

But the post-compile driver feature does not solve all problems. Here is a list mentioning what is not possible or where to take care of some limitations:

- No cross-compile. As the pre-compiled driver is created for one specific architecture (e.g. i386) it is not possible to create a final driver for the x86-64 or ia64 architecture when using an SDK with a pre-compiled driver for the x86 architecture.
- After each change to the kernel, i.e. every configuration change, patching and the following kernel recompilation, you have to perform the post-compile step on the DVS PCI video board driver again. Otherwise a proper behavior of the driver cannot be guaranteed.
- When kernel interface functions used by the DVS driver are changed from one kernel version to another, it is not possible to post-compile the driver.

During the post-compile step it will be automatically checked which kernel is currently running. At least a matching set of kernel header files need to be installed on your system. These header files are pointed to by `lib/modules/<kernel_version>/build/include`.

When rebuilding the final driver files after a kernel version or configuration change, perform the steps as detailed in section “Post-compiling the Driver” on page 3-3 again.



In the Red Hat distribution kernels (e.g. 2.4.18-3) there are normally some header files missing until the ‘make dep’ build step is performed in your kernel source. Prior to this the driver will not compile properly.

2.3.3 The Driver under Windows

The driver for Windows consists of a **.sys* file and an **.inf* file as well as **.pld* and **.ref* files (see also section "Subfolder 'driver' under Windows" on page 2-8).

The **.sys* file is the driver object file of the PCI video board driver. It can be used to install or update the driver.

If provided, there will also be an **.inf* file available. Then, during the first installation of the software, the driver is installed with this file containing further information for the operating system how to handle the driver. However, in case an update is performed, you will only need the **.sys* file.

The **.pld* and **.ref* files are valid for all operating systems.

The driver for Windows can be easily installed, loaded or unloaded with the help of the DVSConf program (see chapter "Appendix B – DVSConf Program" on page B-1).

When installing or updating the driver under Windows, it is copied automatically to a system directory of the operating system (*<operating system folder>/system32/driver*). This will take place automatically during the loading of the driver with the DVSConf program.

2.3.4 Customized Video Rasters

The DVS OEM product can be used with all kinds of video rasters. The most common ones will already be available to you via the SDK. However, if need arises for special rasters not already accounted for, DVS offers you the possibility to develop such rasters for you. Once installed properly, they can be used with the same ease as all other rasters.

To create a raster update you can use a utility named *cpraster* which creates a *userdef.ref* file from raster files. It can be accessed in the folder *sdk<x>.<y>p<z>/<operating system>/driver/ref* and will create a new *userdef.ref* in the driver directory.



Further information about how to create customized rasters with this utility can be found in the text file stored in the directory of the utility.

1

2

3

4

A

B

C

I

2.4 The DVS Header Files

This section introduces all header files that are part of the SDK.

2.4.1 Video C Library ('dvs_clib.h')

The video C library defines prototypes for setup, control, data transfer, and many other functions. It is intended to interface video devices into a user program. The C library is also used by the DVS command line interface (see chapter "Appendix C – svram Program" on page C-1).

All functions except `sv_open()` check for a valid `sv_handle`. The magic field is initialized by `sv_open()`. Treat the handle as a void pointer. The handle data is private and different for each implementation.

Included and Including Files

When using any of the video C library functions, you have to include the header file `dvs_clib.h` in your C source code. This file defines the prototypes for all functions. It also includes the header files `dvs_errors.h` defining error codes and their plain-text messages, and `dvs_version.h` containing internal version information. All prototypes are only used with a compiler that has defined `__STDC__` (standard C).

Linking Information

The video C library contains only global routines starting with the prefix `sv_`. The only global variable is the variable `sv_debug` which determines whether a debug output will be generated.

- Under Linux link your program against the `libdvsoem.a` library.
- Under Windows link your program against the `dvsoem.dll` library. For this include the file `dvsoem.lib` from the `sdk<x>.<y>p<z>/win<32, 64>/lib` directory in your project.

Error Messages

All SDK errors codes are listed in the "SDK – Software Development Kit" reference guide. In the following you can find some general errors that may occur when using the video C library functions:

<code>SV_ERROR_SVNULL</code>	Will be returned by all functions except <code>sv_open()</code> if the parameter <code>sv</code> of <code>sv_handle</code> is <code>NULL</code> .
------------------------------	---

<code>SV_ERROR_SVMAGIC</code>	All functions except <code>sv_open()</code> will return this error code if the magic field in the <code>sv_handle</code> is wrong.
<code>SV_ERROR_NOTIMPLEMENTED</code>	Will be returned by a function if it is not supported by the current operating system.

2.4.2 Error Code Header ('dvs_errors.h')

This header contains the codes, flags, and char descriptions as well as plain text messages for possible errors in the SDK software. It is included in `dvs_clib.h`, so you do not have to include it in your C sources when including `dvs_clib.h`.

2.4.3 FIFO API ('dvs_fifo.h')

The header `dvs_fifo.h` defines prototypes for the FIFO API functions. The FIFO API has to be used in direct I/O video applications with the DVS video and audio hardware. Direct I/O applications are those that fulfill the following conditions:

- They use CPU processing for video/audio in system memory instead of only transferring the data via the PCI bus.
- They are capable of sending data to the output or receiving data from the input in real-time.

The main purpose of the FIFO API is to allow applications or the operating system to access video data during transfers. This way it is possible to display the data directly in an imaging software or to store them on a system disk. For example, with this you can develop a still store application that uses the system's RAM or a disk recording application in DMA operation mode. For other uses consider the usage of the video C library functions (see section "Video C Library ('dvs_clib.h')" on page 2-14).

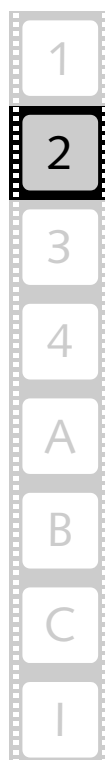
Including Files

When using any of the FIFO API functions, you have to include `dvs_fifo.h` and `dvs_clib.h` in your C source code.

Linking Information

Under Linux link your program against the library `libdvsoem.a`, which will implement the C library as well as the FIFO API.

Under Windows link your program against the library `dvsoem.dll`, which will implement the C library as well as the FIFO API. For this in-



clude the file *dvsoem.lib* from the *sdk<x>.<y>p<z>/win<32, 64>/lib* folder in your project.

Error Messages

All SDK errors codes are listed in the “SDK – Software Development Kit” reference guide. In the following you can find some general errors that may occur when using the FIFO API functions:

SV_ERROR_DISPLAYONLY	Will be returned if the selected video raster is only supported as an output format and an input FIFO is opened.
SV_ERROR_NOTSUPPORTED	If a PCI video board is used that does not support DMA, this error code is returned. In this case the <code>dmaalignment</code> in the <code>sv_fifo_status()</code> function will be zero (0) as well.
SV_ERROR_WRONG_HARDWARE	This error code will be returned if the PCI video board does not support the called operation.
SV_ERROR_NOTIMPLEMENTED	Will be returned by a function if it is not supported by the current operating system. For example, the <code>sv_fifo_dmarectangle()</code> function is not supported under Linux and if it is called, it will return this error code.

2.4.4 File Converter C Library ('dvs_fm.h')

The header *dvfs_fm.h* defines prototypes for file conversion functions. The purpose of the file converter library is to enable a reading and writing of image files in different formats. The file converter will take care of conversions between different color spaces and will rearrange the data in an order suitable for the selected image format.



This header operates in software only. A hardware conversion of color spaces and ranges has to be addressed differently.

There are two interfaces for the file converter. The first interface should be used by an application program that needs to access a file in any format implemented. The other interface enables a reading or writing of an image in a certain file type.

The file converter uses callbacks to perform a linking between the caller function and the actual file access function.

Including Files

When using any of the file converter C library functions, you have to include *dvs_fm.h* in your C source code. This file contains all defines and structures used by the file converter, all defines start with the prefix *FM_*. All non-static functions in the library start with *fm_*. This will not be the case when linking to add-on libraries, such as the JPEG library.

Linking Information

Under Linux link your program against the library *libdvsfm.a*, which will implement the file converter. The library *libdvsfm.a* already includes *libdvstiff.a*, so you do not have to link against *libdvs-tiff.a* yourself.

Under Windows link your program against the library *dvsfm.dll*, which will implement the file converter. For this include the file *dvs-fm.lib* from the *sdk<x>.<y>p<z>/win<32, 64>/lib* folder in your project. The library *dvsfm.lib* already includes *dvstiff.lib*, so you do not have to link against the *dvstiff.dll* yourself.

2.4.5 Setup Header ('dvs_setup.h')

This header defines all data types for C language for each platform, thus establishing data-type standards with a certain lengths.

2.4.6 Thread Header ('dvs_thread.h')

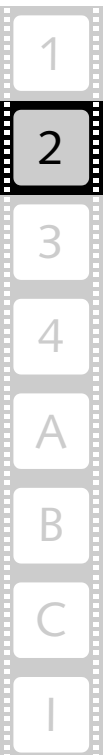
The *dvs_thread.h* header provides basic thread functions for different operating systems. With it you may be able to implement multiple threads in your video and audio I/O application.

2.4.7 Version Header ('dvs_version.h')

The header *dvs_version.h* contains internal version information. It defines the major, minor, and patch version flags of the SDK. It is included in *dvs_clib.h*, so you do not have to include it in your C sources when including *dvs_clib.h*.

The header's content defines an SDK of the version *<x>.<y>p<z>*, for example, an SDK of the version 2.7p47:

```
#define DVS_VERSION_MAJOR      2
#define DVS_VERSION_MINOR     7
#define DVS_VERSION_PATCH     47
```



2.5 The Example Projects

The SDK comes with some example projects which can be analyzed to understand the SDK programming. This section describes the functions and purposes of these example projects. The projects are stored in the *sdk<x>.<y>p<z>/development/examples* folder. To actually run the example projects, you have to compile them first or run the pre-compiled programs that can be found in the *<operating system>/bin* folder in the SDK's main directory.



For more details about the example programs described shortly in the following please refer to their source code directly.

2.5.1 RAM-recorder Example Programs

These examples use functions from the video C library (*dvs_clib.h*). All source codes are linked to the DVSOEM library to demonstrate RAM-recording operations.

pvspeed

Determines the current host transfer speed.

rs422test

The *rs422test* program is an example showing how to use the functions for the 9-pin RS-422 Sony protocol. This example mainly performs subsequent reads and writes on one port as master and on another port as slave. The command sequence is picked up and written to the screen together with its string descriptions.

speedcheck

The *speedcheck* example is similar to the UNIX *dd* command. Mainly, it is a tool to measure data rates. With *speedcheck* you can make transfers between the DVS PCI video board, system RAM and operating system file systems.

sv/svram

This is the source code for the DVS command line, i.e. the *svram* program. With the command line you can set up and control a DVS video device. For further details about how to use the program see chapter "Appendix C – svram Program" on page C-1.

tcfill

Inserts a timecode into the frames in the video storage.

2.5.2 FIFO API Example Programs

The examples in this section demonstrate the utilization of the FIFO API by using functions from the video C library (*dvs_clib.h*) and the FIFO API (*dvs_fifo.h*).

bmpoutput

This program works under Windows only as it uses the Windows 32 bit API and GUI components. It shows the Windows desktop on the video output.

cmodekst

This example demonstrates how to dynamically output frames of different sizes and color modes. It is using the FIFO API to perform the video output.

counter

Generates a counter on black frames in the video buffer (RAM) and displays it on the digital video output. This example shows how to give out image material from an application directly on the video device's output.

dmaloop

This example program demonstrates how to perform a simultaneous input and output. Furthermore, it shows how to add a constant delay between the input and output stream.

dmarect

Shows how to apply the scatter/gather DMA with the FIFO API. It should be used in case you want to have a smaller part of the image only (e.g. as in cutting out an HD frame from a 2K image).

The *dmarect* program captures an image, transfers a small memory area to the CPU memory, removes this area's color information, transfers it back again, and inserts it into the video data.

dmart

This program works under Windows only as it uses the Windows 32 bit API and GUI components. The *dmart* example demonstrates how to



perform a simultaneous video input and output with the least possible delay.

dpxio

Video and audio display/record: The *dpxio* example shows how to display and record DPX file sequences and AIFF audio files. It uses the FIFO API to achieve a simultaneous record or display of video as well as audio. The available options are mainly the same as in the *fileio* example program.



You can find a step-by-step description of the *dpxio* example program in the separate “SDK – Software Development Kit” reference guide.

fileio

Video display/record from/to a file: Records and displays raw or AVI video data from the operating system file system. It demonstrates how to use a single stream transfer with the FIFO API.

Audio display/record from/to a file: Records and plays out AIFF audio data on the specified channels by using the FIFO API.



The example *fileio* does not support I/O of eight-channel audio files. It can only read and write two-channel (stereo) files. When recording, you can select the stereo signal that shall be written to the file. When playing out, the audio stereo data from the file is output simultaneously on all stereo channels.

logo

This example demonstrates how to effect a simultaneous input and output FIFO. It records the incoming video, inserts a logo into each frame and outputs the material again.

preview

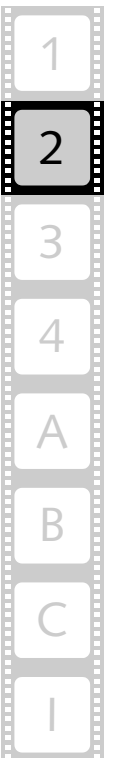
This program works under Windows only as it uses the Windows 32 bit API and GUI components. It shows the incoming video in a window. The data is transferred from the video device with DMA into the main memory and then converted into RGB and displayed.

proxy

This program works under Windows only as it uses the Windows 32 bit API and GUI components. It shows the usage of the Capture API and displays the current output signal in a down-scaled format inside a desktop overlay.

vsync

This program works under Windows only as it uses the Windows 32 bit API and GUI components. It is an example of how to synchronize threads on the basis of vertical syncs.

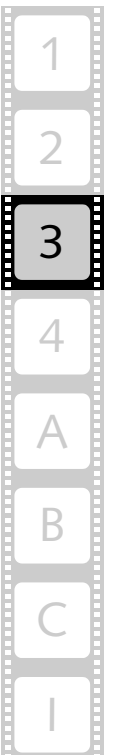


Installing the SDK



This chapter describes how to install the software package of the SDK and how to set up the development and runtime environment. Because the latter is different for the various operating systems the SDK is provided for, the installation procedures will be described for each operating system separately.

Afterwards some information will be given what to do in case you want to update an existing SDK to a newer version.



3.1 Installation under Linux

This section describes how to install the SDK under the Linux operating system. First the installation will be described, followed by explanations about how to create and load the driver.

3.1.1 Installing the SDK

The following describes how to install the SDK under the Linux operating system. It comes without an installation routine to minimize the effects on the installed operating system.

- Install the hardware of the DVS OEM product as detailed in its separate installation guide.
- Next, log on to the computer system with administrative rights.
- Copy the file of the SDK software package to a location of your choice on your computer system's hard disk where the SDK should be installed.
- Then extract the contents of the file, for example, with the tar program by typing in `tar [xvzf] <sdk software package file>`.

This will unpack the SDK to the selected location. The main directory of the SDK `sdk<x>.<y>p<z>` will be created automatically. It will contain all the files that you need to use the SDK. For more detailed information about the installed folders and files please see section "Overview of Folders and Files" on page 2-4.

After this is done you have to continue the installation with the next step, i.e. with the creation of the device nodes.

3.1.2 Creating Device Nodes

During the extraction of the SDK software package file, the driver is placed into the folder `sdk<x>.<y>p<z>/linux/driver`. Before you can operate the PCI video board you have to create device nodes. Under Linux all devices are accessed through device nodes, which are usually placed in the `dev` folder of the operating system. The PCI video board may not be operable when these nodes are missing.

To create driver device nodes perform the following:

- Change to the directory `linux/driver` of the installed SDK and enter the command `./mkdev`.

Once this script has finished, the device nodes are created and will be permanently available.



The script has to be executed only once per system.

Depending on the kernel configuration, the kernel may dynamically create/delete all device nodes (e.g. when using `devfs`). In such a case the DVS driver device nodes will be lost after a reboot and you have to recreate them by using the command `mkdev` again after each booting.

As a next step you have to continue the installation with the creation (i.e. post-compilation) of the driver.

3.1.3 Post-compiling the Driver

Prior to using the DVS PCI video board driver under Linux it has to be post-compiled.

A loading of the driver without performing this step is not possible because the final driver file does not exist yet. There is deliberately no finished driver included in the SDK for Linux due to the fact that kernel configurations may vary from system to system. The Linux kernel is compile-time dependent, thus it is not possible to compile 'one-for-all' binary driver.



Further information about the post-compilable driver can be found in section "The Post-compilable Driver" on page 2-12.

To post-compile the driver the DVS SDK under Linux provides you with a script file that will help you in this step.



Be sure your kernel is already configured properly before proceeding with the following.

During the post-compile step it will be automatically checked which kernel is currently running. At least a matching set of kernel header files have to be installed on your system. These header files are pointed to by `lib/modules/<kernel_version>/build/include`.

- Change into the directory `linux/driver` of the installed SDK and enter the command `./driver_create`.

This will perform the post-compile step and create the final loadable driver module files matching your currently running kernel. As a result you will receive the following files in the directory `linux/driver`:

<code>dvsdriver.ko</code>	The file of the final release driver for the PCI video board.
<code>dvsdebug.ko</code>	The file of the final debug driver.





After each change to the kernel, i.e. every configuration change, patching and the following kernel recompilation, you have to perform the post-compile step of the DVS PCI video board driver again. Otherwise a proper behavior of the driver cannot be guaranteed.

Using these files you can now load the DVS PCI video board driver to control the board.

3.1.4 Loading the Driver

When the above mentioned steps to create device nodes and post-compilation of the driver are finished, you will be able to load the driver. For this perform the following:

- Change into the directory *linux/driver* of the installed SDK and enter the command `./driver_load`.
Alternatively, you can load the debug driver of the SDK with the command `./driver_load debug`.



Be sure you always call the script *driver_load* directly from the *linux/driver* directory because it needs the additional files stored there to load the driver correctly.

If a PCI video board driver is already loaded (e.g. in case of an SDK update), you have to unload the driver first using the script *driver_unload*.

Loading the driver may take some seconds because of the hardware being programmed.

The debug driver is for diagnostics purposes only and should not be used for real operations.

Once the loading of the driver has finished, the installation of the SDK is finished and you are ready to set the license key(s) of your OEM product as described in the installation guide.

3.2 Installation under Windows

This section describes how to install the SDK under the Windows operating system. First the installation will be described, followed by an explanation about how to load and install the driver.

3.2.1 Installing the SDK

The following describes how to install the SDK under the Windows operating system. It comes without an installation routine to minimize the effects on the installed operating system.

- Install the hardware of the DVS OEM product as detailed in its separate installation guide.
- Next, log on to the computer system with administrative rights.
- Copy the file of the SDK software package to your computer system's hard disk.
- Then extract the contents of the ZIP file to a location of your choice.

This will unpack the SDK to the selected location. The main directory of the SDK *sdk<x>.<y>p<z>* will be created automatically. It will contain all the files that you need to use the SDK. For more detailed information about the installed folders and files please see section "Overview of Folders and Files" on page 2-4.

After this is done you have to continue the installation with the next step, i.e. with the loading of the driver.

3.2.2 Loading the Driver

The easiest way to install and load the driver under Windows is to use the DVSCnf program included in the SDK software package. This section describes what to do to load the driver of the DVS PCI video board. There are two possible ways to perform this:

1. If there was never an SDK installed on the computer system and you want to load the driver for the very first time, you have to use the **.inf* file provided with the SDK software package.
2. If a DVS driver was at least once loaded on the system and the PCI video board was not exchanged, you may select the appropriate **.sys* file to update/load the driver.

Both ways are described in this section and you have to act according to your computer system where the DVS hardware is installed.



Loading the Driver for the Very First Time

In case there was never an SDK installed on the computer system where the DVS hardware is installed and you want to load the driver for the very first time you have to do the following:



You may also install the driver with the usual methods provided under Windows (e.g. with the Windows Device Manager). When asked to select a driver, choose the file *dvs.inf* stored in the directory *<operating system>/driver* of the SDK software.

Further information about the DVSConf program can be found in chapter "Appendix B – DVSConf Program" on page B-1.

- Execute the DVSConf program available in the folder *<operating system>/bin* of the SDK software.

This will start the DVSConf program and its user interface will be displayed on the screen.

- In the area **Driver Location** specify the storage location of the *dvs.inf* file by typing in the path and file name in the available entry field (e.g. *C:/sdk<x>.<y>p<z>/<operating system>/driver/dvs.inf*) or click the **BROWSE** button to locate and select the file.

As soon as an **.inf* file is selected, the button **INSTALL** will be made available.

- Then click on the **APPLY** button and afterwards press the **INSTALL** button.

This will install the driver of the DVS PCI video board. As is standard when operating under Windows, it will be copied to the Windows operating system folder *system32/drivers*. Once the loading of the driver has finished, its version number appears in the **Version Information** area of the DVSConf's user interface. Then you may click on the button **OK** to close the program which will finish the installation of the SDK: You are now ready to set the license key(s) delivered with your OEM product as described in the installation guide.

Updating and Loading the Driver

When a driver was already loaded on the computer system and the DVS hardware was not exchanged, you can update/load a driver by using its **.sys* file. For this perform the following:



Further information about the DVSConf program can be found in chapter "Appendix B – DVSConf Program" on page B-1.

- Execute the DVSConf program available in the folder *<operating system>/bin* of the SDK software.

This will start the DVSConf program and its user interface will be displayed on the screen.

- In the area **Driver Loading** select the button **UNLOAD** to unload the driver currently selected for the DVS hardware.
- Next, specify in the area **Driver Location** the storage location of the *.sys file by typing in the path and file name in the available entry field (e.g. *C:/sdk<x>.<y>p<z>/<operating system>/driver/dvswin2k.sys*) or click the **BROWSE** button to locate and select the file.

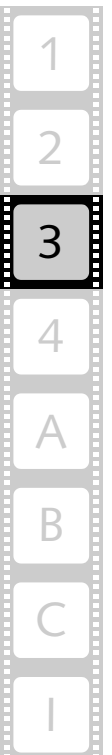


Further information about the different driver object files included in the SDK can be found in section "Subfolder 'driver' under Windows" on page 2-8.

- Then click on the **APPLY** button and afterwards press the **LOAD** button.

This will install the driver of the DVS PCI video board. As is standard when operating under Windows, it will be copied to the Windows operating system folder *system32/drivers*.

Once the loading of the driver has finished, its version number appears in the **Version Information** area of the DVSConf's user interface. Then you may click on the button **OK** to close the program. The installation of the SDK is now complete and you are ready, for instance, to set the license key(s) delivered with your OEM product as described in the installation guide.



3.3 Updating an Existing SDK

DVS is constantly perfecting its products. In time there will be new firmware versions or new versions of the SDK available. This section describes how to update the software package of the SDK.



We recommend to visit the DVS OEM web pages (<http://private.dvs.de/oem>) at regular intervals to check for new versions of firm- and/or software.

Information about how to update the firmware of the PCI video board can be found in the installation guide of the DVS OEM product.

3.3.1 Updating the SDK

To update the SDK to a newer version perform the following:

- Simply replace the old files of the SDK with the new ones.

After this is done you have to update the driver as described in the next section.

3.3.2 Updating the Driver

Once the files of the SDK are updated, you have to replace the old driver version with the new one. For this you have to do the following with regard to the appropriate operating system:

Updating the Driver under Linux

To update the driver under the Linux operating system act as follows:

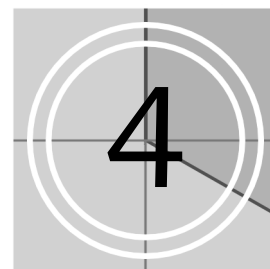
- Perform the post-compile step of the driver as detailed in section “Post-compiling the Driver” on page 3-3.
- Change into the directory *linux/driver* of the installed SDK and enter the command `./driver_unload`.
- Afterwards load the driver as described in section “Loading the Driver” on page 3-4.

With this the SDK is completely updated and you are ready to use your developed application with the new version.

Updating the Driver under Windows

When operating under the Windows operating system, you have to update the driver as described in section “Updating and Loading the Driver” on page 3-6.

Debugging



This chapter contains some information about how to perform a debugging with the SDK.



To assist our OEM customers most efficiently when experiencing troubles, DVS established a web portal solely for the use of the OEM customers. It provides the most up-to-date software versions of the SDK and manuals as well as other information such as frequently asked questions (FAQs).

In case you cannot resolve a problem, it is recommended to check the FAQs provided on the OEM web pages (<http://private.dvs.de/oem>).

4.1 How to Obtain Debug Information

To facilitate you in any debugging task, the driver as well as the DVSOEM libraries are delivered in a release version for normal operation as well as a debug version to track problems that may occur during development.



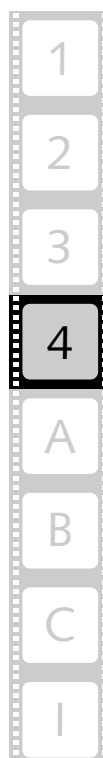
Further information about the release and debug version of the driver can be found in section “Debug Driver vs. Release Driver” on page 2-10.

The DVSOEM debug version of the libraries contains a lot of declarations that will pop up an error window (Windows) or a printf to stdout (UNIX) respectively if anything is wrong. This is the main difference between the debug and the DVSOEM library release version.



See also the define `SV_OPTION_TRACE` in the “SDK – Software Development Kit” reference guide

The debug version of the driver and the libraries run independently from one another. You can use either one of them to track a problem.



4.2 How to Obtain DVS Information Logs

In case of problems with your DVS video device that you are unable to resolve on your own, you may get asked by the DVS service department to create and generate log files, i.e. to gather diagnostic information. This section describes how to accomplish this. It is structured into the operating systems that the SDK is developed for.

4.2.1 Gathering Diagnostic Information under Linux

With the help of the DVS Video Card Info program (DVSInfo) you can write all diagnostic information to an ASCII text file that the DVS support team can evaluate for remote diagnostics of an installation problem. For this perform the following:

- Start the DVSInfo program as described in chapter “Appendix A – DVSInfo Program” on page A-1.
- Then type in 6 to select the menu option ‘6 – Save ALL Information to Logfile’ and press [Enter].

The program now asks for a path and file name to save the log.

- Either press [Enter] right away or specify a path and file name for the log file beforehand.

This will save the file in standard ASCII format. In case you provided a path and file name it will be stored under the specified name in the chosen directory. If not, it will be stored with the default name *dvsinfo.txt* in the directory of the DVSInfo program (i.e. `sdksdk<x>.<y>p<z>/linux/bin`).

- After this locate the file and send it to the E-mail address provided by the DVS personnel.

Once your mail is received, the DVS support team will evaluate the information and, if possible, provide you with a solution for your problem.

4.2.2 Gathering Diagnostic Information under Windows

With the help of the DVS Video Card Info program (DVSInfo) you can write all diagnostic information to an ASCII text file that the DVS support team can evaluate for remote diagnostics of an installation problem. For this perform the following:

- Start the DVSInfo program as described in chapter “Appendix A – DVSInfo Program” on page A-1.

This will start the DVSInfo program and its user interface will be displayed on the screen.

- Enter a file name in the text field in the lower left corner of the DVSInfo window (below the buttons **SAVE** and **EDITOR**).



You may also enter path information in front of the file name to store the file to a specific directory. Otherwise it will be stored in the directory of the DVSInfo program (i.e. *sdk<x>.<y>p<z>/<operating system>/bin*).

- Click on the **SAVE** button.



Alternatively, you may also click on the **EDITOR** button to save the diagnostic information and simultaneously view it in the program that is connected to the file extension of the saved file.

This will save the file in standard ASCII format to the chosen directory.

- After this locate the file and then send it to the E-mail address provided by the DVS personnel.

Once your mail is received, the DVS support team will evaluate the information and, if possible, provide you with a solution for your problem.



4.3 How to Obtain a Streamer Mode Record

A streamer mode record is a complete dump of the SDI stream over a certain period of time. This dump file can be used by the DVS support team to analyze any embedded data packages or data organization. Additionally, this dump file will be useful in case you experienced problems with DVS PCI video boards when recognizing an SDI signal.



A streamer mode record is available for Centaurus and the HDStationOEM only.

To create a streamer mode record you will have received a file named *streamer.ref* or *userdef.ref* from DVS. Then follow the steps described below to the letter:

- If available, make a backup of your *userdef.ref* in the driver directory of the DVS SDK (*sdk<x>.<y>p<z>/<operating system>/driver*).
- Rename the received file to *userdef.ref*, if not already done, and copy it to the driver directory detailed above.
- Next reload the driver (see also section “Updating the Driver” on page 3-8):
 - Under Windows use the DVSConf utility and press the **UNLOAD** button followed by the **LOAD** button after a few seconds.
 - Under Linux use the *driver_unload* and *driver_load* scripts to unload and reload the driver.

After this you will be ready to initialize the streamer mode. There are three different streamer modes for available several signal types:

Streamer Mode	Signal Types
STREAMER	Streamer mode for HDTV signals with a non-dropframe timecode.
STREAMERDF	Streamer mode for HDTV signals with a drop-frame timecode.
STREAMERSD	Streamer mode for SDTV signals.

Make sure you have selected the appropriate mode that matches the signal to be captured and analyzed. For example, if you have a SMPTE274 1080i /29.97 signal, you have to initialize the correct mode with the command *svram mode STREAMERDF*. For further information about the svram program in general see chapter “Appendix C – svram Program” on page C-1.



If a streamer record is performed in the wrong streamer mode, the resulting data will most certainly be useless and you will get a wrong evaluation.

- Initialize the streamer mode with the appropriate svram command as described above.
- Afterwards make sure a correct signal is connected to the SDI input port of the DVS PCI video board and execute the command `svram record ram 0 10`.

This will record the input signal to the RAM of the PCI video board.

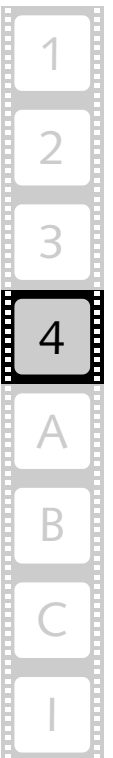
- Next, save the recorded data with the command `svram save streamer file%02d.raw 0 0 10`.
- Locate the saved files `file*.raw` and compress them into an archive.



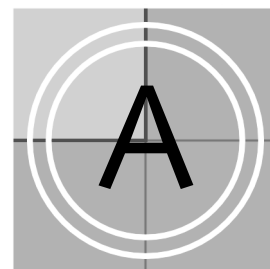
The compression will be most effective when the material that is played out of the source device is either a color bar or a solid-color picture.

- Because each of the streamer's file will be about 10 MB in file size which maybe too large for an E-mail server to handle, make this archive accessible to DVS, for example, on an FTP server or something similar.

Once the files are received by the DVS support team, it will evaluate the information and, if possible, provide you with a solution for your problem.



Appendix A – DVSInfo Program



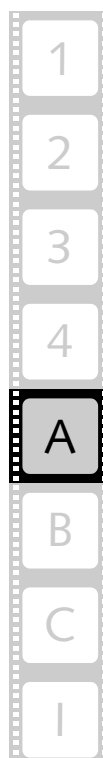
The DVS Video Card Info program (DVSInfo) can be used for diagnostic purposes in case of problems with the PCI video board installation. Under UNIX/Linux operating systems this program can be accessed via a shell or console. Under Windows it offers a user interface.

With the DVSInfo program you can view the PCI configuration of your computer system and generate debug information that can be used for remote diagnosis if a problem occurred on your video system. In essence, with this program you can view and save the log files created by your DVS OEM product.

This chapter explains the basic usage and, in case of Windows, the user interface of the DVSInfo program.



The DVSInfo program is used for system maintenance and service tasks only. If you want to create log files, you can find a description about how to do this in chapter “Debugging” on page 4-1.



A.1 DVSTInfo under UNIX/Linux

This section details how to use the DVSTInfo program under UNIX/Linux operating systems. First its most basic usage is explained, i.e. how to start and end it, followed by a description of the options that can be used with this program.

A.1.1 Basic Usage

This section describes the most basic usage of the DVSTInfo program, i.e. it is explained how to start the program and how to exit it.



Further descriptions about how to use DVSTInfo can be found in section “How to Obtain DVS Information Logs” on page 4-2.

Starting the Program

This section provides you with a description about how to start the DVSTInfo program:

- In a shell/console switch to the installation path of the SDK (its main directory) and change into the folder *<operating system>/bin*.
- There execute the file *dvsinfo*.

This will start the DVSTInfo program by DVS and its starting message is displayed.

Exiting the Program

To end the DVSTInfo program and exit it perform the following:

- Enter *q* at the shell/console or press the key combination [Ctrl + C] alternatively.

After this the DVSTInfo program will be exited and you will return to the prompt of the shell/console.

A.1.2 Options of DVSTInfo

After starting the DVSTInfo program (see section “Starting the Program” on page A-2) a starting message will be displayed:

```
[admin@abc bin]$ dvstinfo

Welcome to the DVS PCI Videocard Information Program

This program is to be used for diagnose problems
with the installation of the DVS videocards in your
computer. With this program you can see the PCI
configuration of your computer and generate
debug information that can be used for remote
diagnose of an installation problem.

DVS GmbH, 1998-2006

The file produced is in normal ascii format and you
can check the contents of it before submitting it.

dvstinfo (? for help):
```

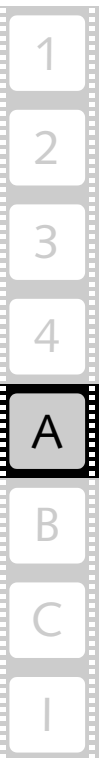
Figure A-1: Starting message of the DVSTInfo program

A menu prompt asks for the menu option to be executed. You can list the possible menu options by entering ? at the prompt:

```
dvstinfo (? for help): ?
1 - Driver
2 - DebugInfo
3 - PCI Scan
4 - PCI Info
5 - PCI Dump
6 - Save ALL Information to Logfile
q - Quit this program
dvstinfo (? for help):
```

Figure A-2: The menu options

The menu options enable you to view or save various log entries recorded during the operation of the PCI video board. When you select one of the menu options 1 to 5, their output will be displayed directly on the screen. The menu option 6 will generate and save a log file.



In detail the menu options of the DVSIInfo program provide the following functions:

Menu Option	Description
1 – Driver	The menu option '1 – Driver' displays detailed information about the driver of the DVS PCI video board.
2 – DebugInfo	Additional information that are useful for a remote error diagnosis can be seen by entering this menu option.
3 – PCI Scan 4 – PCI Info 5 – PCI Dump	These menu options display the PCI configuration of the video system. They will show you in descending order increasingly more information about the PCI configuration.
6 – Save ALL Information to Logfile	In case of problems with your video device, you may get asked by the DVS service department to create and generate a log file. Then use this menu option to store all log entries in one file (in ASCII format). Once entered you may detail a path and/or file name for the log. If no such information is provided, the log file will be stored with the default name <i>dvsinfo.txt</i> in the directory of the DVSIInfo program (i.e. <i>sdk<x>.<y>p<z>/<operating system>/bin</i>).
q – Quit this program	This menu option exits the DVSIInfo program and you will be returned to the prompt of the shell/console.

A.2 DVSIInfo under Windows

When working with the Windows operating system, the DVSIInfo program offers an easy to use user interface. This section describes how to use it. First its most basic usage is explained, i.e. how to start and end the program, followed by a description of the items offered by the user interface of the program.

A.2.1 Basic Usage

This section describes the most basic usage of the DVSIInfo program, i.e. it is explained how to start the program and how to exit it.



Further descriptions about how to use the program can be found in section “How to Obtain DVS Information Logs” on page 4-2.

Starting the Program


This section provides you with a description about how to start the DVSIInfo program:

- Switch to the installation path of the SDK (its main directory) and open the folder *<operating system>/bin*.
- There execute the file *dvsinfo.exe*.

This will start the DVSIInfo program by DVS and its user interface will be displayed on the screen.

Exiting the Program

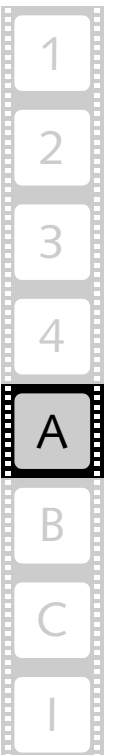
To end the DVSIInfo program and exit it perform the following:

- Click on the close button () or press the key combination [Alt + F4] alternatively.

After this the DVSIInfo program will be closed.

A.2.2 The User Interface

After starting the DVSIInfo program (see section “Starting the Program” on page A-5) its user interface will be displayed on the screen:



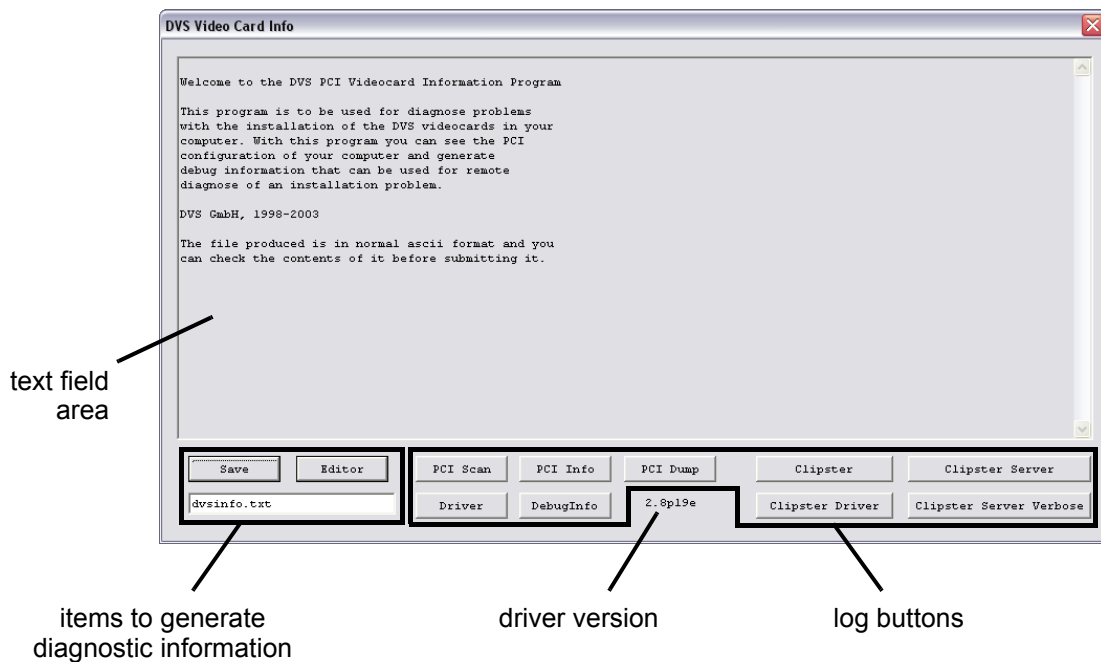



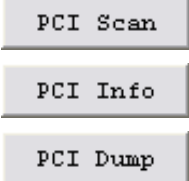
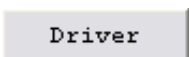
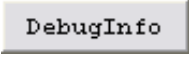

Figure A-3: User interface of the DVSInfo program

The items at the bottom of the window enable you to view and save various log entries recorded during the operation of the software and the video system. When you select one of the log buttons, its entries will be displayed in the text field area in the upper part of the window.



The text field is capable of holding several kilobytes of information. However, the entries logged during operation and displayed here via the log buttons may contain considerably more data. To view all information available it is best to create a log file with the help of the **SAVE** and **EDITOR** buttons.

In detail the user interface of the DVSTInfo program provides the following items:

Item	Description
	<p>In case of problems with your video device, you may get asked by the DVS service department to create and generate a log file. Then use the button SAVE to store all log entries in one file (in ASCII format). The file name of the log file is stated and can be changed via the entry field below the buttons. There you can also enter further path information to store the file in a specific directory. If no additional path information is provided, the file will be stored in the directory of the DVSTInfo program (i.e. <code>sdk<x>.<y>p<z>/<operating system>/bin</code>).</p> <p>With the button EDITOR you can simultaneously save and view the generated log file in the editor that is connected to the file extension of the saved log file.</p>
	<p>These buttons show you the PCI configuration of the video system. The buttons will show you in descending order increasingly more information about the PCI configuration. The information will be displayed in the text field area of the DVSTInfo program.</p>
	<p>The button DRIVER shows detailed information about the driver of the DVS PCI video board in the text field area of the DVSTInfo program.</p>
	<p>Additional information that are useful for a remote error diagnosis can be seen via the button DEBUGINFO. The information will be displayed in the text field area of the DVSTInfo program.</p>
	<p>These buttons provide no functionality when using the DVSTInfo program together with an SDK.</p>

1

2

3

4

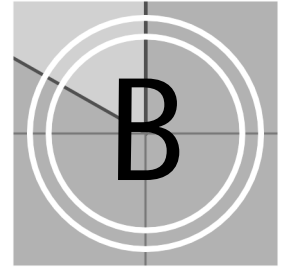
A

B

C

I

Appendix B – DVSCConf Program



Under the Windows operating system you can configure with the DVS Configuration program (DVSCConf) the local PCI video board and its driver as well as test its installation. The settings will be used as the default settings.



This program is available for the Windows operating system only.

The DVS PCI video board and its driver are the central components of the video system because they provide its functionality. Without them your video system would not be able to display any video signals nor would the developed software components be operational. The PCI video board driver controls the video board and thus the in- and output of video signals. It runs in the background of your video system and is automatically loaded under Windows during the starting of the operating system. With the DVSCConf program you can load a PCI video board driver, configure its loading behavior or select another driver instead of the currently loaded one.

Furthermore, with the DVSCConf program you can test the installation of the PCI video board in the system. However, then the DVSCConf affects the PCI video board and its buffer (RAM) only. It does not receive any information from hard disks nor can it access them, for example, for a play-out.

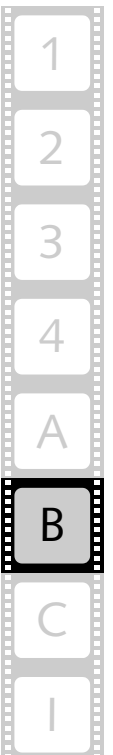
In this chapter the basic usage and the user interface of the DVSCConf program are explained.



Usually, the DVSCConf program is used for system maintenance and service tasks only. Therefore, the following explanations will be brief and provide the most necessary information only.



Because the DVSCConf program configures the driver used for the PCI video board as well as some other video system settings, do not change any of its default configurations unless you are absolutely sure about their outcome, or are instructed to by this user guide or the DVS service department.



B.1 Basic Usage

This section describes the basic usage of the DVSConf program, i.e. it is explained how to start the program and how to exit it.



Further descriptions on how to use the program as well as explanations of the tasks that can be achieved with it can be found in the installation guide of the DVS OEM product delivered to you.

B.1.1 Starting the Program

This section provides you with a description about how to start the DVSConf program:


- Switch to the installation path of the SDK (its main directory) and open the folder *<operating system>/bin*.
- There execute the file *dvscnf.exe*.



This will start the DVSConf program by DVS and its user interface will be displayed on the screen.

B.1.2 Exiting the Program

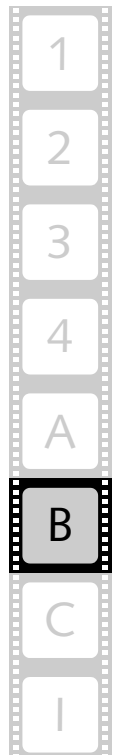
To end the DVSConf program and exit it perform the following:

- Use either one of the following possibilities:

button OK	<p>This button confirms your alterations to the settings in the DVSConf program and closes its user interface. Then the new settings will be in effect for the PCI video board. They will be used as the default settings.</p> <div data-bbox="699 1496 786 1583" data-label="Image">  </div> <p>The settings will be in effect as long as they are not changed by another program such as your developed application.</p>
------------------	---

button CANCEL	<p>The button CANCEL closes the user interface of the DVSConf program without confirming your settings. The latest confirmed settings will be used for the PCI video board.</p> <p> Some changes to the settings will be directly applied to the PCI video board as soon as they are performed in the program. These cannot be reversed with the button CANCEL.</p>
[Alt + F4]	Same as button CANCEL .
	Same as button CANCEL .

After this the DVSConf program will be closed.



B.2 The User Interface

After starting the DVSConf program (see section “Starting the Program” on page B-2) its user interface will be displayed on the screen:

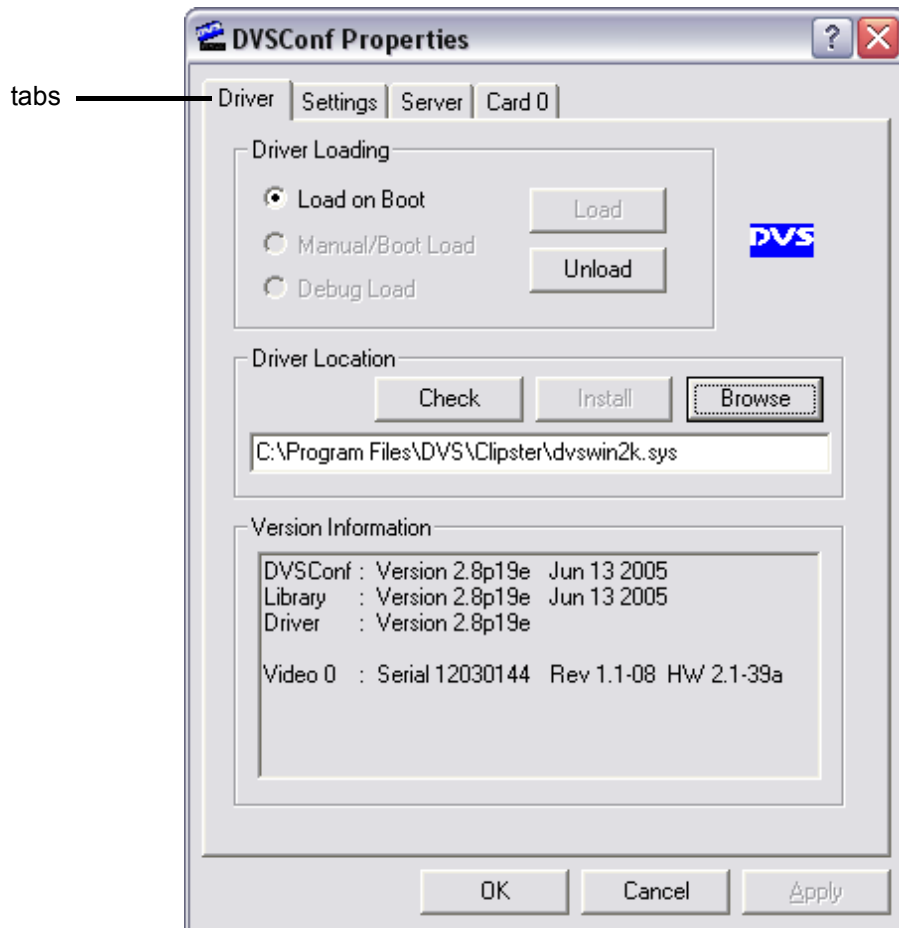


Figure B-1: User interface of the DVSConf program

With the tabs at the top of the window you can switch between the different settings possibilities of the program. Each tab will be explained separately in the next sections.

B.2.1 The Tab 'Driver'

The tab **Driver** of the DVSConf program is used to load a PCI video board driver, configure its loading behavior or select another driver instead of the currently loaded one.

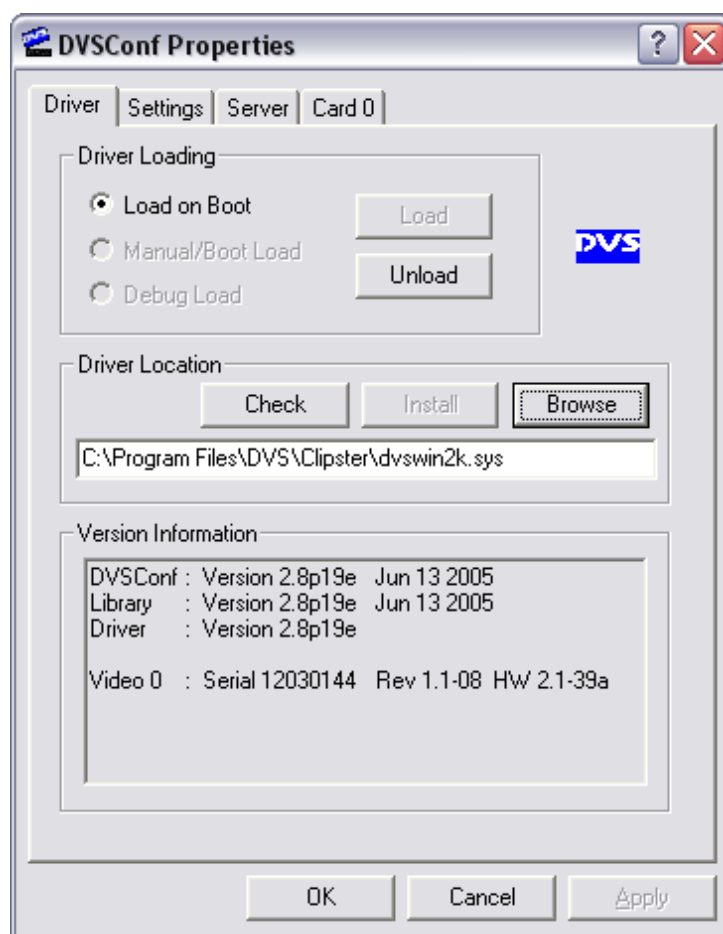




Figure B-2: 'Driver' tab of DVSConf


On the tab **Driver** you can find the following items:



Changes to the driver configuration are directly applied to the video device. There is no undo or cancel function provided and you cannot return to, for example, a formerly activated driver with the button **CANCEL**.

Item	Description
Load on Boot	When this radio button is selected, the system will load the PCI video board driver automatically during the starting of the video system.
Manual/Boot Load	<i>Windows NT only:</i> Offers you the possibility to manually load and unload the PCI video board driver by clicking the respective button to the right of this radio button. Nevertheless, if this radio button is selected, the system will load the driver automatically when the system is started.

Item	Description
Debug Load	<p><i>Windows NT only:</i> With this setting you can manually load and unload the PCI video board driver without initializing any IRQs. This is only useful in conjunction with the PCI video board information program (DVSInfo) to resolve driver problems.</p> <div data-bbox="692 555 791 645">  </div> <p>This setting should not be selected during normal operation. Please note that the system will not be operable until the driver is loaded by automatic or manual loading properly.</p>
LOAD	<p>The button LOAD loads the driver selected via the area Driver Location and connects it to the PCI video board(s) installed in the video system.</p> <div data-bbox="692 920 791 1010">  </div> <p>After clicking on the button LOAD (or after the selection of a new driver) you have to click on the button APPLY to take on the changes. Otherwise, if the program is closed, your changes may not be taken into effect, even if a new driver was loaded with the button LOAD.</p>
UNLOAD	<p>This button reverses the loading of the driver and 'unloads' it. After this you can select another driver with the items of the area Driver Location and then load it with the button LOAD.</p>

Item	Description
Driver Location BROWSE	<p>The text field in this area together with the BROWSE button allows you to locate and select a specific PCI video board driver (*.inf or *.sys files).</p> <p>To install a new driver you have to use and select its *.inf file which incorporates further information for the operating system how to handle the driver. Then click on the button INSTALL to install the driver.</p> <p>However, in case of a driver update, e.g. after installing a new software version, you only have to locate its *.sys file (with the text field or the BROWSE button). After that click the button UNLOAD to unload the currently set driver and shortly afterwards the button LOAD to update the system to the new driver.</p> <div>  <p>After clicking on the button LOAD (or after the selection of a new driver) you have to click on the button APPLY to take on the changes. Otherwise, if the program is closed, your changes may not be taken into effect, even if a new driver was loaded via the button LOAD.</p> </div>
INSTALL	By selecting an *.inf file you can install a new DVS driver instead of updating one. For this you have to use the INSTALL button which will be available as soon as a valid *.inf file is selected in the text field of the area Driver Location .
CHECK	If this button is available, you can confirm with the button CHECK that the driver is compatible to the other modules or libraries of the installed DVS software. If this is not the case, it is recommended to install matching releases of the software or to switch back to a compatible driver.
Version Information	In this text field you can find information about the program, the driver and the hardware versions installed.

1

2

3

4

A

B

C

I

B.2.2 The Tab 'Settings'

The settings on the tab **Settings** are for troubleshooting and problem detection only. They allow for a configuration of the PCI interface as well as of the driver.

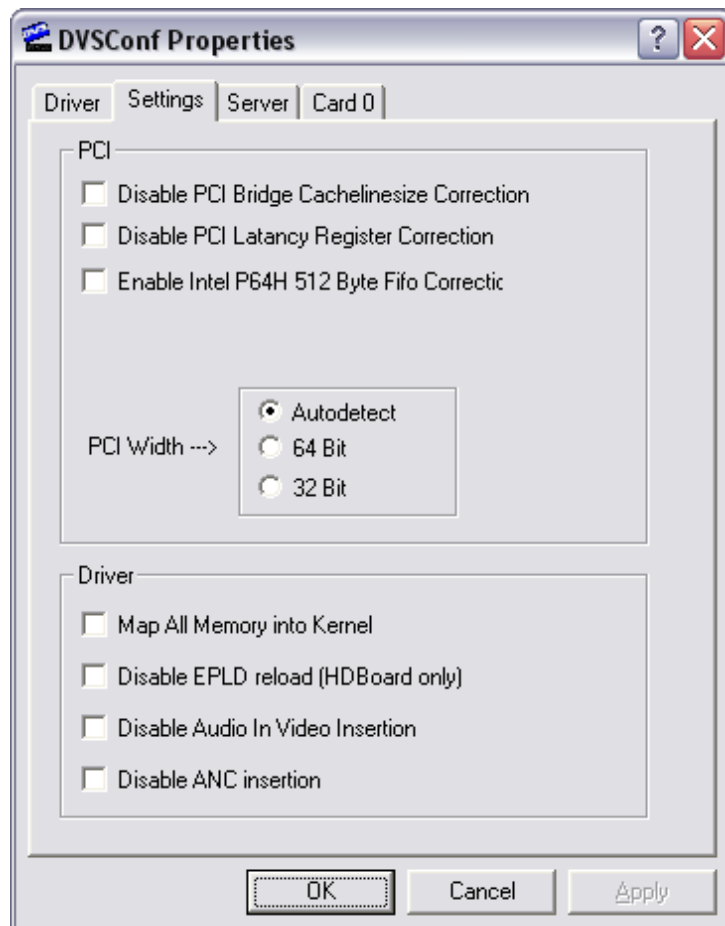


Figure B-3: 'Settings' tab of DVSConf



When changing any of these settings, the video system might not work properly any more. Do not change the default settings unless you are instructed to by the DVS service department.

B.2.3 The Tab 'Server'

The tab **Server** of the DVSConf program is used to configure an automatic log-on feature as well as some operating system settings.

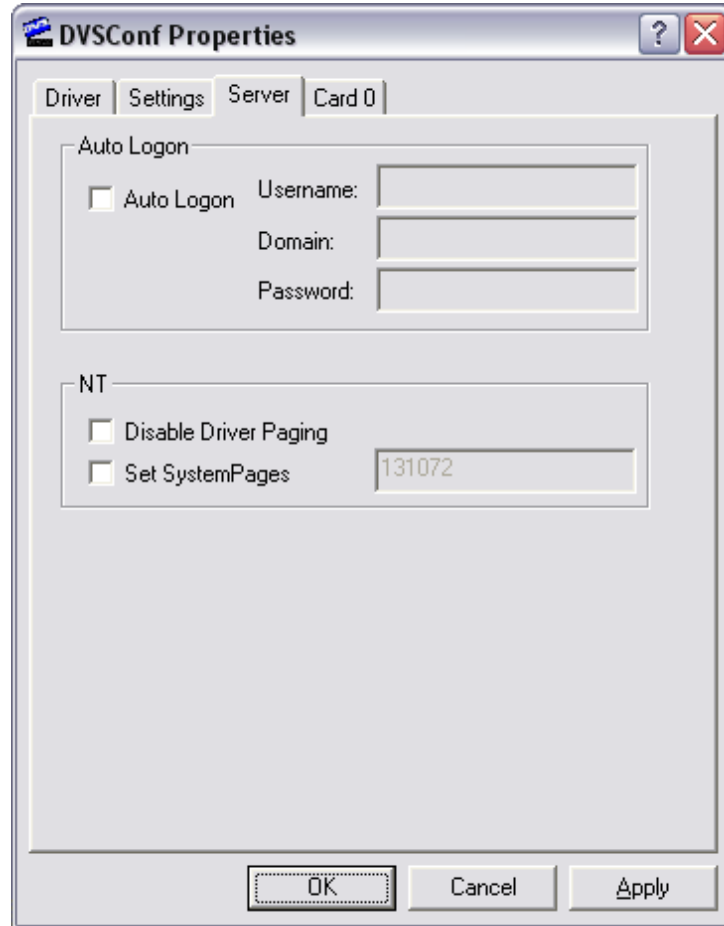



Figure B-4: 'Server' tab of DVSConf

On the tab **Server** you can find the following items:



When changing any of these settings, the video system might not work properly any more. Do not change the default settings unless you are instructed to by the DVS service department.

Item	Description
Auto Logon	<i>ProntoServer only:</i> Enables an automatic log-on to the ProntoServer.  Please note that the password is stored in the Windows registry without any encryption.

Item	Description
Disable Driver Paging	<i>Windows NT only:</i> Drivers will not be paged out of memory. This affects all drivers (see the Windows resource kit). Enable the check box to ensure a proper real-time operation.
Set System Pages	<i>Windows NT only:</i> Normally, when using one PCI video board by DVS, you do not need to set the system pages. Then leave the check box deactivated. In case you have more than one DVS PCI video board installed or little kernel memory free, it may be necessary to increase the default value. A value of 65536 page table entries should be sufficient ($4K \times 65536 = 256 \text{ MB}$). See also the Windows resource kit or contact the DVS service department for details.

B.2.4 The Tab 'Card'

The tab **Card** controls and configures the DVS PCI video board(s) installed in your video system. Via this tab you can set and enable new optional features (license keys) or test the PCI video board installation via its video output. Additionally, with this tab you are able to display and view further information about the respective PCI video board.



The **Card** tab only affects the buffer and the I/O functions of the PCI video board and you can display the data that is already present in the board buffer only.

Prior to using the **Card** tab of the DVSConf program your developed software has to be closed.

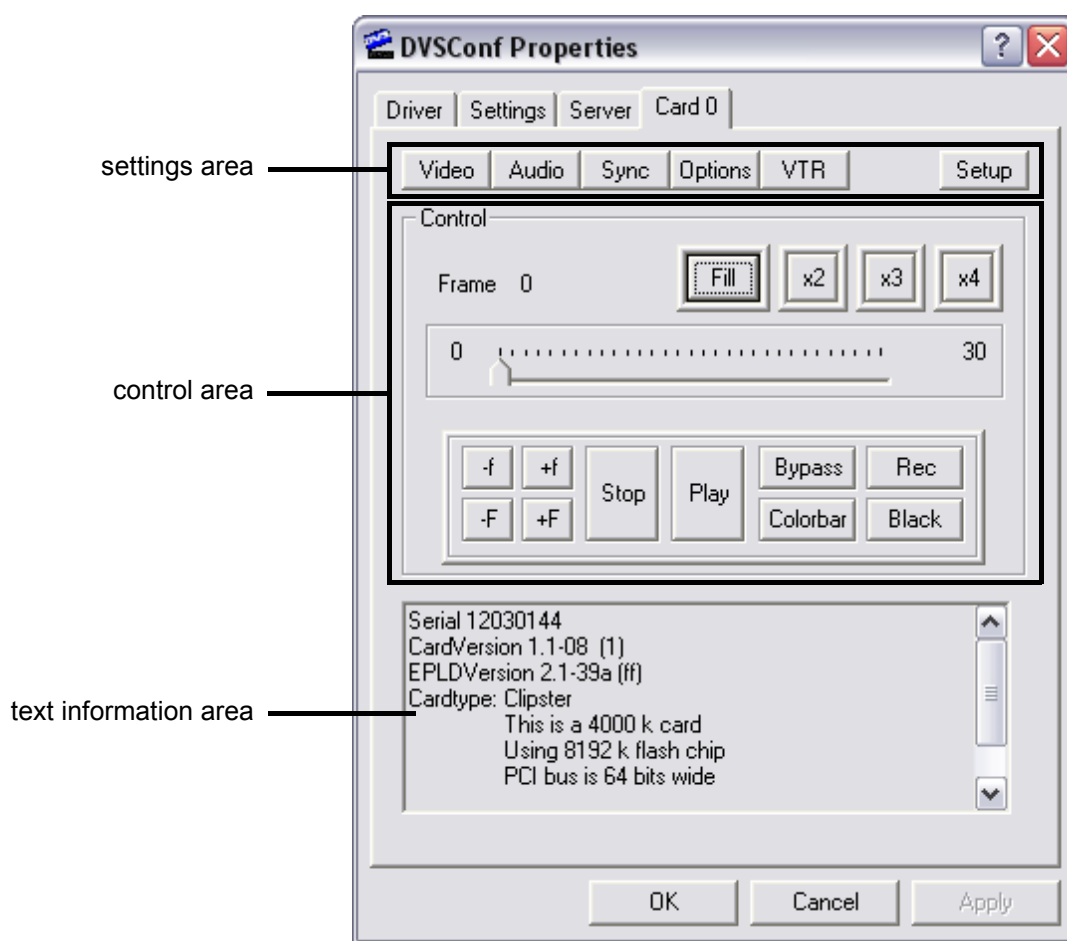


Figure B-5: 'Card' tab of DVSConf

For each installed PCI video board there will be one **Card** tab available. When you have more than one DVS PCI video board installed in your computer system, the tabs will be numbered in ascending order starting with zero (**Card0**, **Card1**, **Card2**, etc.).

On each **Card** tab available you can find a settings, a control and a text information area. These allow you to control and configure the selected PCI video board. They will be explained in detail in the following.

The Settings Area

The settings area allows you to configure and set the digital and analog video as well as audio I/Os of the PCI video board. Additionally, you can configure VTR settings here and view general information about the PCI video board. Furthermore, the setting of new optional features via provided license keys can also be performed here.


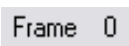



When you click on one of the buttons available in the settings area of the **Card** tab, you will receive a menu where you can select from several options or submenus the desired function. The following settings are provided by the buttons in the settings area:

Item	Description
VIDEO	With this button you can set the video I/O as well as the video storage options of the PCI video board, i.e. the video raster, bit depth and color as well as storage mode.
AUDIO	The menu of the button AUDIO allows you to set the audio I/O and the audio storage options for the PCI video board, such as the audio channels, bit depth and audio frequency. Furthermore, you can specify settings for mute as well as wordclock.
SYNC	Use this menu to set the sync I/O settings for the DVS PCI video board. You can configure the input and output sync differently as well as define various delays.
OPTIONS	With this button you can configure other settings of the in- and output, like the I/O mode of the SDI connectors (digital video) or the analog output.

Item	Description
VTR	Via the menu of the button VTR you can set VTR related options such as edit lag, preroll or postroll settings.
SETUP	<p>With this button to the right of the settings area you can view general information about the hardware and the current settings. They will be displayed in the text information area at the bottom of the Card tab (see section “Text Information Area” on page B-14).</p> <p>Furthermore, the setting of new optionally available features via provided license keys can also be performed here (see installation guide of the DVS OEM product).</p>

The Control Area

The control area on the tab **Card** provides items to generate and display test pictures. This way you can test the PCI video board installation, for example, on the analog monitoring output. In this area you can find the following items:

Item	Description
	Fills the whole PCI video board buffer with test pictures. Using the x buttons you can superimpose the frame's time-code in different sizes over the test images.
	Shows the frame number that is currently displayed at the outputs.
	The slider shows a timeline of the video material loaded into the PCI video board buffer. It visualizes the current position within the material. You can also drag the slider to move to a certain frame. Its length depends on the board buffer size and currently set video raster.
	Use these buttons to move a field backward or forward in the interlaced video material currently loaded into the buffer.
	With these buttons you can move a frame backward or forward in the video material currently loaded into the buffer.

1

2

3



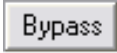
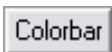
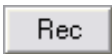
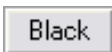
4

A

B

C

I

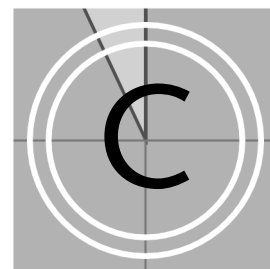
Item	Description
	This button stops the play-out of the video material currently loaded into the PCI video board buffer.
	With this button you can start a play-out of the video material currently loaded into the PCI video board buffer.
	This button displays the input signal that is currently connected to the inputs of the PCI video board.
	This button displays a color bar at the outputs of the PCI video board.
	With the REC button you can record the video signal connected to the PCI video board's input into the board buffer.
	With this button you can display a black frame at the outputs of the PCI video board.

Text Information Area

In the text information area you can view further information about the respective PCI video board installed in your video system. For this simply select from the menu of the button **SETUP** (see section "The Settings Area" on page B-12) the respective information that you want to view. The following information can be displayed in this area:

Info Hardware	Shows information about the PCI video board hardware.
Info License	Shows information about the licensed features that are enabled for the video device.
Info Raster	Displays information about the currently set video raster.
Info Updated	Provides currently no function.

Appendix C – svram Program



This chapter describes briefly the svram program and some of its most useful commands. It is available as an example source code project in the SDK (see section “sv/svram” on page 2-18). The svram program is a command line interface (shell or MS DOS prompt) that controls a DVS video device. With it you can operate the device as a RAM recorder using the on-board RAM of the PCI video board, i.e. the system’s RAM is not affected. An experienced user can use the svram program for setup, control and automated processing with the help of batch files.



This chapter describes the most used commands of the svram program. For further information about any other commands provided by the program please refer to the source code of the example project directly.



C.1 Using the svram Program

After the installation of the DVS SDK the svram program will already be available in the `<operating system>/bin` folder of the SDK's installation path (in case of Linux as pre-compiled versions). To use it you have to call up a shell (MS DOS prompt) manually and then change to the folder `bin` in the installation directory of the SDK. From here you can enter svram commands. Furthermore, you can use the svram program to control the video device with the help of self-created batch files.



Please note that all settings configured via svram are of a temporary nature only, i.e. will not be in effect after a reboot.

C.1.1 svram versus sv

The svram program comes in two versions that use both the same command syntax:

svram	svram only affects the buffer (RAM) of the PCI video board. You call up the commands by entering svram <code><subcommand></code> .
sv	Obsolete. The sv program was used to affect video disks in a proprietary file format of a DVS video system. It was called by entering sv <code><subcommand></code> .



All commands in this chapter are described by using the 'svram' syntax.

C.1.2 Operation Modes

With the svram program you can use a DVS video device as a RAM recorder by affecting the on-board RAM of the PCI video board. It is provided with two kinds of operation modes: the standard user mode and the pipe mode.

When you start the svram program via a command line, it will be in its standard user mode and all commands have to be entered as described in section "svram versus sv" on page C-2 with `svram` in front followed by a subcommand.

You also have the possibility to switch to the pipe mode with the command `svram pipe` (see also command `svram pipe` on page C-12). In the pipe mode the svram program stays active and is not terminated after the execution of a command as in the standard user mode. All commands will be directly passed to the svram program and any oper-

ating system commands will not work from this prompt as long as the pipe mode is active.

In the pipe mode you do not have to type `svram` at the beginning of each command. The command line prompt will change to `<pipe>` to remind you that this special mode is active. You can exit the pipe mode by entering `quit`, or pressing [Ctrl + D] (UNIX/Linux) or [Ctrl + C] (Windows).

C.1.3 Obtaining Help

The `svram` program offers an online help that can be invoked with **`svram <specifier> help`**. For `<specifier>` you may enter a command name without the `svram`.

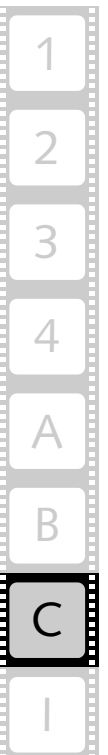
For example, to get help on `svram analog` enter `svram analog help`.

C.1.4 Setting the Environment Variable

The environment variable `SCSIVIDEO_CMD` specifies the DVS video device to be accessed. With it you can access a particular installed PCI video board directly, for example, if you have more than one board installed in a system. Then use `PCI, card: <x>` for the environment variable with `<x>` as the board number to access a particular board (counting from zero (0)):

UNIX: **`setenv SCSIVIDEO_CMD=PCI,card:<x>`**

Windows: **`set SCSIVIDEO_CMD=PCI,card:<x>`**



C.2 Command Reference

This section provides a reference of the most used commands of the svram program in alphabetical order.



Please note that some of parameters of the svram program may not be supported on your DVS PCI video board. The capability of the svram program depends on the hardware features and the licensed options.

For further information about any other commands provided by the program please refer to the source code of the example project directly.

svram analog

Syntax:	svram analog {info, auto, input, output, black, colorbar, {forcenone, forcepal, forcentsc} {blacklevel0, blacklevel7.5} {YC, YUV, YUVS, RGB, RGBS, CVBS}}	
Usage:	Configures the analog video output. You can configure the following:	
Parameters:	info	Returns the current analog output settings.
	auto	Automatic switch between input and output.
	input	Shows the input signal only.
	output	Shows the output signal only.
	black	Shows a black frame on the output.
	colorbar	Shows a color bar frame on the output.
	forcenone	Switches the color carrier according to the video mode.
	forcepal	Forces the color carrier to PAL, independent of the video mode.
	forcentsc	Forces the color carrier to NTSC, independent of the video mode.
	blacklevel0	NTSC voltage level for Japan.
	blacklevel7.5	NTSC voltage level for USA.
	YC	Sets the analog output to Y/C mode.
	YUV	Sets the analog output to YUV mode with sync on Y.
	YUVS	Sets the analog output to YUV mode with separate sync.

RGB	Sets the analog output to RGB mode with sync on green.
RGBS	Sets the analog output to RGB mode with separate sync.
CVBS	Sets the analog output to CVBS (composite) mode.

Related SDK `sv_option(SV_OPTION_ANALOG)`
 Functions: `sv_query(SV_QUERY_ANALOG)`

svram audioinput

Syntax: **svram audioinput {aes, aiv, info}**
 Usage: Selects the port that is used for the audio input during recording or live mode.
 Parameters: `aes` Selects the digital AES/EBU audio input.
 `aiv` Selects the video input, i.e. records the audio data that is embedded in the video signal (audio in video).
 `info` Returns information about the current audio input setting.
 Related SDK `sv_option(SV_OPTION_AUDIOINPUT)`
 Functions: `sv_query(SV_QUERY_AUDIOINPUT)`

svram black

Syntax: **svram black**
 Usage: Generates a black output signal.
 Related SDK `sv_black()`
 Functions:

svram colorbar

Syntax: **svram colorbar**
 Usage: Generates a color bar output signal.
 Related SDK `sv_colorbar()`
 Functions:



svram debugprint

Syntax: **svram debugprint**

Usage: Displays internal driver information and calls of the driver functions.

Related SDK `sv_debugprint()`
Functions:

svram display ram

Syntax: **svram display ram** [`<#start>=0`
`[<#nframes>=1]`]

Usage: Displays uncompressed images from the PCI video board's buffer. Any display speed that you have set with the command `svram speed` will be maintained (see command `svram speed` on page C-13).

`<#start>` First frame of the buffer's timeline to be displayed.

`<#nframes>` Specifies the number of image files that should be displayed.

Related SDK `sv_display()`
Functions:

svram goto

Syntax: **svram goto** `<#frame>` [`{field1, field2, frame}`]

Usage: Changes the current position in the PCI video board's buffer to frame number `<#frame>` and displays it. Optionally, you may specify whether `field1`, `field2` or the whole `frame` shall be displayed.

Related SDK `sv_position()`
Functions:

svram info

Syntax: **svram info**

Usage: Displays information about the current operation mode.

Related SDK `sv_query()`
Functions:

svram inputport

Syntax: **svram inputport** {**analog**, **dvi**, **sdi**, **sdi2**, **sdi3**, **info**}=**sdi**

Usage: Selects the port that is used as the video input for recording or live mode.

Parameters: **analog** Selects the analog input port.
dvi Selects the DVI input port.
sdi Selects the first SDI port.
sdi2 Selects the second SDI port.
sdi3 Selects the third SDI port.
info Shows the currently selected input port.

Related SDK **sv_option**(SV_OPTION_INPUTPORT)
 Functions: **sv_query**(SV_QUERY_INPUTPORT)

svram iomode

Syntax: **svram iomode** {**YUV**, **YUV422**, **YUV444**, **RGB**, **YUV422A**, **YUV444A**, **RGBA**, **YUV422/12**, **YUV444/12**, **RGB/12**, **info**}=**YUV422**

Limitations: Video device with RGB option only.

Usage: Switches the SDI I/O color format between YC_bC_r (single link) and RGB (dual link). YUV provides the same format as YUV422. The /12 parameters switch the I/O mode to a 12 bit color depth.

Related SDK **sv_option**(SV_OPTION_IOMODE)
 Functions: **sv_query**(SV_QUERY_IOMODE)

svram licence

Syntax: **svram licence** {**key1** <key value>, **key2** <key value>, **key3** <key value>, **show**}

Usage: Sets the license keys for the video device. Each license key enables one or more (optional) features of the system until date of expiration. When the video system starts, all keys are checked and their functions are combined.

Parameters: **key1** Sets the license code <key value> for this license key. A key value can enable several features at once. This key is usually used to license the features you have ordered for your video device.



key2, key3 Same as key1 but usually used for temporary licenses that you obtained for evaluation purposes.

show Returns information about the currently installed license keys as well as system specific data.

Note: After entering new license keys you have to reboot the video system to enable the newly licensed features. The license keys will be stored non-volatile.

Related SDK Functions: `sv_getlicence()`

svram live

Syntax: **svram live**

Usage: Direct bypass of input to output (also known as EE or live mode). If the sync mode is set to any other than external, a black frame will be output instead of the live image.

Related SDK Functions: `sv_live()`

svram load

Syntax: **svram load** <fileformat> <filename>
<#frame> [<#start>=0] [<#nframes>=1]
[<channels>=v]

Usage: Loads uncompressed images and audio from the system disk to the PCI video board's buffer.

Parameters: <file-format> Supported file formats are, for instance, `bmp` and `tiff` for video and `aiff` for audio.

<filename> Insert the file name (if necessary with path) of the file that you want to load. If you want to load a continuously numbered sequence of images, the file name should follow the C notation (as, for example, in `scene%04d.bmp`).

<#frame> Specifies the frame number in the buffer's timeline where the sequence should start.

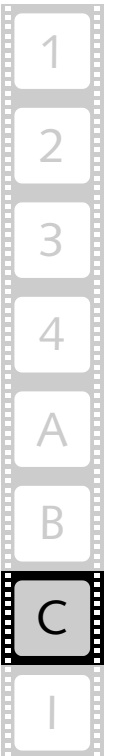
<#start> If you do not want to start with the first image file of the sequence, specify a start file number.

- <#nframes> Specifies the amount of image files that should be loaded.
- <channels> This parameter specifies the data (channels) that should be loaded. It is a combination of *v* for video, 12 for the first audio channel pair, 34 for the second audio channel pair, and so on, and *k* for key.

Related SDK `sv_host2sv()`
Functions:

svram mode

- Syntax:** `svram mode <standard>/<framerate>[/ {8B, 10B}=8B] [/ {YUV422, YUV422A, YUV444, YUV444A, RGB, RGBA, MONO, CHROMA}=YUV422] [/ {FIELD, FRAME, RAW}=FIELD] [/BOTTOM2TOP]`
- Usage:** Sets the video mode of the PCI video board's RAM which is used as the buffer between I/O (SDI) and storage. It allows to configure the I/O independently of the storage. The parameters following a slash (/) may come in any order; the last parameter specified overrides any specified earlier.
- Parameters:**
- <standard> *Obligatory:* Specifies the scan format standard of the video raster. This parameter must precede all other parameters.
 - <framerate> *Obligatory and for HD devices only:* Specifies frame rate and scanning format of the video raster. For rasters with 1/1001 frequencies the decimal place is omitted (e.g. 59P is used for a raster in 59.94 Hz progressive). For a list of the supported video rasters see the "SDK – Software Development Kit" reference guide.
 - 8B, 10B The quantization, i.e. bit depth.
 - YUV422, YUV422A, etc. The color mode.
 - FIELD The storage format independent of the I/O format. The images are stored as fields.



FRAME	The storage format independent of the I/O format. The images are stored as frames.
RAW	The storage format independent of the I/O format. The images are stored as streamer data.
BOTTOM2TOP	Inverts the incoming video data by turning each image from bottom to top. This is useful for Windows files that represent images starting with the bottom line. For example, with the mode /RGB/BOTTOM2TOP, a 24 bit BMP file can be loaded into memory and be displayed without any conversions.



Formats that do not have an integral number of bytes per line cannot be handled by this parameter (e.g. SMPTE296/10B).

Related SDK Functions: `sv_option(SV_OPTION_IOMODE)`
`sv_query(SV_QUERY_FEATURE)`
`sv_videomode()`

svram outduringrec

Syntax: **svram outduringrec {default, bypass, black, colorbar, input, output}**

Usage: Sets the signal that the video device should output during a record operation.

Parameters: `default` Restores the default behavior.
`bypass` Shows the live signal present at the video input (direct bypass mode).



When in direct bypass mode, make sure that the input device is not connected to the video system's output or that it is capable of managing its own sync signal. Otherwise both systems try to lock on to each other's sync, which will most likely result in strange reactions.

`black` Shows a black frame.
`colorbar` Shows a color bar frame.

input Shows the input video (via DRAM).
 output Shows the signal that was output before the record operation.

Related SDK Obsolete when using the FIFO API, otherwise:

Functions:

`sv_option(SV_OPTION_OUTDURINGREC)`

`sv_query(SV_QUERY_OUTDURINGREC)`

svram outputport

Syntax: **svram outputport {default, mirror, swapped}**

Usage: Some DVS PCI video boards provide two dual-link outputs, i.e. both outputs give out the same signal. With this command you can set which port of the dual-link outputs shall give out the main (e.g. YC_bC_r video) and which shall give out the secondary signal (e.g. key).

Parameters:

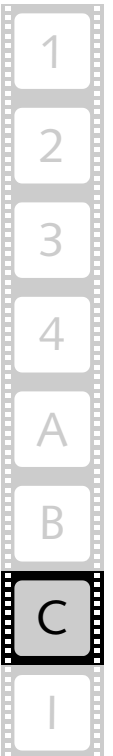
<code>default</code>	The A port is the main output: The A port gives out the video data, the B port gives out the key data.
<code>mirror</code>	Output ports A and B show both the output signal of port A.
<code>swapped</code>	The B port is the main output: The A port gives out the key data, the B port gives out the video data.

Note: Swapping the output ports is always possible, but the output signal of the secondary port depends on the licensed options. If a certain signal (e.g. key) is not available, because the corresponding feature is not licensed, a gray signal will be given out instead.

Related SDK `sv_option(SV_OPTION_OUTPUTPORT)`

Functions:

`sv_query(SV_QUERY_OUTPUTPORT)`



svram pipe

Syntax: **svram pipe**

Usage: Sets the svram program into pipe mode meaning that the program stays active and does not terminate after executing a command as in its normal operation mode. All commands will be directly passed to the svram program. Therefore, any operating system commands will not work at this prompt as long as the pipe mode is active.

In pipe mode you do not have to type `svram` at the beginning of each command. The command line prompt will change to `<pipe>` to remind you that this special mode is active. You can leave the pipe mode by entering `quit`, or pressing [Ctrl + D] (UNIX) or [Ctrl + C] (Windows).

Related SDK –
Functions:

svram record ram

Syntax: **svram record ram** [`<#start>=0`
[`<#nframes>=1`]]

Usage: Records uncompressed frame(s) from the video input to the PCI video board's buffer (RAM).

Parameters: `<#start>` First frame that will be overwritten in the buffer's timeline.
`<#nframes>` Number of frames to be recorded.

Related SDK `sv_option(SV_OPTION_RECORD_...)`

Functions: `sv_record()`

svram save

Syntax: **svram save** `<fileformat>` `<filename>`
`<#frame>` [`<#start>=0`] [`<#nframes>=1`]
[`<channels>=v`]

Usage: Saves uncompressed images and audio from the PCI video board's buffer to the system disk.

Parameters: `<file-format>` Supported file formats are, for instance, `bmp` and `tiff` for video and `aiff` for audio.

- <filename> Insert the file name (if necessary with path) that the saved file(s) should provide. If you want to save more than one frame, you can generate a continuously numbered sequence of images. Then the file name should follow the C notation (as, for example, in *scene%04d.bmp*).
- <#frame> Specifies the frame number in the buffer's timeline that should be the start frame of the sequence to save.
- <#start> If you do not want the saved image sequence to start with zero (0), specify a start file number. This is useful when appending frames to an already existing sequence of files.
- <#nframes> Specifies the number of frames that should be saved.
- <channels> This parameter specifies the data (channels) that should be saved. It is a combination of *v* for video, 12 for the first audio channel pair, 34 for the second audio channel pair, and so on, and *k* for key.

Note: Directories will not be created automatically when using this command.

Related SDK `sv_sv2host()`
Functions:

svram speed

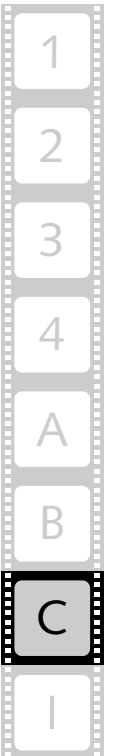
Syntax: **svram speed** <#speed> [{**once**, **loop**, **shuttle**, **reverse**}=**loop**]

Usage: Starts a display with the specified speed and the optionally specified play mode.

Parameters: <#speed> Floating point number or a fraction where one (1) represents normal speed. Negative numbers effect a reverse play-out. A speed value of zero (0) results in a still picture display.

once Plays the timeline in the buffer once.

loop Starts all over again when it reaches the end of the timeline.



shuttle Plays the timeline forward and backward alternately. This mode is also called swing or ping-pong mode.

reverse Plays the timeline in reverse direction.

Related SDK Functions: `sv_option(SV_OPTION_SPEED, SV_OPTION_LOOPMODE)`

svram stop

Syntax: **svram stop**

Usage: Stops all activities of the video device (soft reset). A VTR connected via RS-422 will be stopped as well.

Related SDK Functions: `sv_stop()`

svram sync

Syntax: **svram sync {internal, external, analog, bilevel <level>, trilevel, hvttl <form>, hdelay, vdelay}=internal**

Usage: Sets the sync source for the video device. Usually, the video device is running in a non-genlock mode.

Parameters:

internal	Uses a free running internal sync.
external	Uses the external sync of the digital video input.
analog	Uses the external sync of the analog reference input (genlock mode).
bilevel	Uses the analog genlock mode with a bi-level sync. <level> sets the sync level: <ul style="list-style-type: none"> – 0.3v (0.3 voltage level) – 4.0v (4.0 voltage level)
trilevel	Uses the analog genlock mode with a tri-level sync.
hvttl	<i>HD devices only:</i> Uses the analog genlock mode with a hvttl sync. <form> sets the sync signal form: <ul style="list-style-type: none"> – hfvf (horizontal falling, vertical falling) – hrvf (horizontal rising, vertical falling) – hfvr (horizontal falling, vertical rising) – hrvr (horizontal rising, vertical rising)

hdelay Sets the number of pixels that the horizontal sync output gets delayed in relation to the incoming sync. The interval is in pixels: half-pixels for SD devices, full pixels for HD devices.

vdelay Sets the number of lines that the vertical sync output gets delayed in relation to the incoming sync. The interval is in lines.

Related SDK `sv_sync()`

Functions:

```
sv_option(SV_OPTION_SYNCMODE,
SV_OPTION_HDELAY, SV_OPTION_VDELAY)

sv_query(SV_QUERY_SYNCMODE,
SV_QUERY_HDELAY, SV_QUERY_VDELAY)
```

svram version

Syntax: **svram version** [**info**]

Usage: Returns version information of the installed hard-, firm- and software.

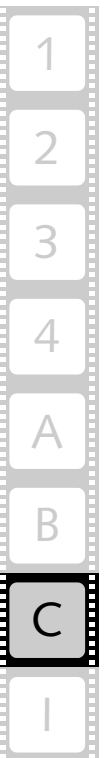
Parameters: **info** Returns a more user-friendly output of the version information.



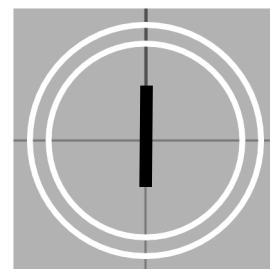
This parameter will be especially useful in conjunction with the SDStationOEM II. Use it to determine its PCI interface version more easily.

Related SDK `sv_version_status()`

Functions:



Index



A

AIFF 2-20
 AIV C-5
 analog output B-12, C-4
 analog video B-12, C-4
 audio channels B-12, C-9
 audio frequency B-12
 audio input C-5
 Auto Logon B-9
 AVI 2-20

B

bit depth B-12, C-7, C-9
 black frame B-14, C-4, C-5, C-8, C-10
 buffer (PCI video board) B-11, B-13,
 C-2, C-6, C-8, C-12
 button
 Apply B-6, B-7
 Audio B-12
 Black B-14
 Browse 3-6, 3-7, B-7
 Bypass B-14
 Cancel B-3
 Check B-7
 Clipster A-7
 Clipster Driver A-7
 Clipster Server A-7
 Clipster Server Verbose A-7
 Colorbar B-14
 Debug A-7
 Driver A-7
 Editor 4-3, A-7
 Fill B-13
 Install 3-6, B-7
 Load 3-7, 4-4, B-6, B-7
 OK 3-6, 3-7, B-2

Options B-12
 PCI Dump A-7
 PCI Info A-7
 PCI Scan A-7
 Play B-14
 Rec B-14
 Save 4-3, A-7
 Setup B-13, B-14
 Stop B-14
 Sync B-12
 Unload 3-7, 4-4, B-6, B-7
 Video B-12
 VTR B-13
 x B-13

C

C header 2-4
 C source 2-4
 chapter overview 1-2
 color bar B-14
 color mode B-12, C-7, C-9
 command syntax 1-4
 svram C-2
 configuration (PCI video board) A-1,
 A-4, A-7, B-11
 confirmation of settings B-2
 conventions
 command descriptions 1-4
 of user guide 1-3
 customized rasters 2-13

D

debug driver 2-7, 2-8, 2-10, 3-3
 load (Linux) 3-4



debug information	2-10, 2-14, 4-1, A-1
debug libraries	2-10, 4-1
Debug Load	B-6
debugging	4-1
device node	2-7, 3-2
diagnostic information	4-2
gathering under Linux	4-2
gathering under Windows	4-2
digital audio	C-5
digital video	B-12, C-9
Disable Driver Paging	B-10
DPX	2-20
driver	2-2, 2-7, 2-8, 2-10, 3-3, B-1
debug driver	2-10
driver files	2-7, 2-8, B-7
driver files (Linux)	2-10
driver files (Windows)	2-13
file extensions (Linux)	2-7
file extensions (Windows)	2-8
high memory	2-11
information	A-4, A-7
load (Windows)	B-6
object file	2-10, 2-13
post-compile	2-5, 2-7, 2-11, 2-12, 3-3
release driver	2-10
script files	2-7, 2-11, 3-3, 3-4
settings	B-8
update (Linux)	3-8
update (Windows)	3-6, B-7
version information	B-7, C-15
driver folder (Linux)	2-5, 2-7
driver folder (Windows)	2-8
Driver Location	B-7
dump file	4-4
DVS Configuration program	see DVSSConf
DVS service department	4-2, A-4, A-7
DVS Video Card Info program	see DVSSInfo
DVSSConf	3-5, B-1
exit program	B-2
start program	B-2
tab 'Card'	B-11
tab 'Driver'	B-4
tab 'Server'	B-9
tab 'Settings'	B-8
user interface	B-4
version information	B-7
DVSSInfo	2-10, 4-2
exit program (Linux)	A-2
exit program (Windows)	A-5
start program (Linux)	A-2
start program (Windows)	A-5
user interface (Linux)	A-2
user interface (Windows)	A-5
dynamic link libraries	2-6

E

edit lag	B-13
environment variable	C-3
error messages	2-10, 2-14, 2-15, 2-16, 4-1
example projects	2-18
folder	2-5
makefile	2-5
workspace file	2-5
exiting DVSSConf	B-2
exiting DVSSInfo	
Linux	A-2
Windows	A-5

F

field backward	B-13
field forward	B-13
FIFO API	2-15, 2-19
file converter library	2-16
formatting	2-5
frame backward	B-13
frame forward	B-13
frame number	B-13, C-6, C-8, C-12, C-13

H

hardware information	B-14, C-15
header files	2-5, 2-14
high memory	2-11

I

I/O mode	B-12, C-7
icon	2-4
information	
debug	A-4, A-7
debug information	2-10, 2-14, 4-1, A-1
diagnostic	4-2
driver	A-4, A-7
hardware	B-14, C-15
licensed features	B-14, C-8

video raster B-14, C-6
 input signal B-14, C-5
 installation 3-1
 Linux 3-2
 Windows 3-5
 installation test (video board) .. B-1, B-13
 IRQ B-6

K

kernel change 2-11
 kernel header files 2-12
 kernel memory B-10

L

libraries 2-9
 debug libraries 2-10, 4-1
 release libraries 4-1
 license key setting . 3-4, 3-6, 3-7, B-12,
 B-13, C-7
 Load on Boot B-5
 loading the driver
 Linux 2-11, 3-4
 Windows 3-5, B-5, B-6
 log buttons A-6
 log file 4-2, A-1, A-4, A-7

M

main folder 2-4
 makefile 2-4
 Manual/Boot Load B-5
 menu options (logs) A-3
 microcode 2-8
 mute B-12

O

OEM products 1-1
 OEM web pages 4-1
 operating systems 1-5
 overview
 files 2-4
 folders 2-4
 of chapters 1-2
 of DVSConf B-4
 of DVSInfo (Linux) A-2
 of DVSInfo (Windows) A-5
 SDK 2-2

P

PCI interface settings B-8
 PCI interface upgrade C-15
 SDStationOEM II C-15
 shell output C-15
 PCI video board 1-1, 2-10, B-1
 buffer .. B-11, B-13, C-2, C-6, C-8,
 C-12
 configuration ... A-1, A-4, A-7, B-11
 driver 2-2, 2-7, 2-8, 2-10, B-1
 hardware information B-14, C-15
 test B-1, B-13
 version information B-7, C-15
 pipe mode C-2, C-12
 post-compile 2-5, 2-7, 2-11, 2-12, 3-3
 restrictions 2-12
 postroll B-13
 preroll B-13
 project file 2-4
 prompt A-3, C-1, C-2

R

RAM recorder C-1, C-2
 raster definition file 2-8
 customization 2-13
 real time operation B-10
 record B-14
 release driver 2-10
 release libraries 4-1
 requirements 1-5
 Linux 1-5
 operating system 1-5
 resource script 2-4
 RS-422 2-18, C-14

S

script files 2-7, 2-11, 3-3, 3-4
 SCSIVIDEO_CMD C-3
 SDI 4-4, B-12, C-7
 SDK 2-2
 contents 2-2
 debugging 4-1
 files 2-4
 folders 2-4
 installation (Linux) 3-2
 installation (Windows) 3-5
 main folder 2-4
 update 3-8
 version information 2-17

1

2

3

4

A

B

C

I

- version numbering 2-4
- SDStationOEM II C-15
- Set System Pages B-10
- setting
 - analog video B-12, C-4
 - confirmation B-2
 - digital video B-12, C-9
 - license keys 3-4, 3-6, 3-7, B-12, C-7
 - VTR B-12
- slider B-13
- start play-out B-14, C-13
- starting DVSCnf B-2
- starting DVSInfo
 - Linux A-2
 - Windows A-5
- stop play-out B-14, C-14
- storage mode B-12, C-9
- streamer mode 4-4
- svram
 - analog C-4
 - audioinput C-5
 - black C-5
 - colorbar C-5
 - debugprint 2-10, C-6
 - display ram C-6
 - goto C-6
 - info C-6
 - inputport C-7
 - iomode C-7
 - licence C-7
 - live C-8
 - load C-8
 - mode C-9
 - outduringrec C-10
 - outputport C-11
 - pipe C-2, C-12
 - record ram C-12
 - save C-12
 - speed C-13
 - stop C-14
 - sync C-14
 - version C-15
- svram program C-1
 - command syntax C-2
 - help C-3
 - operation modes C-2
 - pipe mode C-2, C-12
 - RAM recorder C-1, C-2
 - source 2-18

- sync settings B-12, C-4, C-14

T

- tab
 - 'Card' B-11
 - 'Driver' B-4
 - 'Server' B-9
 - 'Settings' B-8
- target group 1-2
- test picture display B-13
- thread functions 2-17
- timeline B-13
- tools 2-2
- typographical conventions 1-3

U

- unloading the driver
 - Linux 2-11, 3-4
 - Windows 3-7, B-6
- updating the driver
 - Linux 3-8
 - Windows 3-6, B-7
- updating the SDK 3-8

V

- version information 2-17, B-7
- version numbering 2-4
- video board 2-10, B-1
 - buffer .. B-11, B-13, C-2, C-6, C-8, C-12
 - configuration .. A-1, A-4, A-7, B-11
 - driver 2-2, 2-10, B-1
 - hardware information B-14, C-15
 - test B-1, B-13
 - version information B-7, C-15
- video C library 2-14, 2-18, 2-19
- video raster 2-8, B-12
 - customized rasters 2-13
 - information B-14, C-6
- video setting B-12, C-4, C-9
- VTR B-13, C-14
 - setting B-12

W

- Windows Device Manager 3-6
- wordclock B-12
- workspace file 2-4