DVS Digital Video Systems GmbH

DVS SDK - DVS Software Development Kit

Reference Guide

DVS Digital Video Systems GmbH
Version 4.1.1 for the DVS SDK 4.1.1.0 or higher

Table of Contents

Main Page iv
Introduction iv
Target Group iv
Conventions Used in this Reference Guide iv
General Information v
What's New in the DVS SDK 4.<x> v
Supported DVS Video Board Products v
Supported Video Rasters vi
Working under Microsoft Windows vi
What's New in this Reference Guide vi
Module Index viii
Data Structure Index ix
Module Documentation 1
API – Basic Functions 1
API – Audio Functions 10
API – Video Functions 15
API – FIFO API 34
API – Direct API 61
API – Jack API 68
API – The sv_option() Functions 74
API – The sv_query() Function 77
API – Control Functions 78
API – Status Functions 85
API – RS-422 High-level API (Master) 90
API – RS-422 High-level API (Slave) 97
API – RS-422 Low-level API 98
API – Timecode 100
API – GPI Functionality 111
API – Proxy Capture 113
API – Hardware 117
API – Tracing 120
API – Storage Functions 121
Obsolete Defines and Functions 132
Info – Bit Formats 135
Info – Pixel Formats 137
Info – Audio Formats 138
Info – Storage Formats 139
Info – Error Codes 140
Example Projects Overview 147
Example – dpxio 149
Data Structure Documentation 153
sv_direct_bufferinfo 153
sv_direct_info 154
sv_direct_timecode 155
sv_fifo_ancbuffer 156
sv_fifo_buffer 157
sv_fifo_bufferinfo 159
sv_fifo_configinfo 160

sv_fifo_info 161
Index 162

DVS SDK Main Page

Introduction

This document provides a reference to all commands, defines, functions, and structures of the software development kit (SDK) by DVS. Additional information, for example, about the bit or audio formats, are provided in this reference guide as well.

The DVS SDK can be used together with the video board 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 video board 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 video board 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 video board products, meaning your code can be used with other DVS PCI video boards as well.

Target Group

To use this guide and the DVS 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.

Furthermore, you should know how to work with the DVS video device at hand as well as how to handle its driver.

Conventions Used in this Reference Guide

The following typographical conventions will be used in this documentation:

- Texts preceded by this symbol are parts of a list, first level as well as subordinated levels.

italic	Functions, parameter names (variables) or structures (structs).
`typewriter`	Defines, values, code examples, or commands (e.g. in your code).
`typewriter italic`	Programs, directories or directory structures, or files.

`<xxx>` is a place holder. If it is used, for example, with an option call or flag, it indicates a group of at least two of these calls/flags.

`<a>..<b>` indicates a value range from value `<a>` to value `<b>`.

General Information

This section contains some general information about the DVS SDK and this reference guide.

Note:

Most structures and parameter defines are documented in the source code of the DVS SDK directly. For further information about a structure or parameter define not described in this reference guide please refer to its comments in the respective header file of the DVS SDK.

For any additional information about the DVS SDK, for example, about its installation, the general driver handling or general information about debugging, please consult the "DVS Software Development Kit" user guide as well as any other guide or manual delivered with the DVS video board product.

What's New in the DVS SDK 4.<x>

The following details the most important features implemented in the DVS SDK 4.<x> as well as the decisive differences compared to its predecessor version 3.<x>:

New Features:

- Processing pipeline for newer DVS video boards such as Atomix for up- and downscaling with filtering, CSC, 1D LUT, and 3D LUT (see the reference guide to the DVS Render API Extension).
- Support for 3.0 Gbit/s SDI.
- Support of Mac OS X.
- New DVS control panel (currently Mac OS X only).

Differences to Version 3.<x>:

- Different license model for newer DVS video boards such as Atomix (see the functions *sv_licence()* and *sv_licencebit2string()*).
- Windows: Separate setup information files (`*.inf`) for different DVS video board products.
- Windows and Linux: Separate default raster list files (`*.ref`) for different DVS video board products.

Major Changes:

- Function *sv_fifo_status()* will now return the exact FIFO ring buffer size.
- Ceased support for SDStationOEM and SDStationOEM II.

Supported DVS Video Board Products

The following DVS video board products are supported by the DVS SDK 4.<x>. They are listed in the order of their release dates (newest at the bottom):

DVS Video Board Product	**Serial Number** (first two digits)
Centaurus II	PCI-X: 20 PCIe: 23
Centaurus II LT*	24
Atomix	27
Atomix LT*	5 BNC: 41 4 BNC and D-Sub: 42

* The LT versions are in most respects identical to their respective counterpart without 'LT'.

Therefore, in this reference guide they will be subsumed under the name of their counterpart, meaning e.g. whenever 'Atomix' is mentioned 'Atomix LT' is meant as well.

Whether a functionality is available for certain DVS video boards only, will be detailed in the descriptions of the defines/functions in this reference guide. For further details about the availability of a DVS SDK tested and released for a certain video board or firmware please refer to the *readme.txt* or *changelog.txt* of the DVS SDK.

Supported Video Rasters

For a list of the video rasters supported by your DVS video board product please run the command `svram mode help` or `svram guiinfo init` (extended information) at a command line.

Working under Microsoft Windows

The following provides some details that you should note when operating a DVS video board under Microsoft Windows operating systems.

Handling of Sleep and Hibernation Modes

The DVS SDK provides drivers that support the recovery from sleep and hibernation modes on Windows.

When a system enters one of the two mentioned power saving modes, the board will be automatically closed and all opened handles will be void. Thus, after wake-up you cannot just proceed and use the DVS video board immediately again. In such a case, when trying to use a DVS API function without calling *sv_open()* or *sv_openex()* anew, you will get the error code 219 (`SV_ERROR_SLEEPING`).

To avoid this error as well as any other unusual behavior it is recommended to open and configure the board once again before performing an operation with the DVS video board.

What's New in this Reference Guide

The following details the major additions and changes that were made to this reference guide in its latest revisions:

New in Version 4.1.1:

- Added the Direct API (see chapter API – Direct API).
- Enabled the hardware watchdog (`SV_OPTION_HWWATCHDOG_<xxx>`) on Atomix LT.
- Enabled the extended ANC data handling (`SV_ANCCOMPLETE_ON`) on Atomix.
- Updated documentation of `SV_OPTION_AUDIOAESROUTING`.
- Added the defines `SV_OPTION_ASSIGN_LTCA` (made `SV_OPTION_ASSIGN_LTC` obsolete), `SV_OPTION_AUDIOAESSOURCE`, `SV_AUDIOAESROUTING_8_0`, `SV_AUDIOAESROUTING_0_8`, `SV_OPTION_AUDIONOFADING`, `SV_OPTION_DVI_OUTPUT`, and `SV_QUERY_LTCAVAILABLE`.
- Added new error codes and example programs.

New in Version 4.0.1:

Completely revised: For example, the no longer supported DVS video boards have been removed as well as most obsolete functions and defines.

Other major changes:

- Updated documentation of *sv_licence()*, *sv_rs422_open()*, `SV_OPTION_RS422A`, `SV_FIFO_FLAG_AUDIOINTERLEAVED`, and *sv_fifo_info*.

- Added the functions *sv_licenceinfo()* and *sv_licencebit2string()*.
- Added the defines SV_FIFO_LUT_TYPE_1D_RGBA_4K, SV_OPTION_AUDIODRIFT_ADJUST, SV_OPTION_IOMODE_AUTODETECT, SV_OPTION_IOSPEED, SV_OPTION_PULLDOWN_STARTLTC, SV_OPTION_PULLDOWN_STARTPHASE, SV_OPTION_PULLDOWN_STARTVTRTC, SV_OPTION_SYNCSELECT, SV_QUERY_IOCHANNELS, SV_QUERY_IOLINKS_INPUT, SV_QUERY_IOLINKS_OUTPUT, SV_QUERY_IOMODEINERROR, SV_QUERY_IOSPEED, SV_QUERY_IOSPEED_SDI [ABCD], SV_QUERY_IOLINK_MAPPING, SV_QUERY_SMPTE352, and SV_SWITCH_TOLERANCE_DETECT_CYCLES(x).

New in Version 3.4.1:

- Amended descriptions of the defines SV_MASTER_EDITFIELD_START and -_END.
- Changed parameter *dma* in function *sv_fifo_init()*.
- Added section for specific information about Windows operating systems (see Working under Microsoft Windows).
- Added error code SV_ERROR_SLEEPING.

DVS SDK Module Index

DVS SDK Modules

Here is a list of all modules:

API – Basic Functions 1
API – Audio Functions 10
API – Video Functions 15
API – FIFO API 34
API – Direct API 61
API – Jack API 68
API – The sv_option() Functions 74
API – The sv_query() Function 77
API – Control Functions 78
API – Status Functions 85
API – RS-422 High-level API (Master) 90
API – RS-422 High-level API (Slave) 97
API – RS-422 Low-level API 98
API – Timecode 100
API – GPI Functionality 111
API – Proxy Capture 113
API – Hardware 117
API – Tracing 120
API – Storage Functions 121
Obsolete Defines and Functions 132
Info – Bit Formats 135
Info – Pixel Formats 137
Info – Audio Formats 138
Info – Storage Formats 139
Info – Error Codes 140
Example Projects Overview 147
Example – dpxio 149

DVS SDK Data Structure Index

DVS SDK Data Structures

Here are the data structures with brief descriptions:

sv_direct_bufferinfo 153
sv_direct_info 154
sv_direct_timecode 155
sv_fifo_ancbuffer 156
sv_fifo_buffer 157
sv_fifo_bufferinfo 159
sv_fifo_configinfo 160
sv_fifo_info 161

DVS SDK Module Documentation

API – Basic Functions

Detailed Description

This chapter describes basic functions for the DVS video device, for example, to open and close the connection to the device or query its status.

Defines

- #define SV_OPTION_DEBUG
- #define SV_OPTION_NOP

Functions

- int sv_close (sv_handle *sv)
- int sv_currenttime (sv_handle *sv, int type, int *ptick, uint32 *pclockhigh, uint32 *pclocklow)
- int sv_debugprint (sv_handle *sv, char *buffer, int buffersize, int *pbuffercount)
- void sv_errorprint (sv_handle *sv, int errorcode)
- char * sv_errorstring (sv_handle *sv, int errorcode)
- char * sv_geterrortext (int errorcode)
- int sv_getlicence (sv_handle *sv, int *ptype, int *phwver, int *pserial, int *pver, int *pram, int *pdisk, int *pflags, int dim, uint *pkeys)
- int sv_licence (sv_handle *sv, int knum, char *code)
- char * sv_licencebit2string (sv_handle *sv, int bitno)
- int sv_licenceinfo (sv_handle *sv, int *pdevtype, int *pserial, int *pexpire, unsigned char *pfeatures, int featuresize, unsigned char *pkeys, int keysize)
- sv_handle * sv_open (char *setup)
- int sv_openex (sv_handle **psv, char *setup, int openprogram, int opentype, int timeout, int spare)
- int sv_usleep (sv_handle *sv, int usec)
- int sv_version_certify (sv_handle *sv, char *path, int *required_sw, int *required_fw, int *bcertified, void *spare)
- int sv_version_check (sv_handle *sv, int major, int minor, int patch, int fix)
- int sv_version_check_firmware (sv_handle *sv, char *current, int current_size, char *recommended, int recommended_size)
- int sv_version_verify (sv_handle *sv, unsigned int neededlicence, char *errorstring, int errorstringsize)

Define Documentation

#define SV_OPTION_DEBUG

Debug define. For DVS internal use only.

#define SV_OPTION_NOP

No Operation. For DVS internal use only.

Function Documentation

int sv_close (sv_handle * *sv*)

This function closes the connection to the DVS video device. After this command the *sv_handle* structure will be invalid.

Parameters:

sv – Handle returned from the function *sv_open()*.

Returns:

If the function succeeds, it returns `SV_OK`. Otherwise it will return the error code `SV_ERROR_<xxx>`.

Note:

The function *sv_close()* should be the last function called. It will free the video device for other users or usages.

Example:

```
void example_closedevice(sv_handle * sv)
{
  int res = sv_close(sv);
  if(res != SV_OK) {
    printf("Error: sv_close(sv) failed = %d '%s'", res, sv_geterrortext(res));
  }
}
```

int sv_currenttime (sv_handle * *sv*, int *type*, int * *ptick*, uint32 * *pclockhigh*, uint32 * *pclocklow*)

This function returns various driver tick (timestamp) values.

Parameters:

sv – Handle returned from the function *sv_open()*.

type – Defines the time that should be returned. See list below.

ptick – Returns the tick value.

pclockhigh – Returns the upper 32 bits of the clock.

pclocklow – Returns the lower 32 bits of the clock.

Parameters for *type*:

- `SV_CURRENTTIME_CURRENT` – Returns the current clock and tick.
- `SV_CURRENTTIME_VSYNC_DISPLAY` – Returns the current display tick and the last display vertical sync clock.
- `SV_CURRENTTIME_VSYNC_RECORD` – Returns the current record tick and the last record vertical sync clock.

- SV_CURRENTTIME_FRAME_DISPLAY – Returns the current display tick and the last display frame clock.
- SV_CURRENTTIME_FRAME_RECORD – Returns the current record tick and the last record frame clock.

Returns:

If the function succeeds, it returns SV_OK. Otherwise it will return the error code SV_ERROR_<xxx>.

int sv_debugprint (sv_handle * *sv*, char * *buffer*, int *buffersize*, int * *pbuffercount*)

This function returns debug information logged by the DVS video device driver.

Parameters:

sv – Handle returned from the function *sv_open()*.

buffer – Buffer that will contain the debug information.

buffersize – Size of the buffer *buffer*.

pbuffercount – Actual size of the buffer on return (used size).

Returns:

If the function succeeds, it returns SV_OK. Otherwise it will return the error code SV_ERROR_<xxx>.

void sv_errorprint (sv_handle * *sv*, int *errorcode*)

This function gives out an error message according to *errorcode* at the output (stderr).

Parameters:

sv – Handle returned from the function *sv_open()*.

errorcode – The code of the error that should be given out.

char* sv_errorstring (sv_handle * *sv*, int *errorcode*)

This function returns a string that describes an error code.

Parameters:

sv – Handle returned from the function *sv_open()*.

errorcode – Code of the error that should be described.

Returns:

A string that describes the error code.

char* sv_geterrortext (int *errorcode*)

Same as the function *sv_errorstring()*.

Parameters:

errorcode – Code of the error that should be described.

Returns:

A string that describes the error code.

int sv_getlicence (sv_handle * *sv*, int * *ptype*, int * *phwver*, int * *pserial*, int * *pver*, int * *pram*, int * *pdisk*, int * *pflags*, int *dim*, uint * *pkeys*)

This function reads the license and licensed features from the DVS video device.

Parameters:

sv – Handle returned from the function *sv_open()*.

ptype – Device type, not the same as `SV_DEVTYPE_<xxx>` (see `SV_QUERY_DEVTYPE`). For possible returns see list below.

phwver – Device hardware version.

pserial – Device serial number.

pver – Driver version.

pram – Licensed size of RAM.

pdisk – Licensed size of hard disks.

pflags – Licensed mask.

dim – Size of the array *pkeys*.

pkeys – Returns the set licenses.

Return Values for *ptype*:

- `25` – Centaurus II.

Returns:

If the function succeeds, it returns `SV_OK`. Otherwise it will return the error code `SV_ERROR_<xxx>`.

Note:

This function is available for Centaurus II only. For other DVS video devices use the function *sv_licenceinfo()*.

The serial numbers of the supported DVS video board products (first digits) are listed in the section Supported DVS Video Board Products.

int sv_licence (sv_handle * *sv*, int *knum*, char * *code*)

This function programs the license of the DVS video device. Please note that newer DVS video devices require different values for *knum*.

Parameters:

sv – Handle returned from the function *sv_open()*.

knum – Number of the license key that will hold the license. Possible values for Centaurus II range from one to three (`1..3`). For all other DVS video devices use the values four (`4`) or five (`5`).

code – String pointer to the license key to be programmed.

Returns:

If the function succeeds, it returns `SV_OK`. Otherwise it will return the error code `SV_ERROR_<xxx>`.

Note:

The passed license key string can contain comments or special characters: Lines starting with # above or below the actual license code will be treated as comments and spaces or carriage returns will be ignored.

char* sv_licencebit2string (sv_handle * *sv*, int *bitno*)

This function decodes a given license bit to a readable description (string). The license bits can be obtained by using the function *sv_licenceinfo()*.

Parameters:

sv – Handle returned from the function *sv_open()*.

bitno – The license bit that should be decoded.

Returns:

If the function succeeds, it returns `SV_OK`. Otherwise it will return the error code `SV_ERROR_<xxx>`.

Note:

This function is available on newer DVS video devices such as Atomix only.

int sv_licenceinfo (sv_handle * *sv*, int * *pdevtype*, int * *pserial*, int * *pexpire*, unsigned char * *pfeatures*, int *featuresize*, unsigned char * *pkeys*, int *keysize*)

This function reads the license and licensed features from the DVS video device. It should be used on newer DVS video devices such as Atomix.

Parameters:

sv – Handle returned from the function *sv_open()*.

pdevtype – Device type `SV_DEVTYPE_<xxx>` (see `SV_QUERY_DEVTYPE`).

pserial – Device serial number.

pexpire – Date when a temporary license will expire (expiration date).

pfeatures – Array containing the license options activated by the license key.

featuresize – Size of the array *pfeatures*.

pkeys – Array containing the currently set license key.

keysize – Size of the array *pkeys*.

Returns:

If the function succeeds, it returns `SV_OK`. Otherwise it will return the error code `SV_ERROR_<xxx>`.

Note:

This function is available on newer DVS video devices such as Atomix only. For Centaurus II use the function *sv_getlicence()*.

The license options of the parameter *pfeatures* can be decoded using the function *sv_licencebit2string()*.

sv_handle* sv_open (char * *setup*)

This function opens a DVS video device.

Parameters:

setup – String that controls the opening of the device. For information about its syntax see below.

Syntax of *setup*:

Normally the syntax of the parameter *setup* is `"PCI,card:<n>[,channel:<m>]"` (e.g. `sv_open("PCI,card:0")`).

You may leave the *setup* string empty to globally open the very first DVS video board (i.e. `card:0`).

To open a particular board in an environment where more than one DVS video board is installed, the respective board index can be specified with the substring `card:<n>` (with `<n>` as the number of the board).

In a multi-channel environment (see the define `SV_OPTION_MULTICHANNEL` and the introduction to chapter API – Jack API) each pair of input/output channels can be associated with separate *sv_handle* pointers by specifying the substring `"channel:<m>"`. In case the channel substring is left out during the opening of the board (`SV_OPTION_MULTICHANNEL` is set), all jacks will be addressed at the same time with the resulting *sv_handle* pointer.

Returns:

This function returns an *sv_handle* pointer. It has to be passed to all other SV functions.

See also:

The function *sv_openex()*.

int sv_openex (sv_handle ** *psv*, char * *setup*, int *openprogram*, int *opentype*, int *timeout*, int *spare*)

This function opens a DVS video device similar to the function *sv_open()*. Additionally, it can open different ports of the device which will be useful when different processes cannot share the same *sv_handle* pointer.

Parameters:

psv – Returns an *sv_handle* pointer. It has to be passed to all other SV functions.

setup – String that controls the opening of the device. For further information about its syntax see the function *sv_open()*.

openprogram – Defines the opening program type. See list below.

opentype – Defines which port of the device to open. See list below.

timeout – Currently not available. Sets a timeout for a delayed opening.

spare – Reserved for future use. It has to be set to zero (`0`).

Parameters for *openprogram*:

- `SV_OPENPROGRAM_DEFAULT` – Program type not specified. This value is internally set when the function *sv_open()* is called.
- `SV_OPENPROGRAM_SVPROGRAM` – SV program.
- `SV_OPENPROGRAM_TESTPROGRAM` – DVS test program.
- `SV_OPENPROGRAM_DEMOPROGRAM` – Example program.
- `SV_OPENPROGRAM_VSERVER` – Obsolete.
- `SV_OPENPROGRAM_KERNEL` – Opened from another kernel device.
- `SV_OPENPROGRAM_OPENML` – OpenML driver.
- `SV_OPENPROGRAM_QUICKTIME` – QuickTime driver.
- `SV_OPENPROGRAM_APPLICATION` – Application.
- `SV_OPENPROGRAM_APPID(appid)` – Mask to set a 24-bit application ID.

Parameters for *opentype*:

- `SV_OPENTYPE_DEFAULT` – All ports.
- `SV_OPENTYPE_VOUTPUT` – Video output.
- `SV_OPENTYPE_AOUTPUT` – Audio output.
- `SV_OPENTYPE_OUTPUT` – Video and audio output.

- SV_OPENTYPE_VINPUT – Video input.
- SV_OPENTYPE_AINPUT – Audio input.
- SV_OPENTYPE_INPUT – Video and audio input.
- SV_OPENTYPE_RS422A – Serial port A or master port. For this you can also use the type SV_OPENTYPE_MASTER.
- SV_OPENTYPE_RS422B – Serial port B or slave port. For this you can also use the type SV_OPENTYPE_SLAVE.
- SV_OPENTYPE_MASTER – Same as SV_OPENTYPE_RS422A.
- SV_OPENTYPE_SLAVE – Same as SV_OPENTYPE_RS422B.
- SV_OPENTYPE_MASK_ONCE – Mask for ports that can only be opened once.
- SV_OPENTYPE_CAPTURE – Opens the capture port (to be used with the function *sv_capture()*).
- SV_OPENTYPE_WAITFORCLOSE – Waits for another program to close.
- SV_OPENTYPE_MASK_MULTIPLE – Mask for ports that can be opened multiple times.
- SV_OPENTYPE_MASK – Mask for all ports.
- SV_OPENTYPE_VALID – Mask for all valid ports.

Returns:

If the function succeeds, it returns SV_OK. Otherwise it will return the error code SV_ERROR_<xxx>.

Note:

As this function binds the *sv_handle* pointer to the callers process ID, you have to make sure to close the *sv_handle* using the function *sv_close()* from the same process. Otherwise the resource will stay blocked until the corresponding process gets finally terminated.

See also:

The function *sv_open()*.

int sv_usleep (sv_handle * *sv*, int *usec*)

This function delays an execution for the specified amount of microseconds.

Parameters:

sv – Handle returned from the function *sv_open()*.

usec – Sets the number of microseconds to sleep.

Returns:

SV_OK.

Note:

This function internally employs *Sleep()* under Windows and *usleep()* under all other platforms.

int sv_version_certify (sv_handle * *sv*, char * *path*, int * *required_sw*, int * *required_fw*, int * *bcertified*, void * *spare*)

This function checks if the driver and firmware versions match the actually installed versions. If there is any mismatch, the return code as well as the error string contain the error. This provides an easy way for an application to check that all parts of the DVS setup are up to date on the system. Note that even if the function fails, it might still be a functional setup. It is suggested to use it as a check that does not prohibit a starting completely, even if the function returns an error. A good way would be to inform the user, but to continue if he decides so.

Parameters:

sv – Handle returned from the function *sv_open()*.

path – Path to the version file.

required_sw – Pointer to the required software version. This can be zero (`0`).

required_fw – Pointer to the required firmware version. This can be zero (`0`).

bcertified – Pointer to the test result. This can be zero (`0`).

spare – Reserved for future use.

Returns:

The following returns are possible:

- `SV_OK` – Everything checks out and is okay.
- `SV_ERROR_FIRMWARE` – Firmware is not up to date.
- `SV_ERROR_DRIVER_MISMATCH` – Driver is not up to date.

int sv_version_check (sv_handle * *sv*, int *major*, int *minor*, int *patch*, int *fix*)

This function checks if the application version matches the version numbers of library and driver. It is mainly used for diagnostic purposes. The version numbering is `<major>.<minor>.<patch>.<fix>`.

Parameters:

sv – Handle returned from the function *sv_open()*.

major – Version major number (`DVS_VERSION_MAJOR`).

minor – Version minor number (`DVS_VERSION_MINOR`).

patch – Version patch number (`DVS_VERSION_PATCH`).

fix – Version fix number (`DVS_VERSION_FIX`).

Returns:

If all versions match, it returns `SV_OK`. Otherwise it will return the error code `SV_ERROR_<xxx>`.

int sv_version_check_firmware (sv_handle * *sv*, char * *current*, int *current_size*, char * *recommended*, int *recommended_size*)

This function checks if the firmware version of the board matches the recommended version (revision). It also returns strings of the recommended and current version.

Parameters:

sv – Handle returned from the function *sv_open()*.

current – Returns the current firmware revision.

current_size – Size of the char array for *current*.

recommended – Returns the recommended firmware revision.

recommended_size – Size of the char array for *recommended*.

Returns:

If the current firmware matches the recommended firmware version, it returns `SV_OK`. Otherwise it will return the error code `SV_ERROR_<xxx>`.

int sv_version_verify (sv_handle * *sv*, unsigned int *neededlicence*, char * *errorstring*, int *errorstringsize*)

This function checks if driver, DVSOEM library and firmware versions match. If there is any mismatch, the return code as well as the error string contain the error. This provides an easy way for an application to check that all parts of the DVS setup on the system are in a valid state. Note that even if the function fails, it might still be a functional setup, only that this was not the correct setup for the SDK at the time it was compiled. It is suggested to use it as a check that does not prohibit a starting completely, even if the function returns an error. A good way would be to inform the user, but to continue if he decides so.

The function performs its checks in the following priority:

1. License mismatch (to disable this error set *neededlicence* to zero (`0`)).
2. Driver and/or DVSOEM version mismatch.
3. Firmware version mismatch.

Parameters:

sv – Handle returned from the function *sv_open()*.

neededlicence – Mask of the license bits that are needed for the operation.

errorstring – Returns an error message if the versions are wrong.

errorstringsize – Size of the char array for *errorstring*.

Returns:

The following returns are possible:

- `SV_OK` – Everything checks out and is okay.
- `SV_ERROR_NOLICENCE` – License bit check failed.
- `SV_ERROR_FIRMWARE` – Firmware mismatch detected.
- `SV_ERROR_DRIVER_MISMATCH` – Mismatch of DVSOEM library and/or driver version detected.

API – Audio Functions

Detailed Description

This chapter details various functions to control audio related features of the DVS video device.

Defines

- #define SV_OPTION_AUDIOAESROUTING
- #define SV_OPTION_AUDIOAESSOURCE
- #define SV_OPTION_AUDIOANALOGOUT
- #define SV_OPTION_AUDIOBITS
- #define SV_OPTION_AUDIOCHANNELS
- #define SV_OPTION_AUDIODRIFT_ADJUST
- #define SV_OPTION_AUDIOFREQ
- #define SV_OPTION_AUDIOINPUT
- #define SV_OPTION_AUDIOMAXAIV
- #define SV_OPTION_AUDIOMUTE
- #define SV_OPTION_AUDIONOFADING
- #define SV_OPTION_WORDCLOCK
- #define SV_QUERY_AUDIO_AESCHANNELS
- #define SV_QUERY_AUDIO_AIVCHANNELS
- #define SV_QUERY_AUDIO_MAXCHANNELS
- #define SV_QUERY_AUDIOBITS
- #define SV_QUERY_AUDIOCHANNELS
- #define SV_QUERY_AUDIOFREQ
- #define SV_QUERY_AUDIOINERROR
- #define SV_QUERY_AUDIOINPUT
- #define SV_QUERY_AUDIOMUTE
- #define SV_QUERY_WORDCLOCK

Define Documentation

#define SV_OPTION_AUDIOAESROUTING

This define configures the routing of the physical AES channels (digital audio) to logical audio channels in the memory layout. It is especially useful when the DVS video board is running in a multi-channel configuration.

The following routings can be selected:

- `SV_AUDIOAESROUTING_DEFAULT` – Default routing. The defaults are `SV_AUDIOAESROUTING_8_8` when the hardware supports multi-channel operation and `SV_AUDIOAESROUTING_16_0` when the hardware supports a single I/O channel only. On Atomix LT 4 BNC the default is `SV_AUDIOAESROUTING_8_0`.
- `SV_AUDIOAESROUTING_16_0` – I/O channel 0: Physical channels 1 to 16 are connected to the logical channels 1 to 16. I/O channel 1: No channels connected.

- SV_AUDIOAESROUTING_8_8 – I/O channel 0: Physical channels 1 to 8 are connected to the logical channels 1 to 8. I/O channel 1: Physical channels 9 to 16 are connected to the logical channels 1 to 8.
- SV_AUDIOAESROUTING_4_4 – I/O channel 0: Physical channels 1 to 4 and 9 to 12 are connected to the logical channels 1 to 8. On Atomix LT 4 BNC the physical channels 9 to 12 are connected to the logical channels 1 to 4. I/O channel 1: Physical channels 5 to 8 and 13 to 16 are connected to the logical channels 1 to 8. On Atomix LT 4 BNC the physical channels 13 to 16 are connected to the logical channels 1 to 4.
- SV_AUDIOAESROUTING_8_0 – Only available on Atomix LT 4 BNC. I/O channel 0: Physical channels 9 to 16 are connected to the logical channels 1 to 8. I/O channel 1: No channels connected.
- SV_AUDIOAESROUTING_0_8 – Only available on Atomix LT 4 BNC. I/O channel 0: No channels connected. I/O channel 1: Physical channels 9 to 16 are connected to the logical channels 1 to 8.

The following table may help clarifying the distribution of the physical channels to the I/O channels. In brackets the logical channels are indicated:

	Physical Channels 1 to 8	**Physical Channels 9 to 16**
SV_AUDIOAESROUTING_16_0	I/O channel 0 (1 to 8)	I/O channel 0 (9 to 16)
SV_AUDIOAESROUTING_8_8	I/O channel 0 (1 to 8)	I/O channel 1 (1 to 8)
SV_AUDIOAESROUTING_4_4	I/O channel 0 (1 to 4) I/O channel 1 (1 to 4) n/a on Atomix LT 4 BNC	I/O channel 0 (5 to 8) I/O channel 1 (5 to 8) Atomix LT 4 BNC: I/O channel 0 (1 to 4) I/O channel 1 (1 to 4)
SV_AUDIOAESROUTING_8_0	n/a on Atomix LT 4 BNC	I/O channel 0 (1 to 8)
SV_AUDIOAESROUTING_0_8	n/a on Atomix LT 4 BNC	I/O channel 1 (1 to 8)

See also:

The define SV_OPTION_MULTICHANNEL.

#define SV_OPTION_AUDIOAESSOURCE

This define configures the AES input on the Atomix Breakout Box 4 x SDI I/O+A that should be used as the source. The following sources can be selected:

- SV_AUDIOAESSOURCE_DEFAULT – Default input source. Default is SV_AUDIOAESSOURCE_BACK.
- SV_AUDIOAESSOURCE_BACK – Rear connectors are used.
- SV_AUDIOAESSOURCE_FRONT – Front connectors are used.

#define SV_OPTION_AUDIOANALOGOUT

This define selects the audio channels that should be sent to the analog output. You can select two mono channels for the analog output. The value represents the mono channels, i.e. channels *n* and *n* + 1 are selected by applying a value of 0x(n-1)n.

Example:

For channels 1 and 2 use the value 0x01 and for channels 3 and 4 use the value 0x23.

#define SV_OPTION_AUDIOBITS

This define selects the audio bit depth. It describes the audio memory depth on the DVS video board:

- `32` – 32 bit audio.

#define SV_OPTION_AUDIOCHANNELS

This define sets the number of stereo audio channel pairs.

#define SV_OPTION_AUDIODRIFT_ADJUST

This define activates an audio recording mode which enables you to record from unsynchronized audio sources (when audio is asynchronous to the video signal). In this mode the amount of samples that are recorded per frame is not fix. Thus it can compensate for any existing audio drift.

#define SV_OPTION_AUDIOFREQ

This define sets the audio frequency:

- `48000` – Default is 48000 Hz.

Note:

On Atomix this setting adjusts the audio frequency on all input and output jacks globally.

#define SV_OPTION_AUDIOINPUT

This define selects the audio input:

- `SV_AUDIOINPUT_AIV` – Audio embedded in the video signal will be used.
- `SV_AUDIOINPUT_AESEBU` – Digital audio (AES/EBU) will be used.

Note:

This operation takes about two frames to take effect.

#define SV_OPTION_AUDIOMAXAIV

This define sets the maximum number of audio channels that will be embedded in the SDI video stream. This may be necessary because some VTRs have difficulties when handling embedded audio that provides more audio channels than they are capable of (DVS has experienced this with older Digibetas).

The following audio outputs can be selected:

- `0` – No embedded audio.
- `4` – Four mono channels, one group.
- `8` – Eight mono channels, two groups.
- `12` – Twelve mono channels, three groups.
- `16` – 16 mono channels, four groups.

#define SV_OPTION_AUDIOMUTE

When this define is set, the audio output will be muted.

#define SV_OPTION_AUDIONOFADING

The default of this define is zero (0), meaning on most DVS video boards audio will be faded in and out to avoid clicks at the beginning and end of an audio play-out. When this define is set to one (1), the audio output will not be faded.

#define SV_OPTION_WORDCLOCK

This define turns the audio wordclock output on or off:

- SV_WORDCLOCK_OFF – Turns the wordclock output off.
- SV_WORDCLOCK_ON – Turns the wordclock output on.

#define SV_QUERY_AUDIO_AESCHANNELS

This define returns a bit mask of all detected AES/EBU channels (digital audio). Each bit will be a mono audio channel.

#define SV_QUERY_AUDIO_AIVCHANNELS

This define returns a bit mask of all detected AIV channels (audio embedded in video). Each bit will be a mono audio channel.

#define SV_QUERY_AUDIO_MAXCHANNELS

This define returns the maximum number of stereo audio channel pairs that can be configured with the define SV_OPTION_AUDIOCHANNELS.

#define SV_QUERY_AUDIOBITS

This define returns the currently set audio bit depth. See SV_OPTION_AUDIOBITS.

#define SV_QUERY_AUDIOCHANNELS

This define returns the number of the currently configured stereo audio channel pairs. See SV_OPTION_AUDIOCHANNELS.

#define SV_QUERY_AUDIOFREQ

This define returns the currently set audio frequency. See SV_OPTION_AUDIOFREQ.

#define SV_QUERY_AUDIOINERROR

This define returns the audio input error. In case no error is detected, it returns SV_OK.

#define SV_QUERY_AUDIOINPUT

This define returns the currently selected way to input audio. See SV_OPTION_AUDIOINPUT.

#define SV_QUERY_AUDIOMUTE

If audio is muted, this define returns TRUE. See SV_OPTION_AUDIOMUTE.

#define SV_QUERY_WORDCLOCK

This define returns the current setting of the audio wordclock output. See SV_OPTION_WORDCLOCK.

API – Video Functions

Detailed Description

In this chapter you can find various functions to control video related features of the DVS video device.

Defines

- #define SV_OPTION_ALPHAGAIN
- #define SV_OPTION_ALPHAMIXER
- #define SV_OPTION_ALPHAOFFSET
- #define SV_OPTION_DETECTION_NO4K
- #define SV_OPTION_DETECTION_TOLERANCE
- #define SV_OPTION_DISABLESWITCHINGLINE
- #define SV_OPTION_DVI_OUTPUT
- #define SV_OPTION_FIELD_DOMINANCE
- #define SV_OPTION_HDELAY
- #define SV_OPTION_HWWATCHDOG_ACTION
- #define SV_OPTION_HWWATCHDOG_REFRESH
- #define SV_OPTION_HWWATCHDOG_RELAY_DELAY
- #define SV_OPTION_HWWATCHDOG_TIMEOUT
- #define SV_OPTION_HWWATCHDOG_TRIGGER
- #define SV_OPTION_INPUTFILTER
- #define SV_OPTION_INPUTPORT
- #define SV_OPTION_IOMODE
- #define SV_OPTION_IOMODE_AUTODETECT
- #define SV_OPTION_IOSPEED
- #define SV_OPTION_MAINOUTPUT
- #define SV_OPTION_OUTPUTFILTER
- #define SV_OPTION_OUTPUTPORT
- #define SV_OPTION_PULLDOWN_STARTLTC
- #define SV_OPTION_PULLDOWN_STARTPHASE
- #define SV_OPTION_PULLDOWN_STARTVTRTC
- #define SV_OPTION_SWITCH_TOLERANCE
- #define SV_OPTION_SYNCMODE
- #define SV_OPTION_SYNCOUT
- #define SV_OPTION_SYNCOUTDELAY
- #define SV_OPTION_SYNCOUTVDELAY
- #define SV_OPTION_SYNCSELECT
- #define SV_OPTION_VDELAY
- #define SV_OPTION_VIDEOMODE
- #define SV_QUERY_CARRIER
- #define SV_QUERY_DISPLAY_LINENR
- #define SV_QUERY_GENLOCK
- #define SV_QUERY_HDELAY

- #define SV_QUERY_INPUTFILTER
- #define SV_QUERY_INPUTPORT
- #define SV_QUERY_INPUTRASTER
- #define SV_QUERY_INPUTRASTER_GENLOCK
- #define SV_QUERY_INPUTRASTER_GENLOCK_TYPE
- #define SV_QUERY_INPUTRASTER_SDIA
- #define SV_QUERY_INPUTRASTER_SDIB
- #define SV_QUERY_INPUTRASTER_SDIC
- #define SV_QUERY_IOCHANNELS
- #define SV_QUERY_IOLINK_MAPPING
- #define SV_QUERY_IOLINKS_INPUT
- #define SV_QUERY_IOLINKS_OUTPUT
- #define SV_QUERY_IOMODE
- #define SV_QUERY_IOMODEINERROR
- #define SV_QUERY_IOSPEED
- #define SV_QUERY_IOSPEED_SDIA
- #define SV_QUERY_IOSPEED_SDIB
- #define SV_QUERY_IOSPEED_SDIC
- #define SV_QUERY_IOSPEED_SDID
- #define SV_QUERY_MODE_AVAILABLE
- #define SV_QUERY_MODE_CURRENT
- #define SV_QUERY_OUTPUTFILTER
- #define SV_QUERY_OUTPUTPORT
- #define SV_QUERY_RASTER_DROPFRAME
- #define SV_QUERY_RASTER_FPS
- #define SV_QUERY_RASTER_INTERLACE
- #define SV_QUERY_RASTER_SEGMENTED
- #define SV_QUERY_RASTER_XSIZE
- #define SV_QUERY_RASTER_YSIZE
- #define SV_QUERY_RASTERID
- #define SV_QUERY_RECORD_LINENR
- #define SV_QUERY_SMPTE352
- #define SV_QUERY_STORAGE_XSIZE
- #define SV_QUERY_STORAGE_YSIZE
- #define SV_QUERY_SYNCMODE
- #define SV_QUERY_SYNCOUT
- #define SV_QUERY_SYNCOUTDELAY
- #define SV_QUERY_SYNCOUTVDELAY
- #define SV_QUERY_SYNCSTATE
- #define SV_QUERY_TICK
- #define SV_QUERY_VDELAY
- #define SV_QUERY_VIDEOINERROR

Functions

- int sv_pulldown (sv_handle *sv, int cmd, int param)
- int sv_sync (sv_handle *sv, int sync)
- int sv_sync_output (sv_handle *sv, int syncout)

- int sv_videomode (sv_handle *sv, int mode)
- int sv_vsyncwait (sv_handle *sv, int operation, sv_vsyncwait_info *pinfo)

Define Documentation

#define SV_OPTION_ALPHAGAIN

Once the option SV_OPTION_ALPHAMIXER is activated, this define can be applied to specify the alpha gain. You can use it together with the define SV_OPTION_ALPHAOFFSET to convert alpha value ranges from any existing source value range (e.g. 10 bit, range 64..960) into the full value range expected by the DVS mixer hardware.

The mixer hardware always expects a value range from zero to 65535 (0..65535). The presentation of the alpha gain value is fixed point float and its default value is 0x10000 (i.e. 1.00000).

With SV_OPTION_ALPHAGAIN alone you can convert source alpha data available in full range to the 16-bit range that is expected by the DVS video hardware (e.g. convert 8-bit full range alpha data to 16-bit full range).

In case the source alpha range does not start with zero (0), you have to specify an offset in addition with the define SV_OPTION_ALPHAOFFSET. If set correctly, it will set the calculated value range back to zero (0).

The following source code shows how the alpha gain and offset values are calculated:

```
  int main(int argc, char * argv[])
  {
    int min = atoi(argv[1]);
    int max = atoi(argv[2]);
    int width = atoi(argv[3]);

    int min16 = min * (1 << (16 - width));
    int max16 = max * (1 << (16 - width));

    int gain   = (int) (((float) 65536 / ((max16 + 1) - min16)) * 65536);
    int offset = (int) (-min16 * ((float) 65536 / ((max16 + 1) - min16)));

    printf("min:     0x%05x\n", min16);
    printf("max:     0x%05x\n", max16);

    printf("gain:    hex:0x%05x dec:%5d float:%1.5f\n", gain, gain, (float) gain /
65536);
    printf("offset:  hex:0x%05x dec:%5d float:%1.5f (shift:%1.5f)\n", offset &
0xfffff, offset, (float) offset / 65536, (float) offset / (1 << (16 - width)));
  }
```

Below you can find some examples of alpha gain and offset values for several sample value ranges of source alpha data. With them the sample value ranges can be set to the full 16-bit value range:

Examples for full range source data:

- 8 bit, range 0..255 – gain: 0x100ff (1.00389), offset: 0
- 10 bit, range 0..1023 – gain: 0x1003f (1.00096), offset: 0
- 12 bit, range 0..4095 – gain: 0x1000f (1.00023), offset: 0
- 16 bit, range 0..65535 – gain: 0x10000 (1.00000), offset: 0

Examples for headroom range source data:

- 8 bit, range 1..254 – gain: 0x10308 (1.01184), offset: -259 (-1.01172)

- 10 bit, range `4..1019` – gain: `0x10244` (`1.00885`), offset: `-258` (`-4.03125`)
- 8 bit, range `16..240` – gain: `0x12490` (`1.14282`), offset: `-4681` (`-18.28516`)
- 10 bit, range `64..960` – gain: `0x12490` (`1.14282`), offset: `-4681` (`-73.14062`)

Note:

If set, this define will overload any other alpha gain value, such as the one determined by the functions *sv_matrix()*, *sv_matrixex()* or *sv_fifo_matrix()*.

This define is available on Centaurus II only.

See also:

The defines `SV_OPTION_ALPHAMIXER` and `SV_OPTION_ALPHAOFFSET`.

#define SV_OPTION_ALPHAMIXER

This define can be used to activate the alpha mixer of the DVS video board. With it you can mix (merge) two images (source A and B) stored in the video board buffer when the multi-channel operation mode (the define `SV_OPTION_MULTICHANNEL`) is activated. Possible values are:

- `SV_ALPHAMIXER_OFF` – Turns the mixer off.
- `SV_ALPHAMIXER_AB` – Turns the mixer on using the sources A and B.
- `SV_ALPHAMIXER_AB_PREMULTIPLIED` – Turns the mixer on using the sources A and B, where source A is expected to be multiplied already with the alpha data.

The two sources A and B are the first two output jacks. The source A always has to carry the alpha data. In the multi-channel operation mode one can use these two jacks independently with two instances of the FIFO API. This way you can send images of different color spaces into the two mixer inputs. Because this functionality uses the alpha channel (key), the storage of source A of the mixer must be either in YUVA or RGBA.

In normal mixer operation mode, i.e. when using `SV_ALPHAMIXER_AB`, the mixed value is calculated as follows: `value = A * alpha + B * (1 - alpha)`.

When using `SV_ALPHAMIXER_AB_PREMULTIPLIED`, the mixer source A carrying the alpha data is not multiplied with the alpha data anymore. In this mode the mixed value is calculated by the following formula: `value = A + B * (1 - alpha)`.

The expected color space value range of the alpha channel is always the full range 16-bit channel ranging from zero to 65535 (`0..65535`). It can be controlled with the defines `SV_OPTION_ALPHAGAIN` and `SV_OPTION_ALPHAOFFSET`. Instead of these you may as well use the functions *sv_matrix()*, *sv_matrixex()* and *sv_fifo_matrix()* to control the value range of the alpha channel.

Another option to handle the two sources A and B is to use only one jack and one FIFO API instance with the flag `SV_FIFO_FLAG_VIDEO_B`. With this approach and for backwards compatibility reasons, the structure element *sv_fifo_buffer.video_b* actually is the source A of the mixer hardware due to the fact that it has to carry the alpha data. The element *sv_fifo_buffer.video* is then the source B of the mixer. When setting the define `SV_FIFO_FLAG_VIDEO_B`, you do not have to set `SV_OPTION_ALPHAMIXER` additionally. It is internally applied automatically and set to the value `SV_ALPHAMIXER_AB`. With `SV_FIFO_FLAG_VIDEO_B` the images of source A and B must be of the same color space.

Note:

One limitation in the data path of source B exists: Any conversion from YUV422 to RGB444 data will be performed without filtering. Instead a value repetition is made.

This define is available on Centaurus II only.

See also:

The defines `SV_OPTION_MULTICHANNEL`, `SV_OPTION_ALPHAGAIN` and `SV_FIFO_FLAG_VIDEO_B`.

#define SV_OPTION_ALPHAOFFSET

This define can be used to specify an alpha offset. Its presentation is fixed point float where a value of `0x10000` (i.e. `1.00000`) represents the full value range. The default value is zero (`0`).

You may use it in combination with the define `SV_OPTION_ALPHAGAIN` to convert the value range of existing source alpha data into the full value range, i.e. the value range expected by the mixer hardware.

Note:

If set, this define will overload any other alpha offset value, such as the one determined by the functions *sv_matrix()*, *sv_matrixex()* or *sv_fifo_matrix()*.

This define is available on Centaurus II only.

See also:

The defines `SV_OPTION_ALPHAMIXER` and `SV_OPTION_ALPHAGAIN`.

#define SV_OPTION_DETECTION_NO4K

This define is not necessary when using Centaurus II with a firmware version of <x>.2.68.7_11_3 or higher. It was implemented to regain the speed of the raster detection which decreased with preparations made for 4K rasters in the DVS SDK. You may use it with Centaurus II (firmware version lower than the above mentioned) when a 4K raster detection is not needed and you want to increase the detection of all other rasters. Later firmware versions and newer hardware already provide a fast raster detection for all rasters.

This define switches off the automatic detection of 4K rasters connected to the input. The default value of this define is zero (`0`) and it can be enabled with a value of one (`1`).

Note:

This option call can be used with Centaurus II only (firmware version lower than <x>.2.68.7_11_3). For all others it will be without effect.

#define SV_OPTION_DETECTION_TOLERANCE

This define enables a very exact raster detection. You may use it to register even smaller variances of the incoming raster with your application. The default value is `-1`, and it can be enabled with the value zero (`0`).

Note:

This feature is only available on Centaurus II and requires a firmware version of <x>.2.68.8_11_4 or higher.

See also:

The define `SV_OPTION_SWITCH_TOLERANCE`.

#define SV_OPTION_DISABLESWITCHINGLINE

This define disables the switching line in the SDI data. A change of this setting will be in effect after the next *sv_videomode()* call.

#define SV_OPTION_DVI_OUTPUT

By using this option call you can change the DVI output bit depth. Possible values are:

- `SV_DVI_OUTPUT_DVI8` – Bit depth is 8 bits per pixel (default).
- `SV_DVI_OUTPUT_DVI12` – Bit depth is 12 bits per pixel.

#define SV_OPTION_FIELD_DOMINANCE

This option call defines the field dominance of any subsequent video I/O. The field dominance is a temporal concept. It does not change the positions of the fields, but instead their temporal sequence. This also implies that field 2 dominant transfers always start one field later compared to field 1 dominant transfers.

Possible values are:

- `0` – Field 1 is dominant (default for all rasters).
- `1` – Field 2 is dominant.

Note:

In progressive rasters this option call will return `SV_ERROR_VIDEOMODE`.

#define SV_OPTION_HDELAY

This define sets the horizontal sync delay, i.e. the number of pixels that the video output gets delayed in relation to the incoming sync. The notation is in pixels: For SD rasters in half-pixels and HD rasters in full pixels. The define `SV_OPTION_VDELAY` does the same in vertical direction.

#define SV_OPTION_HWWATCHDOG_ACTION

This define determines the behavior of the DVS hardware when the system crashes or freezes. The required actions are taken by a hardware driven watchdog able to control the SDI relays for a direct bypass of the SDI input to the SDI output and an additional GPI line for external notification. You have to call `SV_OPTION_HWWATCHDOG_TIMEOUT` prior to this define to set a valid watchdog timeout.

Possible values are:

- `SV_HWWATCHDOG_NONE` – No action will be taken.
- `SV_HWWATCHDOG_RELAY` – The SDI relays will connect input to output.
- `SV_HWWATCHDOG_GPI2` – The third GPI output line will be pulled to zero (`0`).
- `SV_HWWATCHDOG_MANUAL` – Turns off the automatic watchdog refresh (reset of the timeout counter) driven by the vertical sync. You have to use `SV_OPTION_HWWATCHDOG_REFRESH` to perform a manual watchdog refresh.

Note:

The above values can be combined except for `SV_HWWATCHDOG_NONE`.

This define is available on Centaurus II and Atomix LT only.

See also:

The defines `SV_OPTION_HWWATCHDOG_REFRESH`, `SV_OPTION_HWWATCHDOG_RELAY_DELAY`, `SV_OPTION_HWWATCHDOG_TIMEOUT`, and `SV_OPTION_HWWATCHDOG_TRIGGER`.

#define SV_OPTION_HWWATCHDOG_REFRESH

This define performs a manual watchdog refresh (reset of the timeout counter). It should be used in combination with `SV_HWWATCHDOG_MANUAL` only.

See also:

The define `SV_OPTION_HWWATCHDOG_ACTION`.

#define SV_OPTION_HWWATCHDOG_RELAY_DELAY

This define sets a delay for the activation of the SDI relays after the hardware watchdog has triggered. It will be effective only when SV_OPTION_HWWATCHDOG_ACTION is set to SV_HWWATCHDOG_RELAY. The unit is in milliseconds and the default value is zero (0). Possible values can range from zero to 255 ms (0..255).

See also:

The define SV_OPTION_HWWATCHDOG_ACTION.

#define SV_OPTION_HWWATCHDOG_TIMEOUT

This define sets a timeout after which the hardware watchdog will react (countdown timer). The unit is in milliseconds.

Please note that the timeout counter gets permanently reset by the vertical sync to its initial value as long as the system is operational. This will not be the case if SV_HWWATCHDOG_MANUAL is set. The timeout is the duration from the moment the system crashes or freezes until the watchdog takes the defined action.

Make sure that the timeout value is above the duration of a vertical sync. Otherwise the watchdog will always be triggered.

See also:

The define SV_OPTION_HWWATCHDOG_ACTION.

#define SV_OPTION_HWWATCHDOG_TRIGGER

This call can be used to set or reset the trigger status of the hardware watchdog with immediate effect.

Possible values are:

- SV_HWWATCHDOG_NONE – Clears the triggered SDI relays and the GPI.
- SV_HWWATCHDOG_RELAY – Triggers the SDI relays (will be set to bypass).
- SV_HWWATCHDOG_GPI2 – Triggers the third GPI output line (will be pulled to zero (0)).

With the power off the watchdog of the hardware is always in its triggered state, and it will stay triggered until the driver of the DVS video board is loaded. The driver will then reset the watchdog and output the board's regular signal.

However, this start-up behavior can be changed. Then the driver will leave the watchdog in its triggered state until it is finally cleared by calling SV_OPTION_HWWATCHDOG_TRIGGER. You can configure the watchdog's start-up behavior under Windows in the DVSConf program by adjusting the setting 'Relay startup in bypass' on the tab 'Settings', and under Linux with the driver's load parameter relay.

Note:

This define is available on Centaurus II and Atomix LT only.

#define SV_OPTION_INPUTFILTER

This define sets the video 422-to-444 filter. See the function *sv_matrix()*.

See also:

The define SV_OPTION_OUTPUTFILTER.

#define SV_OPTION_INPUTPORT

This define selects the video input port that will be used for a record operation:

- SV_INPUTPORT_SDI – Selects the SDI input, i.e. link A and B of the SDI will be used. This is the default setting.
- SV_INPUTPORT_SDI2 – Selects the second SDI input, i.e. link B of the SDI will be used.
- SV_INPUTPORT_SDI3 – Selects the third SDI input, i.e. link C of the SDI will be used.

Note:

For a dual-link capturing you have to select SV_INPUTPORT_SDI.

On Centaurus II the call for SV_INPUTPORT_SDI2 uses the explicitly designated (HD) link B input of the board. Otherwise the on-board SD-only input will be used. The call for SV_INPUTPORT_SDI3 will use the on-board SD-only input.

#define SV_OPTION_IOMODE

This define sets the video I/O mode for the SDI ports. When setting its values globally, both in- and output will be set to the same video I/O mode. To set values for each jack (in- or output) independently it is recommended to use the Jack API (see chapter API – Jack API).

- SV_IOMODE_YUV422 – YUV 4:2:2.
- SV_IOMODE_YUV444 – YUV 4:4:4.
- SV_IOMODE_YUV422A – YUVA 4:2:2:4.
- SV_IOMODE_YUV444A – YUVA 4:4:4:4.
- SV_IOMODE_RGB – RGB 4:4:4.
- SV_IOMODE_RGBA – RGBA 4:4:4:4.
- SV_IOMODE_YUV422_12 – YUV 4:2:2 (12 bit).
- SV_IOMODE_YUV444_12 – YUV 4:4:4 (12 bit).
- SV_IOMODE_RGB_12 – RGB 4:4:4 (12 bit).
- SV_IOMODE_CLIP – Enables the clipping of the color value range for the used I/O mode. The color range will be clipped to 64..940 for RGB, Y as well as A (10 bit, 8 bit: 16..235), and to 64..960 for U and V (10 bit, 8 bit: 16..240).
- SV_IOMODE_IO_MASK – Mask for the video I/O mode.

Note:

The flag SV_IOMODE_CLIP is available on Centaurus II only.

A clipping of color value ranges for the input is currently not implemented.

Atomix only: When setting an I/O mode for an input, the automatic I/O mode switching is deactivated. It can be activated again with the define SV_OPTION_IOMODE_AUTODETECT.

The DVI output is RGB always and its bit depth can be changed independently with the define SV_OPTION_DVI_OUTPUT.

#define SV_OPTION_IOMODE_AUTODETECT

This define activates or deactivates the automatic switching of the input I/O mode. The automatic I/O mode switching evaluates the VPID data (SMPTE352) of the incoming signal and the I/O mode will be switched when a different I/O mode is detected. It is by default activated and can be deactivated either with this define or by setting an I/O mode for an input explicitly.

Note:

This define and the automatic I/O mode switching are available on Atomix only.

See also:

The defines SV_OPTION_IOMODE and SV_QUERY_IOMODEINERROR.

#define SV_OPTION_IOSPEED

This define configures the SDI I/O link (port) speed. For an input the speed will be automatically detected and set accordingly. For an output you can set the speed with this define. The default speeds are as detailed below.

- `SV_IOSPEED_1GB5` – I/O speed for HD rasters, i.e. 1.5 Gbit/s (default output speed for HD rasters or larger).
- `SV_IOSPEED_3GBA` – Currently not supported.
- `SV_IOSPEED_3GBB` – 3.0 Gbit/s (level B).
- `SV_IOSPEED_SDTV` – I/O speed for SD rasters (default output speed for SD rasters).

Note:

SD rasters can only be set to `SV_IOSPEED_SDTV`.

Other rasters cannot be set to `SV_IOSPEED_SDTV`.

This define is supported on Atomix only.

#define SV_OPTION_MAINOUTPUT

Most DVS video devices that provide SDI as well as DVI outputs cannot give out fully correct sync pulses on both output ports simultaneously. With this option call you can determine the output that should be optimized for correct sync pulses, i.e. you can select the main output that should be given priority when generating sync pulses.

Possible values are:

- `SV_MAINOUTPUT_SDI` – SDI is the main output (default).
- `SV_MAINOUTPUT_DVI` – DVI is the main output.

#define SV_OPTION_OUTPUTFILTER

This define sets the video 444-to-422 filter. See the function *sv_matrix()*.

See also:

The define `SV_OPTION_INPUTFILTER`.

#define SV_OPTION_OUTPUTPORT

This define configures the video output. Most DVS video devices provide dual-link outputs. With this value you can determine the behavior of these outputs.

- `SV_OUTPUTPORT_DEFAULT` – Output ports A and B are set to their usual behavior: For example, with YUVA the A port gives out the video data, while the B port gives out the key data.
- `SV_OUTPUTPORT_MIRROR` – Output ports A and B show both the output signal of port A (Centaurus II only).

#define SV_OPTION_PULLDOWN_STARTLTC

This define sets a trigger timecode. It performs the same operation as the command `SV_PULLDOWN_CMD_STARTLTC` of the function *sv_pulldown()*. For further information see this function.

Note:

This define is supported on Atomix only.

#define SV_OPTION_PULLDOWN_STARTPHASE

This define sets the pulldown start phase. It performs the same operation as the command SV_PULLDOWN_CMD_STARTPHASE of the function *sv_pulldown()*. For further information see this function.

Note:

This define is supported on Atomix only.

#define SV_OPTION_PULLDOWN_STARTVTRTC

This define sets a trigger timecode. It performs the same operation as the command SV_PULLDOWN_CMD_STARTVTRTC of the function *sv_pulldown()*. For further information see this function.

Note:

This define is supported on Atomix only.

#define SV_OPTION_SWITCH_TOLERANCE

This define sets the tolerance of the automatic SDI raster detection as well as the tolerance of the sync detection. Short-timed interferences within the same video raster will be ignored, for example, when switching between two players.

Normally, a shift of even a few pixels will lead to a false sync state (see the define SV_QUERY_SYNCSTATE). By using this define shifts of up to half a horizontal line will be allowed without loosing the sync state. It can be used in conjunction with the sync modes SV_SYNC_EXTERNAL, SV_SYNC_BILEVEL or SV_SYNC_TRILEVEL (see the function *sv_sync()*).

Possible values are:

- SV_SWITCH_TOLERANCE_OFF – No extra tolerance is set.
- SV_SWITCH_TOLERANCE_DEFAULT – Same as SV_SWITCH_TOLERANCE_OFF.
- SV_SWITCH_TOLERANCE_DETECT – If this flag is set, the SDI raster detection will ignore short instabilities of the video raster.
- SV_SWITCH_TOLERANCE_SYNC – If this flag is set, the input sync will ignore horizontal shifts of up to a half line. If the shift is higher, the sync state will be lost.
- SV_SWITCH_TOLERANCE_DETECT_CYCLES(x) – This flag implicitly sets SV_SWITCH_TOLERANCE_DETECT as well while specifying a cycle duration in frames. During the given duration the SDI raster detection will ignore instabilities of the video raster.

Note:

The values SV_SWITCH_TOLERANCE_DETECT and SV_SWITCH_TOLERANCE_SYNC can be combined.

See also:

The function *sv_sync()* and the defines SV_OPTION_DETECTION_TOLERANCE as well as SV_QUERY_SYNCSTATE.

#define SV_OPTION_SYNCMODE

This define configures the sync output mode of the DVS video device. The inputs will at all times be synchronized with their corresponding incoming signals independent of any sync setting. It performs the same operation as the function *sv_sync()*. For parameters and flags see this function.

#define SV_OPTION_SYNCOUT

This define configures the sync pulse of the analog component signal (RGB). It performs the same operation as the function *sv_sync_output()*. For parameters and flags see this function.

Note:

This define is available on Centaurus II only.

#define SV_OPTION_SYNCOUTDELAY

This define sets the delay of the horizontal sync output, i.e. the number of pixels that the horizontal sync output gets delayed in relation to the video output. The notation is in pixels: For SD rasters in half-pixels and HD rasters in full pixels. The define SV_OPTION_SYNCOUTVDELAY does the same in vertical direction.

#define SV_OPTION_SYNCOUTVDELAY

This define sets the delay of the vertical sync output, i.e. the number of lines that the vertical sync output gets delayed in relation to the video output. SV_OPTION_SYNCOUTDELAY does the same in horizontal direction.

#define SV_OPTION_SYNCSELECT

This define selects the sync source in case there are multiple sync input ports available on the DVS video board:

- SV_SYNCSELECT_LINKA – Selects port A as the sync source (default).
- SV_SYNCSELECT_LINKB – Selects port B as the sync source.

Note:

This define is supported on Atomix only. With it you can select the sync source when SV_SYNC_EXTERNAL is selected and you have more than one I/O channel configured for the board.

#define SV_OPTION_VDELAY

This define sets the vertical sync delay, i.e. the number of lines that the video output gets delayed in relation to the incoming sync. The define SV_OPTION_HDELAY does the same in horizontal direction.

#define SV_OPTION_VIDEOMODE

This define sets the video mode. You can use it with the SV_MODE_<xxx> defines detailed in the header file *dvs_clib.h*. Any audio settings made, for example, via SV_OPTION_AUDIOAESROUTING or SV_OPTION_AUDIOCHANNELS will not be affected.

Note:

To check whether a certain video raster is available on the DVS video device use sv_query(SV_QUERY_MODE_AVAILABLE).

Not all combinations of the various SV_MODE_<xxx> flags may be possible. For example, SV_MODE_FLAG_PACKED is not supported together with SV_MODE_STORAGE_FRAME.

See also:

The function *sv_videomode()* and the define SV_QUERY_MODE_CURRENT.

#define SV_QUERY_CARRIER

This define will return TRUE if the input signals are valid.

#define SV_QUERY_DISPLAY_LINENR

This define returns the current display line number.

#define SV_QUERY_GENLOCK

This query will return TRUE if a genlock is available.

#define SV_QUERY_HDELAY

This define returns the horizontal sync delay setting. See the define SV_OPTION_HDELAY.

#define SV_QUERY_INPUTFILTER

This define returns SV_INPUTFILTER_<xxx>. See the define SV_OPTION_INPUTFILTER and the function *sv_matrix()*.

#define SV_QUERY_INPUTPORT

This define returns the input port setting. See the define SV_OPTION_INPUTPORT.

#define SV_QUERY_INPUTRASTER

This query returns the raster of the signal connected to the SDI input.

See also:

The defines SV_QUERY_INPUTRASTER_SDIA, SV_QUERY_INPUTRASTER_SDIB and SV_QUERY_INPUTRASTER_SDIC.

#define SV_QUERY_INPUTRASTER_GENLOCK

This define returns the raster of the signal connected to the genlock input.

See also:

The define SV_QUERY_INPUTRASTER_GENLOCK_TYPE.

#define SV_QUERY_INPUTRASTER_GENLOCK_TYPE

This define returns the type of signal connected to the genlock input:

- SV_SYNC_INT – No signal type recognized.
- SV_SYNC_BILEVEL – Bilevel signal type recognized.
- SV_SYNC_TRILEVEL – Trilevel signal type recognized.

See also:

The define SV_QUERY_INPUTRASTER_GENLOCK.

#define SV_QUERY_INPUTRASTER_SDIA

This query returns the raster of the signal connected to the SDI input channel A.

See also:

The defines SV_QUERY_INPUTRASTER, SV_QUERY_INPUTRASTER_SDIB and SV_QUERY_INPUTRASTER_SDIC.

#define SV_QUERY_INPUTRASTER_SDIB

This query returns the raster of the signal connected to the SDI input channel B.

See also:

The defines SV_QUERY_INPUTRASTER, SV_QUERY_INPUTRASTER_SDIA and SV_QUERY_INPUTRASTER_SDIC.

#define SV_QUERY_INPUTRASTER_SDIC

This query returns the raster of the signal connected to the SDI input channel C.

Note:

This define is available on Centaurus II only.

See also:

The defines SV_QUERY_INPUTRASTER, SV_QUERY_INPUTRASTER_SDIA and SV_QUERY_INPUTRASTER_SDIB.

#define SV_QUERY_IOCHANNELS

This define returns the number of the available I/O channels. See the define SV_OPTION_MULTICHANNEL.

#define SV_QUERY_IOLINK_MAPPING

This define returns a bit mask of the currently active physical ports. Because it can be different for each jack, it can only be used in conjunction with the function *sv_jack_query()*.

#define SV_QUERY_IOLINKS_INPUT

This define returns the number of the physical input links (SDI) available on the DVS video device.

#define SV_QUERY_IOLINKS_OUTPUT

This define returns the number of the physical output links (SDI) available on the DVS video device.

#define SV_QUERY_IOMODE

This define returns the setting of the output I/O mode. See the define SV_OPTION_IOMODE.

#define SV_QUERY_IOMODEINERROR

This define returns the error if an automatic switching of the input I/O mode fails.

Note:

This define is supported on Atomix only.

See also:

The defines SV_OPTION_IOMODE_AUTODETECT and SV_QUERY_SMPTE352.

#define SV_QUERY_IOSPEED

This define returns the configured SDI I/O speed. See the define SV_OPTION_IOSPEED. Because it can be different for each jack, it can be used in conjunction with the function *sv_jack_query()* only.

#define SV_QUERY_IOSPEED_SDIA

This define returns the detected I/O speed on the SDI input A.

Note:

This define is supported on Atomix only.

#define SV_QUERY_IOSPEED_SDIB

This define returns the detected I/O speed on the SDI input B.

Note:

This define is supported on Atomix only.

#define SV_QUERY_IOSPEED_SDIC

This define returns the detected I/O speed on the SDI input C.

Note:

This define is supported on Atomix only.

#define SV_QUERY_IOSPEED_SDID

This define returns the detected I/O speed on the SDI input D.

Note:

This define is supported on Atomix only.

#define SV_QUERY_MODE_AVAILABLE

This define returns TRUE if the specified mode is available.

#define SV_QUERY_MODE_CURRENT

This define returns the current video and/or audio mode (SV_MODE_<xxx> defines) set, for example, with the define SV_OPTION_VIDEOMODE.

#define SV_QUERY_OUTPUTFILTER

This define returns SV_OUTPUTFILTER_<xxx>. See the define SV_OPTION_OUTPUTFILTER and the function *sv_matrix()*.

#define SV_QUERY_OUTPUTPORT

This query returns the output port setting. See the define SV_OPTION_OUTPUTPORT.

#define SV_QUERY_RASTER_DROPFRAME

This define returns whether the currently set video raster for the storage (output) is a drop-frame raster.

#define SV_QUERY_RASTER_FPS

This define returns the frequency (frames per second) of the video raster currently set for the storage (output).

#define SV_QUERY_RASTER_INTERLACE

This define returns whether the currently set video raster for the storage (output) is an interlaced raster.

#define SV_QUERY_RASTER_SEGMENTED

This define returns whether the currently set video raster for the storage (output) is a segmented-frames raster.

#define SV_QUERY_RASTER_XSIZE

This define returns the horizontal size (x-axis) of the video raster at the output.

See also:

The define SV_QUERY_STORAGE_XSIZE.

#define SV_QUERY_RASTER_YSIZE

This define returns the vertical size (y-axis) of the video raster at the output.

See also:

The define SV_QUERY_STORAGE_YSIZE.

#define SV_QUERY_RASTERID

This define returns the raster index of the currently used raster.

#define SV_QUERY_RECORD_LINENR

This define returns the current record line number.

#define SV_QUERY_SMPTE352

This define returns the payload of the SMPTE352 ANC package detected at the input.

Note:

This define is supported on Atomix only.

#define SV_QUERY_STORAGE_XSIZE

This define returns the horizontal size (x-axis) of the video raster in the DVS video board buffer (storage).

See also:

The define SV_QUERY_RASTER_XSIZE.

#define SV_QUERY_STORAGE_YSIZE

This define returns the vertical size (y-axis) of the video raster in the DVS video board buffer (storage).

See also:

The define SV_QUERY_RASTER_YSIZE.

#define SV_QUERY_SYNCMODE

This define returns the setting of the sync mode. See the define SV_OPTION_SYNCMODE and the function *sv_sync()*.

#define SV_QUERY_SYNCOUT

This query returns the setting of the sync output. See the define SV_OPTION_SYNCOUT and the function *sv_sync_output()*.

#define SV_QUERY_SYNCOUTDELAY

This define returns the setting of the horizontal sync output delay. See the define SV_OPTION_SYNCOUTDELAY.

#define SV_QUERY_SYNCOUTVDELAY

This define returns the setting of the vertical sync output delay. See the define SV_OPTION_SYNCOUTVDELAY.

#define SV_QUERY_SYNCSTATE

This define will return TRUE if the current sync mode is locked. Otherwise it will be FALSE.

#define SV_QUERY_TICK

This define returns the current tick.

#define SV_QUERY_VDELAY

This define returns the setting of the vertical sync delay. See the define SV_OPTION_VDELAY.

#define SV_QUERY_VIDEOINERROR

This define will return SV_OK if no error is detected at the video input. Otherwise it will return the current video input error.

Function Documentation

int sv_pulldown (sv_handle * *sv*, int *cmd*, int *param*)

This function changes settings related to the pulldown feature. Currently only the start phase for transfers to or from a VTR can be changed.

Parameters:

sv – Handle returned from the function *sv_open()*.

cmd – Command. See list below.

param – Parameter.

Parameters for *cmd*:

- SV_PULLDOWN_CMD_STARTPHASE – Sets the pulldown start phase. The pulldown operation is triggered by using a timed FIFO. The following phases are possible:
  - SV_PULLDOWN_STARTPHASE_A – Two fields beginning with field 1.
  - SV_PULLDOWN_STARTPHASE_B – Three fields beginning with field 1.
  - SV_PULLDOWN_STARTPHASE_C – Two fields beginning with field 2.
  - SV_PULLDOWN_STARTPHASE_D – Three fields beginning with field 2.
- SV_PULLDOWN_CMD_STARTVTRTC – Sets a trigger timecode instead of binding this to the timed FIFO. The pulldown operation is triggered when the specified VTR timecode is received.
- SV_PULLDOWN_CMD_STARTLTC – Sets a trigger timecode instead of binding this to the timed FIFO. The pulldown operation is triggered when the specified LTC timecode is received.

Returns:

If the function succeeds, it returns SV_OK. Otherwise it will return the error code SV_ERROR_<xxx>.

int sv_sync (sv_handle * *sv*, int *sync*)

This function configures the sync output mode of the DVS video device. It determines how the output raster generator shall be clocked and synchronized. The inputs will at all times be synchronized with their corresponding incoming signals independent of any sync setting. This function performs the same operation as the define SV_OPTION_SYNCMODE.

Parameters:

sv – Handle returned from the function *sv_open()*.

sync – SV_SYNC_<xxx> define. See list below.

Parameters for *sync*:

- SV_SYNC_INTERNAL – Sync mode is set to internal.
- SV_SYNC_EXTERNAL – If an input signal exists (at the SDI input) lock to it. If no signal is present, revert back to internal.
- SV_SYNC_GENLOCK_ANALOG – Genlocked to an analog sync. This define is obsolete, use SV_SYNC_BILEVEL or SV_SYNC_TRILEVEL instead.
- SV_SYNC_GENLOCK_DIGITAL – Genlocked to an external digital sync (currently not used).
- SV_SYNC_SLAVE – Currently not used.
- SV_SYNC_AUTO – Same as SV_SYNC_EXTERNAL.
- SV_SYNC_MODULE – Currently not used.
- SV_SYNC_BILEVEL – Sets an analog bilevel sync and uses the reference input signal. Usually used for SD rasters.
- SV_SYNC_TRILEVEL – Sets an analog trilevel sync and uses the reference input signal. Usually used for HD rasters.
- SV_SYNC_HVTTL – Sets a separate H and V sync mode. Combine with one of the following signal forms:
  - SV_SYNC_HVTTL_HFVF – Falling edge of H and falling edge of V.
  - SV_SYNC_HVTTL_HRVF – Rising edge of H and falling edge of V.
  - SV_SYNC_HVTTL_HFVR – Falling edge of H and rising edge of V.
  - SV_SYNC_HVTTL_HRVR – Rising edge of H and rising edge of V.
- SV_SYNC_LTC – Obsolete. It was used for a previous disk recorder product by DVS to synchronize to the LTC sync.

Parameters for *sync* (Flags):

- SV_SYNC_FLAG_SDTV – Enables a cross-sync on an NTSC/PAL source. Not all sync combinations may be possible. Can be used in combination with a bilevel or trilevel sync.

Returns:

If the function succeeds, it returns SV_OK. Otherwise it will return the error code SV_ERROR_<xxx>.

Note:

The DVS video device automatically switches to an internal sync if it is set to an external sync and no sync signal is connected. To check if a sync input signal is available and the board is locked to it use sv_query(SV_QUERY_SYNCSTATE).

Centaurus II (firmware 3.2.71.2_19_11 or higher) and SDK 3.2.14.0 or higher: The driver automatically toggles between SV_SYNC_BILEVEL and SV_SYNC_TRILEVEL depending on the signal connected to the reference input.

Example:

```
  int example_setsynctobilevel(sv_handle * sv)
  {
    int res = sv_sync(sv, SV_SYNC_BILEVEL);
    if(res != SV_OK) {
      fprintf(stderr, "Error: sv_sync() failed = %d '%s'", res,
sv_geterrortext(res));
      return FALSE;
    }

    return TRUE;
  }
```

int sv_sync_output (sv_handle * *sv*, int *syncout*)

This function configures the sync pulse of the analog component signal (RGB). It performs the same operation as the define SV_OPTION_SYNCOUT.

Parameters:

sv – Handle returned from the function *sv_open()*.

syncout – SV_SYNCOUT_<xxx> define. See list below.

Parameters for *syncout*:

- SV_SYNCOUT_OFF – No sync output signal (0 V).
- SV_SYNCOUT_BILEVEL – Bilevel sync output (0.7 V).
- SV_SYNCOUT_TRILEVEL – Trilevel sync output (0.7 V).
- SV_SYNCOUT_HVTTL_HFVF – H and V sync TTL signal (4.0 V) with falling H and falling V.
- SV_SYNCOUT_HVTTL_HFVR – H and V sync TTL signal with falling H and rising V.
- SV_SYNCOUT_HVTTL_HRVF – H and V sync TTL signal with rising H and falling V.
- SV_SYNCOUT_HVTTL_HRVR – H and V sync TTL signal with rising H and rising V.
- SV_SYNCOUT_DEFAULT – Raster default sync output mode.

Parameters for *syncout* (Flags):

- SV_SYNCOUT_OUTPUT_GREEN – Enables a sync on green for bilevel and trilevel sync outputs.
- SV_SYNCOUT_LEVEL_SET(<x>) – Obsolete. It was used to set the voltage level for the bilevel sync. Could be set to either 0.7 V (7) or 4.0 V (40).

Returns:

If the function succeeds, it returns SV_OK. Otherwise it will return the error code SV_ERROR_<xxx>.

Note:

This function is available on Centaurus II only.

int sv_videomode (sv_handle * *sv*, int *mode*)

This function sets the video mode for the DVS video device, including settings for raster and storage mode. For backwards compatibly reasons this function sets configurations for audio as well (see note below).

Parameters:

sv – Handle returned from the function *sv_open()*.

mode – SV_MODE_<xxx> define. See the file *dvs_clib.h* for all possible defines.

Returns:

If the function succeeds, it returns SV_OK. Otherwise it will return the error code SV_ERROR_<xxx>.

Note:

To set configurations for video and audio separately use the option calls SV_OPTION_VIDEOMODE and SV_OPTION_AUDIO<xxx>.

To check whether a certain video raster is available on the DVS video device use sv_query(SV_QUERY_MODE_AVAILABLE).

Not all combinations of the various SV_MODE_<xxx> flags may be possible. For example, SV_MODE_FLAG_PACKED is not supported together with SV_MODE_STORAGE_FRAME.

See also:

The define SV_OPTION_VIDEOMODE.

int sv_vsyncwait (sv_handle * *sv*, int *operation*, sv_vsyncwait_info * *pinfo*)

This function stops the program flow and waits for the next vertical sync to occur. After this the function returns the control and continues the program flow.

Parameters:

sv – Handle returned from the function *sv_open()*.

operation – Sets the operation mode. See list below.

pinfo – Pointer to the *sv_vsyncwait_info* structure that will be filled by the function with sync information.

Parameters for *operation*:

- SV_VSYNCWAIT_DISPLAY – Synchronizes for a display.
- SV_VSYNCWAIT_RECORD – Synchronizes for a record.
- SV_VSYNCWAIT_CANCEL – Cancels any sync operation.
- SV_VSYNCWAIT_STATUS – Does not wait but fills in the structure.

Returns:

If the function succeeds, it returns SV_OK. Otherwise it will return the error code SV_ERROR_<xxx>.

Note:

If the SV_VSYNCWAIT_CANCEL value of the *operation* parameter is passed, all sync operations in a wait status will be cancelled.

API – FIFO API

Detailed Description

The FIFO API is the main real-time capture and play-out API for video board integration customers.

This chapter presents the FIFO API to be used in direct I/O video applications with the DVS audio and video hardware. Direct I/O applications are those that fulfill the following conditions:

- They use a CPU processing for the video/audio data in the 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 the video data during transfers. This way it is possible to display the data directly in an imaging software or to store it on formatted hard disks. For example, you could implement still store applications using system RAM or disk recording applications in DMA operation mode.

FIFO API Concept

The FIFO API is a collection of functions and structures that directly access the DVS hardware (the video board memory basically), thus allowing to handle video images in real-time, to work on input video signals, to store the incoming data, etc.

Theory of Operation

The video board's memory is split into buffers. There are two processes that may be asynchronous: The video I/O and the data transfer. The main reason to have two different processes is to be able to fully utilize the on-board memory to level out temporary performance fluctuations, such as networking and disk transfers, that might block the PCI bus for short amounts of time. The API can be considered as a FIFO (first-in-first-out) with two data paths by default, one for input and one for output. It is possible to flush the FIFO for both output and input to catch up and discard or keep the images currently displayed.

Hardware Overview

The video board contains enough memory for at least 0.5 seconds of video. The actual amount of storage time depends on the selected video raster. Normally the video board is used in YUV 4:2:2 mode, but depending on the actual version and type of the video board you are using, it can also support other color modes such as RGB 4:4:4, RGBA 4:4:4:4, YUVA 4:2:2:4, or YUVA 4:4:4:4.

FIFO Transfer Data Mode

The implementation uses the concept of FIFOs. All DVS video devices provide at least one FIFO for an input and one for an output. Internally the FIFOs are implemented using the memory on the video board or the system memory depending on the initialization parameters. To transfer data to the video board you can use the on-board DMA of the video board.

Direct Memory Access Operation (DMA)

You provide the FIFO API buffer with the address where the data shall be transferred to or from, and the actual DMA operation will be performed automatically during the *sv_fifo_putbuffer()* call. When allocating the buffers in main memory, take care to align the buffers to a page boundary. You can query the minimum needed alignment for DMA transfers for your DVS video board product by calling the define SV_QUERY_DMAALIGNMENT. Aligning data to page boundaries properly can result in a little increase of the overall performance. The *preview* example program implements a capturing of video to the main memory using DMA. The *counter* and the *dpxio* example programs demonstrate a DMA from the main memory to the video board. All example programs are shortly explained in chapter Example Projects Overview.

Driver Functionality

The driver uses the on-board memory to store the DMA data before it is sent to the video output and vice versa during input. With this the driver will start a recording as soon as the *sv_fifo_start()* call is sent. You do not need to take much care about when the getbuffer/putbuffer calls are done to transfer the data to the main memory or to another destination. This means that the board buffers up to 32 frames of video and audio; the actual amount depends on the selected video raster and the amount of memory available on the video board.

Audio Handling

Because of the different number of audio samples per field, the *sv_fifo_getbuffer()* function returns the appropriate number of samples in the *size* field of the audio structure in the *sv_fifo_buffer* structure which is the number of bytes needed. For example, 3204 in 16 bit audio mode means (3204 / 2 (channels) / 2 (16 bit = 2 bytes) = 801 samples). If the device is in a drop-frame mode, the number of samples is equal for all frames if (48048 / fps) is an even number, in non-drop-frame mode if (48000 / fps) is an even number. For 24-Hz modes this is always the case, but e.g. for 29.97 Hz this is 800.8 samples per field; this is realized with a sample count alternating between 801 and 800. The `dpxio` example program shows how to display and record AIFF files. The data is in stereo samples where each (mono) sample is of either 32 bit or 16 bit. The data is little endian and each stereo pair is transferred in its own buffer. You can see this in the `dpxio` example where during display of AIFF, the data is reordered due to the AIFF file being big endian.

So for 32 bits:

- 0 left4bytes right4bytes
- 8 left4bytes right4bytes
- 16 ... ...
- ...

Hardware

On all DVS video boards at least one FIFO handle can be opened for an input and one for an output. All functions work on audio, video and timecode at the same time, thus synchronizing audio, video and timecode is a problem left to the video board driver.

FIFOs and Jacks

The Jack API (see chapter API – Jack API) is an extension for the FIFO API to provide independent I/O streams and/or multiple channels of video/audio. For each jack one FIFO has to be opened. However, the FIFO API can be used without specifying any jacks: Then two FIFOs can be applied (the default jacks), i.e one FIFO for an input and one for an output. With the define `SV_QUERY_FEATURE` you can query your DVS video board product about the supported features, such as independent I/O or multiple channels.

Field or Frame Storage

If the video device is using a frame-wise storage, the *addr[1]-addr[0]* pointer is equal to one line of video; if not, it is equal or larger than one field. See also chapter Info – Storage Formats.

Timed Operations

Timed operations are done by specifying a *when* parameter in the optional *sv_fifo_bufferinfo* structure to the *sv_fifo_getbuffer()* call. A *when* value is a driver timestamp, often referred to as a 'tick' value. Furthermore, the flag `SV_FIFO_FLAG_TIMEDOPERATION` must be set. Timed operations can be useful in conjunction with the function *sv_vtrcontrol()*, for instance. This is shown in the `dpxio` example program as well.

Driver Tick Counter

When starting the video board driver, an internal counter is initialized that generates continuous ticks until the driver is stopped. Each tick correlates with one video field or frame, i.e. the driver counts field-wise in interlaced video modes and frame-wise in progressive video modes. The tick

counter is particularly important for timed operations. You can determine the current tick by evaluating the tick field of the *sv_fifo_info* structure.

Global Clock

With the DVS video boards currently supported by the DVS SDK (see section Supported DVS Video Board Products) the global clock is a hardware based microsecond counter.

In-to-Out Delay

The In-delay is the time it takes to get the image into your application from the moment it was captured from the SDI signal. Thus, for a short In-delay the input FIFO should be as small as possible. The depth of the FIFO depends on the frequency that the *sv_fifo_getbuffer()* / *sv_fifo_putbuffer()* pair is called in relation to the raster frequency (frame rate). When the pair is called quicker than the raster frequency, the *sv_fifo_getbuffer()* call will block automatically when no more images are available.

However, the automatic live mode increases the In-delay by up to two frames. It will be activated when only an input FIFO is running and can be disabled manually with the flag `SV_FIFO_FLAG_NO_LIVE`. Of course, when an output FIFO is active, the automatic live mode is deactivated anyway.

For later calculations the tick when the image was captured from the SDI will be of interest. You can retrieve it by reading the *sv_fifo_bufferinfo.when* parameter from the *sv_fifo_putbuffer()* call.

The Out-delay is more dynamic and you may want to control the buffers manually during an output. The Out-delay is the time it takes to give the image out on the SDI signal from the moment its transfer via DMA to the video board storage was started. As with the In-delay, for a short Out-delay the output FIFO depth should be as small as possible, but with the exception that for internal reasons it must contain at least four fields / two frames (the minimum buffer depth). To keep the output FIFO at the minimum buffer depth, you have to wait one or more vertical syncs before the next *sv_fifo_getbuffer()* / *sv_fifo_putbuffer()* pair is called.

```
res = sv_fifo_status(hd->sv, hd->fifo, &status);
if(res != SV_OK)  {
...  // Appropriate error handling
}

// Wait until I need a new buffer
if(hd->bfieldbased || hd->bdualsdi) {
  // Fields
  while(hd->running && ((status.nbuffers - (status.availbuffers + 1)) > 4)) {
    // Wait for next vsync
    res = sv_fifo_vsyncwait(hd->sv, hd->fifo);
    if(res != SV_OK)  {
    ...
    }

    // Check buffer count
    res = sv_fifo_status(hd->sv, hd->fifo, &status);
    if(res != SV_OK)  {
    ...
    }
  }
} else {
  // Frames
  while(hd->running && ((status.nbuffers - (status.availbuffers + 1)) > 2)) {
    // Wait for next vsync
    res = sv_fifo_vsyncwait(hd->sv, hd->fifo);
    if(res != SV_OK) {
    ...
    }

    // Check buffer count
    res = sv_fifo_status(hd->sv, hd->fifo, &status);
    if(res != SV_OK)  {
    ...
    }
```

```
    }
  }
```

For later calculations you need to know when the image will be displayed on the SDI signal. As with an input, the tick can be read from the *sv_fifo_bufferinfo.when* parameter of the *sv_fifo_putbuffer()* call. Thus, the next image that should be given out is the above mentioned tick plus one (`+ 1`) or two (`+ 2`) depending on the storage mode.

With this knowledge about the In- and Out-delay you should be in a position to implement a static In-to-Out delay in your application. For further information about this take a look at the example program *dmaloop* and the parameter `-d=<delay>`.

Mixer Functionality

The FIFO API provides a mixer that can be used to merge two images available in the storage (RAM). Currently it can be used on Centaurus II only. It is possible to run the hardware mixer from either one FIFO with the define `SV_FIFO_FLAG_VIDEO_B` or two FIFOs by using the multi-channel operation mode (see the defines `SV_OPTION_MULTICHANNEL` and `SV_OPTION_ALPHAMIXER`).

Master/Slave

The FIFO API contains a minimal slave mode implementation. This is able to respond to most common status requests without interaction from the user application. It delivers back one command per frame from the connected edit controller; all RS-422 slave commands in the `0x200`/`0x400` section are responded with `ACK`, and status commands are also acknowledged. The connected master polls the controlled device for timecode and status once per frame. On the input FIFO the VTR commands are received as one line per vertical sync in the *vtrcmd* entries. The fields *vtr_tc*, *vtr_ub* and *vtr_info* in the output FIFO are used to respond to slave mode requests as timecode, user bytes, and info bits. On the other hand, the *vtr_tc*, *vtr_ub* and *vtr_info* values in the input FIFO contain the timecode that was polled from the connected VTR slave device. Using the slave mode implementation of the FIFO API, it is not possible to implement a full slave mode. If you need a full featured slave mode, you have to implement the slave mode yourself by using either the *sv_slaveinfo_get()* / *sv_slaveinfo_set()* functions or by using the *sv_rs422_rw()* functionality. They can be used in combination with the FIFO API.

Multi-threading

The functions of the FIFO API are thread-safe. Only one *sv_fifo_<xxx>* function per FIFO will be active at a time. Multiple *sv_fifo_<xxx>* function calls from different threads are executed one after another.

Timecode Handling

The timecode values in the *sv_fifo_buffer* structure are identical to any other timecode occurrence in the DVS SDK, meaning the frames value is in the LSB (least significant byte) while the hours value resides in the MSB (most significant byte).

Data Structures

- struct sv_fifo_ancbuffer
- struct sv_fifo_buffer
- struct sv_fifo_bufferinfo
- struct sv_fifo_configinfo
- struct sv_fifo_info

Defines

- #define SV_FIFO_BUFFERINFO_VERSION_1
- #define SV_FIFO_FLAG_ANC

- #define SV_FIFO_FLAG_AUDIOINTERLEAVED
- #define SV_FIFO_FLAG_AUDIOONLY
- #define SV_FIFO_FLAG_CLOCKEDOPERATION
- #define SV_FIFO_FLAG_DMARECTANGLE
- #define SV_FIFO_FLAG_DONTBLOCK
- #define SV_FIFO_FLAG_FIELD
- #define SV_FIFO_FLAG_FLUSH
- #define SV_FIFO_FLAG_NO_LIVE
- #define SV_FIFO_FLAG_NODMA
- #define SV_FIFO_FLAG_NODMAADDR
- #define SV_FIFO_FLAG_PULLDOWN
- #define SV_FIFO_FLAG_REPEAT_2TIMES
- #define SV_FIFO_FLAG_REPEAT_3TIMES
- #define SV_FIFO_FLAG_REPEAT_4TIMES
- #define SV_FIFO_FLAG_REPEAT_MASK
- #define SV_FIFO_FLAG_REPEAT_ONCE
- #define SV_FIFO_FLAG_SETAUDIOSIZE
- #define SV_FIFO_FLAG_STORAGEMODE
- #define SV_FIFO_FLAG_STORAGENOAUTOCENTER
- #define SV_FIFO_FLAG_TIMEDOPERATION
- #define SV_FIFO_FLAG_VIDEO_B
- #define SV_FIFO_FLAG_VIDEOONLY
- #define SV_FIFO_FLAG_VSYNCWAIT
- #define SV_OPTION_DROPMODE
- #define SV_OPTION_WATCHDOG_ACTION
- #define SV_OPTION_WATCHDOG_TIMEOUT

Typedefs

- typedef void * sv_fifo

Functions

- int sv_fifo_anc (sv_handle *sv, sv_fifo *pfifo, sv_fifo_buffer *pbuffer, sv_fifo_ancbuffer *panc)
- int sv_fifo_ancdata (sv_handle *sv, sv_fifo *pfifo, unsigned char *buffer, int buffersize, int *pcount)
- int sv_fifo_anclayout (sv_handle *sv, sv_fifo *pfifo, char *description, int size, int *required)
- int sv_fifo_bypass (sv_handle *sv, sv_fifo *pfifo, sv_fifo_buffer *pbuffer, int video, int audio)
- int sv_fifo_configstatus (sv_handle *sv, sv_fifo *pfifo, sv_fifo_configinfo *pconfig)
- int sv_fifo_dmarectangle (sv_handle *sv, sv_fifo *pfifo, int xoffset, int yoffset, int xsize, int ysize, int lineoffset)
- int sv_fifo_free (sv_handle *sv, sv_fifo *pfifo)
- int sv_fifo_getbuffer (sv_handle *sv, sv_fifo *pfifo, sv_fifo_buffer **pbuffer, sv_fifo_bufferinfo *bufferinfo, int flags)
- int sv_fifo_init (sv_handle *sv, sv_fifo **ppfifo, int jack, int bshared, int dma, int flagbase, int nframes)
- int sv_fifo_lut (sv_handle *sv, sv_fifo *pfifo, sv_fifo_buffer *pbuffer, unsigned char *buffer, int buffersize, int cookie, int flags)

- int sv_fifo_matrix (sv_handle *sv, sv_fifo *pfifo, sv_fifo_buffer *pbuffer, unsigned int *pmatrix)
- int sv_fifo_putbuffer (sv_handle *sv, sv_fifo *pfifo, sv_fifo_buffer *pbuffer, sv_fifo_bufferinfo *bufferinfo)
- int sv_fifo_reset (sv_handle *sv, sv_fifo *pfifo)
- int sv_fifo_sanitycheck (sv_handle *sv, sv_fifo *pfifo)
- int sv_fifo_sanitylevel (sv_handle *sv, sv_fifo *pfifo, int level, int version)
- int sv_fifo_start (sv_handle *sv, sv_fifo *pfifo)
- int sv_fifo_startex (sv_handle *sv, sv_fifo *pfifo, int *pwhen, int *pclockhigh, int *pclocklow, int *pspare)
- int sv_fifo_status (sv_handle *sv, sv_fifo *pfifo, sv_fifo_info *pinfo)
- int sv_fifo_stop (sv_handle *sv, sv_fifo *pfifo, int flags)
- int sv_fifo_stopex (sv_handle *sv, sv_fifo *pfifo, int flags, int *pwhen, int *pclockhigh, int *pclocklow, int *pspare)
- int sv_fifo_vsyncwait (sv_handle *sv, sv_fifo *pfifo)
- int sv_fifo_wait (sv_handle *sv, sv_fifo *pfifo)
- int sv_memory_dma (sv_handle *sv, int btomemory, char *memoryaddr, int offset, int memorysize, sv_overlapped *poverlapped)
- int sv_memory_dma_ready (sv_handle *sv, sv_overlapped *poverlapped, int resorg)
- int sv_memory_dmaex (sv_handle *sv, int btomemory, char *memoryaddr, int memorysize, int memoryoffset, int memorylineoffset, int cardoffset, int cardlineoffset, int linesize, int linecount, int spare, sv_overlapped *poverlapped)
- int sv_memory_dmarect (sv_handle *sv, int btomemory, char *memoryaddr, int memorysize, int offset, int xoffset, int yoffset, int xsize, int ysize, int lineoffset, int spare, sv_overlapped *poverlapped)
- int sv_memory_dmax (sv_handle *sv, int btomemory, char *memoryaddr, int offseth, int offsetl, int memorysize, sv_overlapped *poverlapped)
- int sv_memory_frameinfo (sv_handle *sv, int frame, int channel, int *field1addr, int *field1size, int *field2addr, int *field2size)

Define Documentation

#define SV_FIFO_BUFFERINFO_VERSION_1

This define is implemented to distinguish between different versions of the structure *sv_fifo_bufferinfo*. Currently the SDK supports this struct version only. It has to be set at all times.

#define SV_FIFO_FLAG_ANC

In case you want to work on streamed ANC data in the memory, this flag has to be set. It enables the ANC streamer data in the structure *sv_fifo_buffer.anc*. The data will be passed to this buffer only when the define `SV_OPTION_ANCCOMPLETE` is set to `SV_ANCCOMPLETE_STREAMER`.

#define SV_FIFO_FLAG_AUDIOINTERLEAVED

By setting or not setting this define you can choose whether you want to store all audio channels multiplexed in one buffer or channel-wise in stereo pairs. For information how to store the audio buffer in the *sv_fifo_buffer* structure see the flag `SV_FIFO_FLAG_NODMAADDR`.

Flag is set (multiplexed):

When setting this define, all audio channels are stored multiplexed in one buffer (character = channel pair, number = sample index):

```
A0B0C0D0E0F0G0H0 A1B1C1D1E1F1G1H1 A2B2C2D2E2F2G2H2 ...
```

If this flag is used together with the flag SV_FIFO_FLAG_NODMAADDR, all audio data will be stored in the first buffer of the array (`pbuffer->audio[0].addr[0]`). Furthermore, you have to adjust the *size* element in the *sv_fifo_buffer* structure to a multiple of its initial value, because as a standard it provides the size for a single stereo pair only.

Flag is not set (channel-wise):

If this flag is not set, the audio channels will be stored channel-wise in stereo pairs one after another (character = channel pair, number = sample index):

Stereo channel 0: `A0A1A2A3A4A5A6A7...`

Stereo channel 1: `B0B1B2B3B4B5B6B7...`

...

#define SV_FIFO_FLAG_AUDIOONLY

This define selects a transfer of audio only (DMA FIFO).

#define SV_FIFO_FLAG_CLOCKEDOPERATION

This define performs the same operation as the define SV_FIFO_FLAG_TIMEDOPERATION, except that the clock is used instead of the vertical sync timestamp.

#define SV_FIFO_FLAG_DMARECTANGLE

This define enables the usage of the DMA rectangle DMA scatter/gather code. This works in combination with the function *sv_fifo_dmarectangle()*.

#define SV_FIFO_FLAG_DONTBLOCK

This flag can be used in conjunction with the function *sv_fifo_getbuffer()*. With it the function will return immediately when no buffer is available or no raster can be detected at the input(s). Then your application has to sleep on its own account to avoid a high CPU usage.

#define SV_FIFO_FLAG_FIELD

This define initiates a field-based operation for the FIFO. Usually, the *sv_fifo_getbuffer()* / *sv_fifo_putbuffer()* pairs have a cycle duration that lasts a whole frame. It can be changed by setting this flag. Then the cycle duration will be one field only, i.e. just one field within the FIFO buffer will be used.

The field that is valid for the cycle is determined on the basis of the tick value of the FIFO buffer. In case the FIFO buffer returns an even tick, the valid field will be field 1, while an odd tick sets field 2 as the valid field.

For a display operation the flag can be specified when calling *sv_fifo_getbuffer()*. But because a capturing starts before the first *sv_fifo_getbuffer()* / *sv_fifo_putbuffer()* pair is performed, for a record operation it has to be set beforehand in the *sv_fifo_init()* call.

Note:

This flag will be overruled when the flag SV_FIFO_FLAG_PULLDOWN is set. Furthermore, this flag has no effect in progressive video rasters. The *counter*, *dmaloop* and *dpxio* example programs demonstrate how to use it (see chapter Example Projects Overview).

It is not possible to use this flag when the video mode is set to SV_MODE_STORAGE_FRAME.

#define SV_FIFO_FLAG_FLUSH

This flag resets the getbuffer/putbuffer pointers to the last issued display or record buffer. This discards any older buffers except the last one. In combination with the function *sv_fifo_stop()* this flag discards all buffers that are pending for a display.

#define SV_FIFO_FLAG_NO_LIVE

When only an input FIFO is active, you can deactivate the automatic live mode with this define. It may be useful in environments where the recorded images must be returned right away. The automatic live mode that is switched on automatically during a record operation delays slightly the return of the recorded image. When an output FIFO is active, the live mode is deactivated anyway.

#define SV_FIFO_FLAG_NODMA

This define disables and skips the DMA transfer for the current frame (DMA FIFO).

#define SV_FIFO_FLAG_NODMAADDR

By setting or not setting this define you can choose whether you want to use a combined buffer or separate buffers for the storing of audio and video data.

Flag is set (separate buffers):

When setting this define, the video and audio data will be stored in separate buffers. With this, you have to put the address of each buffer into their corresponding elements of the *sv_fifo_buffer* structure. This can be done for the *size* elements of the buffers as well, but is not mandatory because they are pre-filled by default:

- Address for video: `pbuffer->video[n].addr`
- Address for audio: `pbuffer->audio[n].addr[m]`
- Size for video: `pbuffer->video[n].size`
- Size for audio: `pbuffer->audio[n].size`

The entries in *pbuffer->dma.addr* and *pbuffer->dma.size* will not be used in this case.

Flag is not set (combined buffer):

If this flag is not set, the complete data (audio and video) will be stored in one buffer. Then you have to put the address of the audio/video buffer into its corresponding element of the *sv_fifo_buffer* structure. This can be done for the *size* element of the buffer as well, but is not mandatory because it is pre-filled by default:

- Address of the buffer: `pbuffer->dma.addr`
- Size of the buffer: `pbuffer->dma.size`

In case you want to address different parts of the buffer, you will find the required offsets in:

- Offset for video: `pbuffer->video[n].addr`
- Offset for audio: `pbuffer->audio[n].addr[m]`

Note:

For information about the way video is stored in the buffer (field vs. frame mode) see chapter Info – Storage Formats.

For information about the way audio is stored in the buffer see the define SV_FIFO_FLAG_AUDIOINTERLEAVED. Please note that when SV_FIFO_FLAG_AUDIOINTERLEAVED is set, the *size* element will be pre-filled with the size of one stereo pair and must be recalculated by the caller (see SV_FIFO_FLAG_AUDIOINTERLEAVED).

#define SV_FIFO_FLAG_PULLDOWN

This define enables a pulldown operation. The start phase of the pulldown operation can be set with the function *sv_pulldown()*.

For a display it can be specified in the function *sv_fifo_getbuffer()*. For a record it has to be set in the *sv_fifo_init()* call, because a capturing starts before the first *sv_fifo_getbuffer()* / *sv_fifo_putbuffer()* pair is performed.

Note:

This flag overrules the define SV_FIFO_FLAG_FIELD.

#define SV_FIFO_FLAG_REPEAT_2TIMES

For further information about this define see the define SV_FIFO_FLAG_REPEAT_ONCE.

#define SV_FIFO_FLAG_REPEAT_3TIMES

For further information about this define see the define SV_FIFO_FLAG_REPEAT_ONCE.

#define SV_FIFO_FLAG_REPEAT_4TIMES

For further information about this define see the define SV_FIFO_FLAG_REPEAT_ONCE.

#define SV_FIFO_FLAG_REPEAT_MASK

Mask for all values of the define group SV_FIFO_FLAG_REPEAT_<xxx>. See the define SV_FIFO_FLAG_REPEAT_ONCE.

#define SV_FIFO_FLAG_REPEAT_ONCE

The defines of the group SV_FIFO_FLAG_REPEAT_<xxx> specify that a frame should be repeated *n* times during a play-out. For example, to play out 24p material at 72p specify the define SV_FIFO_FLAG_REPEAT_3TIMES. However, they will not work for a record operation and generate an error if specified for such an operation. For an example take a look at the *dpxio* example program with the parameter *-x* (see chapter Example Projects Overview).

Note:

The same effect can be achieved by using the *when* parameter of the flag SV_FIFO_FLAG_TIMEDOPERATION.

See also:

The defines SV_FIFO_FLAG_REPEAT_2TIMES, SV_FIFO_FLAG_REPEAT_3TIMES, SV_FIFO_FLAG_REPEAT_4TIMES, and SV_FIFO_FLAG_REPEAT_MASK.

#define SV_FIFO_FLAG_SETAUDIOSIZE

This define changes the audio buffer size for an output FIFO. With it you can modify the audio buffer sizes slightly which will be useful, for example, when playing out audio material with a different audio data distribution. This define cannot be used during an input FIFO.

#define SV_FIFO_FLAG_STORAGEMODE

With this flag it is possible to change the storage format dynamically. The *pbuffer->storage* elements in the *sv_fifo_buffer* structure define the new storage mode and size to be used. The example program `cmodetst` provides an example how to use this functionality (see chapter Example Projects Overview). This flag is not supported for a record operation.

#define SV_FIFO_FLAG_STORAGENOAUTOCENTER

This flag can be used in combination with the define `SV_FIFO_FLAG_STORAGEMODE`. It causes the driver to use the *xoffset* and *yoffset* values from the *pbuffer->storage* substructure. Without this flag the driver automatically centers the image in the buffer.

#define SV_FIFO_FLAG_TIMEDOPERATION

Performs a timed operation with the *when* value of the *sv_fifo_bufferinfo* structure. If no *sv_fifo_bufferinfo* structure is supplied, the *sv_fifo_getbuffer()* function returns `SV_ERROR_PARAMETER`. The value *when* in the *sv_fifo_bufferinfo* structure is used to correlate a frame/sequence with a specific vertical sync timestamp. This can be used together with the function *sv_vtrcontrol()* to perform a VTR synchronized edit or play-out.

See also:

The define `SV_FIFO_FLAG_CLOCKEDOPERATION`.

#define SV_FIFO_FLAG_VIDEO_B

This define activates the mixer functionality of the FIFO when setting it in the function *sv_fifo_getbuffer()* (parameter *flags*). With it you can mix (merge) two images stored in the video board buffer (practically speaking it mixes two outputs). Due to the fact that this functionality uses the alpha channel (key) in the second image, the storage mode must be set to either YUVA or RGBA. The color space in the memory has to be the same as used on the SDI output.

You have to pass the two pointers of the images to the structure *sv_fifo_buffer* (DMA FIFO, for the second image use *sv_fifo_buffer.video_b*). By calling the function *sv_fifo_putbuffer()* the images are transferred automatically and the merging is applied.

Note:

There is another more flexible approach to apply a mixing. It uses the multi-channel operation mode and two independent FIFO API instances. See the define `SV_OPTION_ALPHAMIXER` for further information.

This functionality is available for Centaurus II only.

#define SV_FIFO_FLAG_VIDEOONLY

This flag selects a transfer of video only (DMA FIFO).

#define SV_FIFO_FLAG_VSYNCWAIT

This define waits for the next vertical sync at the input or output and returns the control to the program flow after the vertical sync has occurred. It performs the same operation as the function *sv_fifo_vsyncwait()*.

#define SV_OPTION_DROPMODE

This define determines the immediate behavior of the DVS hardware when the output FIFO API drops a frame:

- `SV_DROPMODE_REPEAT` – The last frame is repeated (default behavior of the DVS hardware).
- `SV_DROPMODE_BLACK` – A black frame is inserted.

See also:

The define `SV_OPTION_WATCHDOG_ACTION`.

#define SV_OPTION_WATCHDOG_ACTION

This define determines the behavior of the DVS hardware when the output FIFO API drops a frame. When called, it will override the behavior determined by the define `SV_OPTION_DROPMODE`. Compared to `SV_OPTION_DROPMODE` this define offers you more possibilities to determine the sent out image. It can also be used to set a time-delayed behavior in conjunction with the define `SV_OPTION_WATCHDOG_TIMEOUT`. Then use the define `SV_OPTION_DROPMODE` to determine the immediate behavior of the DVS video board.

- `SV_WATCHDOG_NONE` – Frame repetition (default).
- `SV_WATCHDOG_BYPASS` – Shows the bypass signal.
- `SV_WATCHDOG_BLACK` – Shows a black image.
- `SV_WATCHDOG_COLORBAR` – Shows a color bar image.

See also:

The defines `SV_OPTION_DROPMODE` and `SV_OPTION_WATCHDOG_TIMEOUT`.

#define SV_OPTION_WATCHDOG_TIMEOUT

This define sets a timeout after which the watchdog should react. The unit is in ticks.

Typedef Documentation

sv_fifo

Void handle to the internal structure describing the FIFO.

Function Documentation

int sv_fifo_anc (sv_handle * *sv*, sv_fifo * *pfifo*, sv_fifo_buffer * *pbuffer*, sv_fifo_ancbuffer * *panc*)

This function is used to transmit and receive user-defined ANC data. You can transfer multiple packets per field and line by calling this function multiple times.

Possible values in the structure *sv_fifo_ancbuffer* regarding field and line number depend on the currently set video raster.

Parameters:

sv – Handle returned from the function *sv_open()*.

pfifo – Handle to the FIFO returned from the function *sv_fifo_init()*.

pbuffer – Current FIFO buffer returned from the function *sv_fifo_getbuffer()*.

panc – Buffer containing the ANC data to be used.

Returns:

If the function succeeds, it returns `SV_OK`. Otherwise it will return the error code `SV_ERROR_<xxx>`.

Note:

To read or write more ANC data than usually included in the default behavior of the SDK you can use the define `SV_OPTION_ANCCOMPLETE` and set it to `SV_ANCCOMPLETE_ON`.

For further information about the structure *sv_fifo_ancbuffer* refer to its description in this document as well as to the comments of the structure in the header file *dvs_fifo.h*.

See also:

The function *sv_fifo_ancdata()*.

int sv_fifo_ancdata (sv_handle * *sv*, sv_fifo * *pfifo*, unsigned char * *buffer*, int *buffersize*, int * *pcount*)

This function is used to transmit and receive user-defined ANC data. With each call you can transfer one packet per field. To transmit in the second field as well issue the command twice between the function calls *sv_fifo_getbuffer()* and *sv_fifo_putbuffer()*. Also see the defines:

- `SV_OPTION_ANCGENERATOR`
- `SV_OPTION_ANCREADER`
- `SV_OPTION_ANCUSER_DID`
- `SV_OPTION_ANCUSER_SDID`
- `SV_OPTION_ANCUSER_LINENR`
- `SV_OPTION_ANCUSER_FLAGS`

To transmit multiple packets of ANC data see the function *sv_fifo_anc()*.

Parameters:

sv – Handle returned from the function *sv_open()*.

pfifo – Handle to the FIFO returned from the function *sv_fifo_init()*.

buffer – Buffer containing the ANC data to be sent or received.

buffersize – Size of the buffer.

pcount – Actual number of bytes received for the input FIFO.

Returns:

If the function succeeds, it returns `SV_OK`. Otherwise it will return the error code `SV_ERROR_<xxx>`.

See also:

The function *sv_fifo_anc()* and the defines mentioned above.

int sv_fifo_anclayout (sv_handle * *sv*, sv_fifo * *pfifo*, char * *description*, int *size*, int * *required*)

This function is used to determine the layout of the ANC streamer data that is returned within the FIFO buffer *sv_fifo_buffer.anc* when using the flag `SV_FIFO_FLAG_ANC`.

The function returns a plain text buffer containing an XML description of the data layout. It describes in detail which range of the buffer is corresponding to which line in the SDI stream.

The following XML tags exist:

- `<anclayout>` – Top level tag surrounding the complete data.
- `<field fieldnr="#">` – Field tag surrounding each field description.
- `<repeat count="#">` – Repeat tag surrounding an area description. Number of lines in current area.
- `<linenr>#</linenr>` – First line number of current area.
- `<hancsize>#</hancsize>` – Size of the HANC area in bytes (within each line of the current area).
- `<vancsize>#</vancsize>` – Size of the VANC area in bytes (within each line of the current area).

Parameters:

sv – Handle returned from the function *sv_open()*.

pfifo – Handle to the FIFO returned from the function *sv_fifo_init()*.

description – Text buffer for the ANC data layout description.

size – Maximum size for the *description* parameter as allocated by the caller.

required – Required buffer size to retrieve the complete description.

Returns:

If the function succeeds, it returns `SV_OK`. Otherwise it will return the error code `SV_ERROR_<xxx>`. The following specific error codes can be returned:

- `SV_ERROR_BUFFERSIZE` – Will be returned when the description buffer is too small.

Example:

The following shows an example what such an XML description might look like:

```
<?xml version="1.0"?>
<anclayout>
  <field fieldnr="0">
    <repeat count="1">
      <linenr>8</linenr>
      <hancsize>0</hancsize>
      <vancsize>3840</vancsize>
    </repeat>
    <repeat count="12">
      <linenr>9</linenr>
      <hancsize>536</hancsize>
      <vancsize>3840</vancsize>
    </repeat>
  </field>
  <field fieldnr="1">
    <repeat count="1">
      <linenr>570</linenr>
      <hancsize>0</hancsize>
      <vancsize>3840</vancsize>
    </repeat>
    <repeat count="13">
      <linenr>571</linenr>
      <hancsize>536</hancsize>
      <vancsize>3840</vancsize>
    </repeat>
  </field>
</anclayout>
```

int sv_fifo_bypass (sv_handle * *sv*, sv_fifo * *pfifo*, sv_fifo_buffer * *pbuffer*, int *video*, int *audio*)

This function is used to mark specific audio and video channels to be bypassed directly from the input to the output. The channels which are marked by this function for an output FIFO are directly taken from the input FIFO, instead of outputting the data which is given in the output FIFO buffer. This function can only be called inbetween *sv_fifo_getbuffer()* and *sv_fifo_putbuffer()* calls.

Parameters:

sv – Handle returned from the function *sv_open()*.

pfifo – Handle to the FIFO returned from the function *sv_fifo_init()*.

pbuffer – Current FIFO buffer returned from the function *sv_fifo_getbuffer()*.

video – Video bypass mask. A value of one (`1`) means to bypass video.

audio – Audio bypass mask. Each mono channel is represented by one bit. The lowest bit represents the first audio channel. Setting a bit to one (`1`) will cause the corresponding channel to be bypassed.

Returns:

If the function succeeds, it returns `SV_OK`. Otherwise it will return the error code `SV_ERROR_<xxx>`. The following specific error codes can be returned:

- `SV_ERROR_PARAMETER` – Will be returned when called for an input FIFO.
- `SV_ERROR_JACK_NOBYPASS` – Will be returned when the FIFO (jack) does not have a corresponding input jack assigned.
- `SV_ERROR_FIFO_STOPPED` – Will be returned when the corresponding input FIFO is stopped.
- `SV_ERROR_VIDEOMODE` – Will be returned when the corresponding input FIFO raster and storage mode configurations do not match.

Note:

This function call is applicable for an output FIFO only. The corresponding input FIFO needs to be in a running state. The input and output jacks have to be configured to the same raster and storage mode settings.

int sv_fifo_configstatus (sv_handle * *sv*, sv_fifo * *pfifo*, sv_fifo_configinfo * *pconfig*)

This function queries system parameters about the FIFO.

Parameters:

sv – Handle returned from the function *sv_open()*.

pfifo – Handle to the FIFO returned from the function *sv_fifo_init()*. If this is `NULL`, the global values *vbuffersize* and *abuffersize* of the *sv_fifo_configinfo* structure are returned.

pconfig – Pointer to the *sv_fifo_configinfo* structure.

Returns:

If the function succeeds, it returns `SV_OK`. Otherwise it will return the error code `SV_ERROR_<xxx>`.

int sv_fifo_dmarectangle (sv_handle * *sv*, sv_fifo * *pfifo*, int *xoffset*, int *yoffset*, int *xsize*, int *ysize*, int *lineoffset*)

This function specifies a DMA scatter/gather operation that enables an image cut-out on the video data in memory, for example, to replace a part of an image. The settings given by this function call affect all subsequent DMA transfers in the specified FIFO.

Parameters:

sv – Handle returned from the function *sv_open()*.

pfifo – Handle to the FIFO returned from the function *sv_fifo_init()*.

xoffset – X-offset for the rectangle in storage.

yoffset – Y-offset for the rectangle in storage.

xsize – X-size of the cut-out.

ysize – Y-size of the cut-out.

lineoffset – Line to line offset in the cut-out.

Returns:

If the function succeeds, it returns `SV_OK`. Otherwise it will return the error code `SV_ERROR_<xxx>` with, for example:

- `SV_ERROR_NOTIMPLEMENTED` – Will be returned when called under an operating system where this function is not supported.
- `SV_ERROR_NOTFRAMESTORAGE` – Will be returned when called in an interlaced storage mode.

Note:

To use this function the flag `SV_FIFO_FLAG_DMARECTANGLE` must be set when calling the function *sv_fifo_getbuffer()*.

This function only works if the memory storage is in a progressive mode, i.e. the `SV_MODE_STORAGE_FRAME` flag is specified in the video mode setup (see also Info – Storage Formats). If this is not set, the function will return `SV_ERROR_NOTFRAMESTORAGE`.

Any gaps in the scan line (e.g. if the lines in the memory are shorter than the ones on the video board) are added at the end of the scan line: The video will not be centered but left aligned if gaps exist.

int sv_fifo_free (sv_handle * *sv*, sv_fifo * *pfifo*)

This function closes and frees the FIFO. After this call the *pfifo* handle will be invalid.

Parameters:

sv – Handle returned from the function *sv_open()*.

pfifo – Handle to the FIFO returned from the function *sv_fifo_init()*.

Returns:

If the function succeeds, it returns `SV_OK`. Otherwise it will return the error code `SV_ERROR_<xxx>`.

Note:

This function must be called before switching the video raster. If this is not the case, the subsequent behavior cannot be predicted.

int sv_fifo_getbuffer (sv_handle * *sv*, sv_fifo * *pfifo*, sv_fifo_buffer ** *pbuffer*, sv_fifo_bufferinfo * *bufferinfo*, int *flags*)

This function returns a buffer structure containing the image buffer that is already filled by an operation (record or display). This call blocks if the output FIFO is full during display or if there is no image captured during record.

For DMA operations you have to provide a user-level buffer. To control how to insert your buffer into the *sv_fifo_buffer* structure see the flag `SV_FIFO_FLAG_NODMAADDR`.

The element *pbuffer.fifoid* has replaced *pbuffer.id*. Due to the necessity of handling multiple jacks in the FIFO API, this structure element is intended as a random unique ID and is no longer linear counting. The content of this value is from the DVS SDK 3.0 on considered private.

Parameters:

sv – Handle returned from the function *sv_open()*.

pfifo – Handle to the FIFO returned from the function *sv_fifo_init()*.

pbuffer – Pointer to the structure *sv_fifo_buffer* that is returned if the function succeeds.

bufferinfo – Provided pointer to the structure *sv_fifo_bufferinfo* with the associated buffer information. Can be `NULL`. This is used in combination with the flag `SV_FIFO_FLAG_TIMEDOPERATION`.

flags – Bit field for several optional features (logical OR). For possible values see `SV_FIFO_FLAG_<xxx>`.

Returns:

If the function succeeds, it returns `SV_OK`. Otherwise it will return the error code `SV_ERROR_<xxx>` with, for example:

- `SV_ERROR_INPUT_AUDIO_NOAESEBU` – Will be returned when no audio could be recorded from the AES input. Nevertheless, there is still a valid *sv_fifo_buffer* structure returned which contains the recorded video, timecodes, etc. This *sv_fifo_buffer* structure must be returned in a subsequent call to the function *sv_fifo_putbuffer()*.
- `SV_ERROR_INPUT_AUDIO_NOAIV` – Will be returned when no audio could be recorded from the AIV input. Nevertheless, there is still a valid *sv_fifo_buffer* structure returned which contains the recorded video, timecodes, etc. This *sv_fifo_buffer* structure must be returned in a subsequent call to the function *sv_fifo_putbuffer()*.
- `SV_ERROR_NODATA` – Will be returned when no buffer is recorded and the flag `SV_FIFO_FLAG_DONTBLOCK` is used.

int sv_fifo_init (sv_handle * *sv*, sv_fifo ** *ppfifo*, int *jack*, int *bshared*, int *dma*, int *flagbase*, int *nframes*)

This function initializes the FIFO for in- or output video operations. It returns a *pfifo* handle which has to be passed to the following FIFO function calls. The FIFO is allocated to contain *nframes* and is associated to one of the I/O modes (input or output). One FIFO handle has to be opened for each input or output jack. Normally, when using the two default jacks only (in- and output), half the memory should be used for the input and the other half for the output.

Parameters:

sv – Handle returned from the function *sv_open()*.

ppfifo – Pointer returning the handle to the FIFO.

jack – The jack number on which this FIFO should operate. For the default output FIFO it has to be set to zero (`0`), while for a default input FIFO it must be one (`1`). In the DVS SDK 2.<x> this parameter was named *binput* and addressed the two possible FIFOs in the same way but as a boolean. Since the DVS SDK 3.0 one can address more than the two default jacks (see chapter API – Jack API).

bshared – Obsolete. Must be set to zero (`0`).

dma – Sets the mode of the DMA transfer for the function *sv_fifo_putbuffer()* (see list below).

flagbase – Base `SV_FIFO_FLAG_<xxx>` flags that are used until the FIFO is finally closed.

nframes – Maximum number of frames in the FIFO. By setting this value to zero (`0`) the maximum number that is possible will be set (depending on the current memory setup and/or buffer size). It can be adjusted manually but will have an effect only when values less than the maximum number are set. However, then make sure that you choose one that is

high enough because the lower the value, the higher the probability of drops. Additionally, this number should be even, otherwise it will be set to the next lower even number.

Parameters for *dma*:

- `SV_FIFO_DMA_MEMMAP` – A mapped FIFO will be used (obsolete).
- `SV_FIFO_DMA_ON` – Enables the automatic DMA for video and audio.
- `SV_FIFO_DMA_OFF` – Disables the automatic DMA.
- `SV_FIFO_DMA_VIDEO` – Enables the automatic DMA for video.
- `SV_FIFO_DMA_AUDIO` – Enables the automatic DMA for audio.

Returns:

If the function succeeds, it returns `SV_OK`. Otherwise it will return the error code `SV_ERROR_<xxx>`.

Note:

The function *sv_fifo_init()* must be called after switching the video raster.

One FIFO handle has to be opened for each jack. As a minimum one FIFO handle must be opened for an input and one for an output.

The *flags* parameter in the function *sv_fifo_getbuffer()* is combined with the *flagbase* parameter of *sv_fifo_init()* on a per-frame basis. *flags* that can be specified at the function *sv_fifo_getbuffer()* can also be passed here in the parameter *flagbase*.

int sv_fifo_lut (sv_handle * *sv*, sv_fifo * *pfifo*, sv_fifo_buffer * *pbuffer*, unsigned char * *buffer*, int *buffersize*, int *cookie*, int *flags*)

This function is used to apply a 1D or 3D look-up table to a specified FIFO buffer. This way each FIFO buffer can be provided with different LUT data, i.e. you can program frame-synchronized the LUTs anew for each frame.

The *flags* parameter describes the type of LUT (either 1D or 3D). In addition it describes the data layout of the LUT buffer.

The native data layout of a 1D LUT is RGBA32. The components are organized in consecutive blocks of 1024 or 4096 entries each. Each component providing a length of 32 bit results in a 16384-bytes LUT buffer. Only the lower 16 bit of a 32-bit word are used.

There is a secondary 1D LUT type which has a native data layout of RGB16. The components are organized in consecutive blocks of 1024 entries each. Each component providing a length of 16 bit results in a 6144-bytes LUT buffer.

The native data layout of a 3D LUT is BGR16. The components are interleaved. There are 17*17*17 entries. The size of the LUT data normally is BGR * 2 bytes * entries (i.e. $3 * 2 * 17^3$), resulting in 29478 bytes for a 16-bit 3D LUT. For performance reasons, the function always expects a 32768-bytes LUT buffer. However, only the first 29478 bytes will be used, and subsequent bytes will be disregarded. In any case, the values always range from zero to 65535 (`0..65535`).

This function can be called multiple times to provide a 1D and a 3D LUT.

By setting the parameter *buffer* to `NULL` and the parameter *buffersize* to zero (`0`) the specified LUT can be disabled.

Parameters:

sv – Handle returned from the function *sv_open()*.

pfifo – Handle to the FIFO returned from the function *sv_fifo_init()*.

pbuffer – Current FIFO buffer returned from the function *sv_fifo_getbuffer()*.

buffer – Buffer containing the LUT data to be used.

buffersize – Size of the buffer.

cookie – Reserved for future use. It has to be set to zero (0).

flags – Optional flags of SV_FIFO_LUT_<xxx>. See list below. If not used, it has to be set to zero (0).

Parameters for *flags*:

- SV_FIFO_LUT_TYPE_1D_RGBA – The buffer describes a 1D LUT (RGBA32, default) with 1024 entries.
- SV_FIFO_LUT_TYPE_1D_RGBA_4K – The buffer describes a 1D LUT (RGBA32) with 4096 entries.
- SV_FIFO_LUT_TYPE_1D_RGB – The buffer describes a 1D LUT (RGB16).
- SV_FIFO_LUT_TYPE_3D – The buffer describes a 3D LUT (BGR16).

Returns:

If the function succeeds, it returns SV_OK. Otherwise it will return the error code SV_ERROR_<xxx>.

Note:

Currently this function works for an output FIFO only.

The 1D LUT RGB16 and the 3D LUT can be used with Centaurus II only.

The 1D LUT RGBA32 with 4096 entries can be used with Atomix only.

For an example see the *dpxio* example program (see also Example Projects Overview).

int sv_fifo_matrix (sv_handle * *sv*, sv_fifo * *pfifo*, sv_fifo_buffer * *pbuffer*, unsigned int * *pmatrix*)

This function is used to transmit a set of matrix coefficients to a specified FIFO buffer. This way, each FIFO buffer can be provided with a different color space conversion.

The matrix coefficients have the following layout in the matrix buffer:

0:g2y 1:b2y 2:r2y

3:g2u 4:b2u 5:r2u

6:g2v 7:b2v 8:r2v

9:alpha

10:inoffset_r 11:inoffset_g 12:inoffset_b 13:inoffset_alpha

The presentation of the above matrix coefficients and matrix offsets is fixed point float, i.e. a value of 0x10000 means 1.00000. For the matrix offsets a value of 1.00000 represents the full value range (e.g. 1024 in 10 bit video modes).

Parameters:

sv – Handle returned from the function *sv_open()*.

pfifo – Handle to the FIFO returned from the function *sv_fifo_init()*.

pbuffer – Current FIFO buffer returned from the function *sv_fifo_getbuffer()*.

pmatrix – Buffer containing the matrix coefficients.

Returns:

If the function succeeds, it returns SV_OK. Otherwise it will return the error code SV_ERROR_<xxx>.

Note:

Currently this function works for an output FIFO only.

Example:

```
// Uses the matrix to reduce the luma value to 50%.
// Implies that storage and I/O mode are both YUV422.

unsigned int matrix[14];

// Y - Luma
matrix[0] = 0x8000; matrix[1] = 0;       matrix[2] = 0;

// U - Chroma
matrix[3] = 0;      matrix[4] = 0x10000; matrix[5] = 0;

// V - Chroma
matrix[6] = 0;      matrix[7] = 0;       matrix[8] = 0x10000;

// Key
matrix[9]  = 0x10000;

// InOffset
matrix[10] = 0;        // Y
matrix[11] = 0;        // U
matrix[12] = 0;        // V
matrix[13] = 0;        // Key

res = sv_fifo_matrix(sv, poutput, pbuffer, matrix);
if(res != SV_OK)  {
  printf("paint_thread: sv_fifo_matrix() failed = %d '%s'\n",
         res, sv_geterrortext(res));
}
```

int sv_fifo_putbuffer (sv_handle * *sv*, sv_fifo * *pfifo*, sv_fifo_buffer * *pbuffer*, sv_fifo_bufferinfo * *bufferinfo*)

This function releases the buffer to the associated FIFO and queues the buffer for reading again or to be scheduled for an output. If buffers are not released fast enough, the last buffer for the input will be overwritten by the video signal or, in case of an output, the last buffer will be repeated (freeze frame). With a DMA FIFO the automatic DMA transfer is performed inside this function. It will not return until the data has been completely transferred.

Parameters:

sv – Handle returned from the function *sv_open()*.

pfifo – Handle to the FIFO returned from the function *sv_fifo_init()*.

pbuffer – Pointer to the *sv_fifo_buffer* structure.

bufferinfo – Pointer to the *sv_fifo_bufferinfo* structure. Can be `NULL`. If this parameter is given, the function fills the tick and clock values as soon as this specific buffer has been recorded (input FIFO) or when it is about to be displayed (output FIFO).

Returns:

If the function succeeds, it returns `SV_OK`. Otherwise it will return the error code `SV_ERROR_<xxx>` with, for example:

- `SV_ERROR_BUFFER_NOTALIGNED` – Will be returned if called with a storage setup where the line alignment of the hardware is not correct.

See also:

The function *sv_fifo_getbuffer()*.

int sv_fifo_reset (sv_handle * *sv*, sv_fifo * *pfifo*)

This function clears the FIFO and resets all counters to zero (`0`). This will automatically happen if you open a new FIFO with the function *sv_fifo_init()*. The *sv_fifo_reset()* function can be called any time when no buffers are pending from the function *sv_fifo_getbuffer()*, as all pending buffers for input and output will be discarded. The next *sv_fifo_getbuffer()* will get the first buffer in the FIFO. On the output the current frame will be shown for a display FIFO after this

call has been used. The *sv_fifo_reset()* call always starts with the first frame of the memory, thus a dual-operation FIFO can be started synchronously. This function resets all other counters as well, such as the dropped-frame counter.

Parameters:

sv – Handle returned from the function *sv_open()*.

pfifo – Handle to the FIFO returned from the function *sv_fifo_init()*.

Returns:

If the function succeeds, it returns `SV_OK`. Otherwise it will return the error code `SV_ERROR_<xxx>` with, for example:

- `SV_ERROR_FIFOCLOSED` – Will be returned when called for an already closed FIFO.

Note:

It is not recommended to call *sv_fifo_reset()* inside an *sv_fifo_getbuffer()* / *sv_fifo_putbuffer()* pair.

If you have called *sv_fifo_reset()* initially with an empty FIFO to be displayed, the dropped-frame counter will start to increment until data is available in the buffer.

int sv_fifo_sanitycheck (sv_handle * *sv*, sv_fifo * *pfifo*)

This function performs a sanity error check.

This sanity check tells whether the underlying hardware is in a proper operation state to run the FIFO. The same check is also automatically performed when the functions *sv_fifo_start()* and *sv_fifo_getbuffer()* are running. So you may use this function prior to calling *sv_fifo_start()* and *sv_fifo_getbuffer()* to check for a potential error situation.

A preceding call of *sv_fifo_sanitylevel()* specifies the severity level that should be reported.

Parameters:

sv – Handle returned from the function *sv_open()*.

pfifo – Handle to the FIFO returned from the function *sv_fifo_init()*.

Returns:

If the function succeeds, it returns `SV_OK`. Otherwise it will return an error code describing best the error situation. Possible error codes are listed in the documentation of the *sv_fifo_sanitylevel()* function.

int sv_fifo_sanitylevel (sv_handle * *sv*, sv_fifo * *pfifo*, int *level*, int *version*)

This function is used to specify the severity level for error codes returned by the function *sv_fifo_sanitycheck()*. There are four severity levels:

- `SV_FIFO_SANITY_LEVEL_OFF` – No errors will be reported.
- `SV_FIFO_SANITY_LEVEL_FATAL` – Only fatal errors will be reported.
- `SV_FIFO_SANITY_LEVEL_ERROR` – Normal errors as well as fatal errors will be reported.
- `SV_FIFO_SANITY_LEVEL_WARN` – Warnings, normal errors and fatal errors will be reported.

Furthermore, the parameter *version* is used to limit the possible error codes to the currently available set of error codes. If a new driver adds new error codes, they will only be returned by the function *sv_fifo_sanitycheck()* when the version parameter in your application was incremented to the corresponding new version as well. This way you can avoid that your application gets confused by unknown error codes when updating to a new driver. Possible values for the version parameter are for the time being:

- `SV_FIFO_SANITY_VERSION_DEFAULT` – No errors will be reported.

- SV_FIFO_SANITY_VERSION_1 – Current set of error codes (listed below).

Parameters:

sv – Handle returned from the function *sv_open()*.

pfifo – Handle to the FIFO returned from the function *sv_fifo_init()*.

level – Error severity level.

version – Version defining the set of possible error codes.

Returns:

If the function succeeds, it returns SV_OK. Otherwise it will return the error code SV_ERROR_<xxx> with, for example:

- SV_ERROR_SYNC_MISSING – (levels: fatal, version_1) Will be returned when called for an output FIFO and the connected sync signal got lost or the hardware is not properly locked to it. This may, for example, happen when disconnecting the reference signal, when the reference is switched or when it is invalid. Fluctuations in the reference signal (sync signal) must not happen during normal operation, so it is recommended to watch for this error and handle it.

Note:

Changing the severity and version level causes the functions *sv_fifo_start()* and *sv_fifo_getbuffer()* to return additional error messages. You need to make sure that your application properly handles these. To test the error situation in advance use the function *sv_fifo_sanitycheck()* before calling them.

The sanity level and version is stored for each FIFO separately and will be used until the FIFO is closed and opened again by using the *sv_fifo_init()* function.

int sv_fifo_start (sv_handle * *sv*, sv_fifo * *pfifo*)

This function needs to be called to start the FIFO as by default after the function *sv_fifo_init()* or *sv_fifo_reset()* the FIFO is in a halted state. Thus by first issuing an *sv_fifo_getbuffer()* / *sv_fifo_putbuffer()* pair, you can preload an output FIFO. When calling the function *sv_fifo_start()*, the FIFO starts with an input (record) or output (play-out) operation depending on its initialization with *sv_fifo_init()*. The operation actually starts delayed at the next possible occasion, that is at the next field 1 in interlaced or the next frame in progressive video modes.

Parameters:

sv – Handle returned from the function *sv_open()*.

pfifo – Handle to the FIFO returned from the function *sv_fifo_init()*.

Returns:

If the function succeeds, it returns SV_OK. Otherwise it will return the error code SV_ERROR_<xxx> with, for example:

- SV_ERROR_FIFOCLOSED – Will be returned if called for a closed FIFO.
- SV_ERROR_ALREADY_RUNNING – Will be returned if called for an already started FIFO.

See also:

The function *sv_fifo_startex()*.

int sv_fifo_startex (sv_handle * *sv*, sv_fifo * *pfifo*, int * *pwhen*, int * *pclockhigh*, int * *pclocklow*, int * *pspare*)

This function offers the same possibilities as the function *sv_fifo_start()*. Additionally, it calculates and returns the time for the first queued frame, i.e. with this function it is possible to know when prebuffered frames will be displayed.

Parameters:

sv – Handle returned from the function *sv_open()*.

pfifo – Handle to the FIFO returned from the function *sv_fifo_init()*.

pwhen – Pointer to the integer that receives the tick when the first frame will start.

pclockhigh – Pointer to the integer that receives the clock for the MSB (most significant byte) of the first frame to be started.

pclocklow – Pointer to the integer that receives the clock for the LSB (least significant byte) of the first frame to be started.

pspare – Currently not used, has to be `NULL`.

Returns:

If the function succeeds, it returns `SV_OK`. Otherwise it will return the error code `SV_ERROR_<xxx>`. For further details about possible error codes see the function *sv_fifo_start()*.

See also:

The function *sv_fifo_start()*.

int sv_fifo_status (sv_handle * *sv*, sv_fifo * *pfifo*, sv_fifo_info * *pinfo*)

This function queries the number of active buffers and the FIFO size. It can be used to avoid a blocking of the FIFO.

Parameters:

sv – Handle returned from the function *sv_open()*.

pfifo – Handle to the FIFO returned from the function *sv_fifo_init()*.

pinfo – Pointer to the *sv_fifo_info* structure.

Returns:

If the function succeeds, it returns `SV_OK`. Otherwise it will return the error code `SV_ERROR_<xxx>`.

int sv_fifo_stop (sv_handle * *sv*, sv_fifo * *pfifo*, int *flags*)

This function stops an output or input. Default is to halt the output/input and to continue when the start command is sent. Currently, the parameter *flags* supports the value `SV_FIFO_FLAG_FLUSH` only and if it is set, all frames in the FIFO will be discarded.

Parameters:

sv – Handle returned from the function *sv_open()*.

pfifo – Handle to the FIFO returned from the function *sv_fifo_init()*.

flags – Optional flags of `SV_FIFO_FLAG_<xxx>`. See list below. If not used, it has to be set to zero (`0`).

Parameters for *flags*:

- `SV_FIFO_FLAG_FLUSH` – Discards all not yet displayed frames or all recorded frames.

Returns:

If the function succeeds, it returns `SV_OK`. Otherwise it will return the error code `SV_ERROR_<xxx>`.

See also:

The function *sv_fifo_stopex()*.

int sv_fifo_stopex (sv_handle * *sv*, sv_fifo * *pfifo*, int *flags*, int * *pwhen*, int * *pclockhigh*, int * *pclocklow*, int * *pspare*)

This function stops an output or input. Default is to halt the output/input and to continue when the start command is sent. Currently, the parameter *flags* supports the value SV_FIFO_FLAG_FLUSH only and if it is set, all frames in the FIFO will be discarded. Compared to the function *sv_fifo_stop()* you also receive the tick and clock of the last frame sent out. The parameters for tick and clock will be filled with values when halting an output operation only and will be mainly of interest, if you have sent SV_FIFO_FLAG_FLUSH as well.

Parameters:

sv – Handle returned from the function *sv_open()*.

pfifo – Handle to the FIFO returned from the function *sv_fifo_init()*.

flags – Optional flags of SV_FIFO_FLAG_<xxx>. See list below. If not used, it has to be set to zero (0).

pwhen – Pointer to the integer that receives the tick when the last frame is sent out.

pclockhigh – Pointer to the integer that receives the clock for the MSB (most significant byte) of the last frame.

pclocklow – Pointer to the integer that receives the clock for the LSB (least significant byte) of the last frame.

pspare – Currently not used, has to be set to NULL.

Parameters for *flags*:

- SV_FIFO_FLAG_FLUSH – Discards all not yet displayed frames or all recorded frames.

Returns:

If the function succeeds, it returns SV_OK. Otherwise it will return the error code SV_ERROR_<xxx>.

See also:

The function *sv_fifo_stop()*.

int sv_fifo_vsyncwait (sv_handle * *sv*, sv_fifo * *pfifo*)

This function waits for the next vertical sync at the input or output and returns the control to the program flow after the vertical sync has occurred. It performs the same operation as the define SV_FIFO_FLAG_VSYNCWAIT.

Parameters:

sv – Handle returned from the function *sv_open()*.

pfifo – Handle to the FIFO returned from the function *sv_fifo_init()*.

Returns:

If the function succeeds, it returns SV_OK. Otherwise it will return the error code SV_ERROR_<xxx> with, for example:

- SV_ERROR_NOCARRIER – Will be returned if no input is connected.

int sv_fifo_wait (sv_handle * *sv*, sv_fifo * *pfifo*)

This function waits until the last frame is transferred on the output. For an input FIFO this function does nothing and returns immediately. You can also use the *sv_fifo_status()* function for the same purpose, but then the user application has to poll the device.

Parameters:

sv – Handle returned from the function *sv_open()*.

pfifo – Handle to the FIFO returned from the function *sv_fifo_init()*.

Returns:

If the function succeeds, it returns SV_OK. Otherwise it will return the error code SV_ERROR_<xxx> with, for example:

- SV_ERROR_FIFO_TIMEOUT – Will be returned if the operation does not complete within 100 driver ticks.

int sv_memory_dma (sv_handle * *sv*, int *btomemory*, char * *memoryaddr*, int *offset*, int *memorysize*, sv_overlapped * *poverlapped*)

This function performs a DMA (read or write) to a specific memory address in the CPU memory or a specific offset in the DVS video board memory.

Parameters:

sv – Handle returned from the function *sv_open()*.

btomemory – If you want to transfer to the video device's memory, set this parameter to TRUE. In case you want to transfer to the CPU memory, set it to FALSE.

memoryaddr – Memory address in the CPU memory.

offset – Low 32 bits of the offset in the video device's memory.

memorysize – Size of the buffer to read or write.

poverlapped – Overlapped structure for I/O operations. If this is set to NULL, a normal synchronous DMA transfer is done, otherwise the transfer will be performed asynchronously. On UNIX systems this parameter should be NULL.

Returns:

If the function succeeds, it returns SV_OK. Otherwise it will return the error code SV_ERROR_<xxx>.

See also:

The functions *sv_memory_dma_ready()*, *sv_memory_dmax()*, *sv_memory_dmarect()*, and *sv_memory_dmaex()*.

int sv_memory_dma_ready (sv_handle * *sv*, sv_overlapped * *poverlapped*, int *resorg*)

In case you have called the functions *sv_memory_dma()*, *sv_memory_dmax()*, *sv_memory_dmarect()*, or *sv_memory_dmaex()* with overlapped I/O transfers enabled, you can pick up the contents of the memory with this function. It has to be called as soon as the overlapped event gets the signal that the data is ready.

Parameters:

sv – Handle returned from the function *sv_open()*.

poverlapped – Pointer to the structure *sv_overlapped*.

resorg – Original result that was returned by the function *sv_memory_dma()* or the other *sv_memory_dma<xxx>()* functions as mentioned above.

Returns:

If the function succeeds, it returns SV_OK. Otherwise it will return the error code SV_ERROR_<xxx>.

int sv_memory_dmaex (sv_handle * *sv*, int *btomemory*, char * *memoryaddr*, int *memorysize*, int *memoryoffset*, int *memorylineoffset*, int *cardoffset*, int *cardlineoffset*, int *linesize*, int *linecount*, int *spare*, sv_overlapped * *poverlapped*)

This function performs a DMA (read or write) to a specific offset in the storage of the DVS video board or to a specific memory address in the CPU memory. Compared to the function *sv_memory_dma()* it offers more advanced DMA capabilities such as a cut-out and/or stride in the system memory.

Parameters:

sv – Handle returned from the function *sv_open()*.

btomemory – If you want to transfer to the device memory, set this parameter to `TRUE`. In case you want to transfer to the CPU memory, set it to `FALSE`.

memoryaddr – Memory address in the CPU memory.

memorysize – Size of the buffer at *memoryaddr*.

memoryoffset – Offset in the CPU memory. This value is relative to *memoryaddr*.

memorylineoffset – Line offset in the CPU memory in bytes (from the beginning of a line to the beginning of the next line).

cardoffset – Offset in the video device memory.

cardlineoffset – Line offset in the video device memory in bytes (from the beginning of a line to the beginning of the next line).

linesize – Size of each line.

linecount – Number of lines.

spare – Currently not used. It has to be set to zero (`0`).

poverlapped – Overlapped structure for I/O operations. If this is set to `NULL`, a normal synchronous DMA transfer is done, otherwise the transfer will be performed asynchronously. On UNIX systems this parameter should be `NULL`.

Returns:

If the function succeeds, it returns `SV_OK`. Otherwise it will return the error code `SV_ERROR_<xxx>`.

See also:

The functions *sv_memory_dma_ready()*, *sv_memory_dma()*, *sv_memory_dmax()*, and *sv_memory_dmarect()*.

int sv_memory_dmarect (sv_handle * *sv*, int *btomemory*, char * *memoryaddr*, int *memorysize*, int *offset*, int *xoffset*, int *yoffset*, int *xsize*, int *ysize*, int *lineoffset*, int *spare*, sv_overlapped * *poverlapped*)

This function specifies a DMA scatter/gather operation to perform an image cut-out of the video data in the video device memory, for example, to replace a part of an image only.

Parameters:

sv – Handle returned from the function *sv_open()*.

btomemory – If you want to transfer to the video device's memory, set this parameter to `TRUE`. In case you want to transfer to the CPU memory, set it to `FALSE`.

memoryaddr – Memory address in the CPU memory.

memorysize – Size of the buffer at *memoryaddr*.

offset – Offset in the video device memory.

xoffset – X-offset for the image on the board.

yoffset – Y-offset for the image on the board.

xsize – X-size of the image on the board.

ysize – Y-size of the image on the board.

lineoffset – Offset between two lines in the CPU memory in bytes (from the end of a line to the beginning of the next line).

spare – Currently not used. It has to be set to zero (`0`).

poverlapped – Overlapped structure for I/O operations. If this is set to `NULL`, a normal synchronous DMA transfer is done, otherwise the transfer will be performed asynchronously. On UNIX systems this parameter should be `NULL`.

Returns:

If the function succeeds, it returns `SV_OK`. Otherwise it will return the error code `SV_ERROR_<xxx>`.

See also:

The functions *sv_memory_dma_ready()*, *sv_memory_dma()*, *sv_memory_dmax()*, and *sv_memory_dmaex()*.

int sv_memory_dmax (sv_handle * *sv*, int *btomemory*, char * *memoryaddr*, int *offseth*, int *offsetl*, int *memorysize*, sv_overlapped * *poverlapped*)

This function is the 64-bit version of the function *sv_memory_dma()*. It performs a DMA (read or write) to a specific offset in the storage of the DVS video board or to a specific memory address in the CPU memory.

Parameters:

sv – Handle returned from the function *sv_open()*.

btomemory – If you want to transfer to the video device's memory, set this parameter to `TRUE`. In case you want to transfer to the CPU memory, set it to `FALSE`.

memoryaddr – Memory address in the CPU memory.

offseth – High 32 bits of the offset in the video device memory.

offsetl – Low 32 bits of the offset in the video device memory.

memorysize – Size of the buffer to read or write.

poverlapped – Overlapped structure for I/O operations. If this is set to `NULL`, a normal synchronous DMA transfer is done, otherwise the transfer will be performed asynchronously. On UNIX systems this parameter should be `NULL`.

Returns:

If the function succeeds, it returns `SV_OK`. Otherwise it will return the error code `SV_ERROR_<xxx>`.

See also:

The functions *sv_memory_dma_ready()*, *sv_memory_dma()*, *sv_memory_dmaex()*, and *sv_memory_dmarect()*.

int sv_memory_frameinfo (sv_handle * *sv*, int *frame*, int *channel*, int * *field1addr*, int * *field1size*, int * *field2addr*, int * *field2size*)

This function retrieves information about a frame for memory operations. Audio and video have to be selected by inserting `SV_PRESET_<xxx>` defines into the *channel* parameter. Please note that the values for address and size fields differ slightly depending on the set storage mode (see chapter Info – Storage Formats).

Parameters:

sv – Handle returned from the function *sv_open()*.

frame – Frame number of the frame that you want to retrieve information about.

channel – SV_PRESET_<xxx> defines. See list below. Combine these values to set the active channels.

field1addr – Field storage mode: Offset (memory address) to the start of field 1. Frame storage mode: Offset (memory address) to the start of the first line of the frame.

field1size – Field storage mode: Size of the active video area of field 1 in bytes. Frame storage mode: Size of the buffer for the complete frame in bytes.

field2addr – Field storage mode: Offset (memory address) to the start of field 2. Frame storage mode: Offset (memory address) to the start of the second line of the frame.

field2size – Field storage mode: Size of the active video area of field 2 in bytes. Frame storage mode: Will be zero (0).

Parameters for *channel*:

- SV_PRESET_VIDEO – Video channel.
- SV_PRESET_KEY – Key channel.
- SV_PRESET_AUDIO12 – First audio channel pair.
- SV_PRESET_AUDIO34 – Second audio channel pair.
- SV_PRESET_AUDIO56 – Third audio channel pair.
- SV_PRESET_AUDIO78 – Fourth audio channel pair.
- SV_PRESET_AUDIO9a – Fifth audio channel pair.
- SV_PRESET_AUDIObc – Sixth audio channel pair.
- SV_PRESET_AUDIOde – Seventh audio channel pair.
- SV_PRESET_AUDIOfg – Eighth audio channel pair.
- SV_PRESET_TIMECODE – Timecode/header.

Returns:

If the function succeeds, it returns SV_OK. Otherwise it will return the error code SV_ERROR_<xxx>.

Note:

This function can interact with the FIFO API. The FIFO/frame ID of the buffer (i.e. the element *sv_fifo_buffer.fifoid*) can be fed into the *frame* parameter of this function. This way one can perform the DMA manually instead of using the automatic DMA of the FIFO API.

API – Direct API

Detailed Description

The Direct API is a low-latency I/O interface for real-time capture and play-out. It is not intended to be used in conjunction with the FIFO API.

In particular this API incorporates an integration of the following third-party GPU interfaces:

- AMD FirePro™ SDI-Link
- NVidia GPUDirect™

The integration with the above interfaces is realized via specialized DVSOEM libraries that communicate directly with the corresponding third-party GPU driver. The following DVSOEM libraries exist:

- *dvsoemgpu* – Supports both the AMD and NVidia integration.
- *dvsoemamd* – Supports the AMD integration only.
- *dvsoemnv* – Supports the NVidia integration only.

You can link your application to either one of these libraries to use the respective GPU interface with the Direct API. Of course, the Direct API can also be used with the regular DVSOEM library alone to handle standard system memory buffers with a low latency (native operation mode of the Direct API).

The third-party GPU drivers are not part of the DVS SDK and have to be obtained from the manufacturer directly.

For each jack of the DVS device you can open a Direct API instance. This way it is possible to implement applications which pass the video data from input to output with video processing via the GPU inbetween. When limiting the number of used buffers for each Direct API instance (jack) to one (`1`), it is possible to achieve an In-to-Out delay of two frames. When using the field-based mode in interlaced rasters, you can even achieve an In-to-Out delay of one frame (realized as a two-field delay).

The concept of the Direct API is to allocate a specific number of buffers which are later addressed via an index in all follow-up function calls. To such a buffer index the caller can bind various video objects like buffer addresses or OpenGL textures. Optionally, the caller can bind a timecode structure to the buffer index as well. Each video object can thus be scheduled for play-out or filled with recorded data.

Data Structures

- struct sv_direct_bufferinfo
- struct sv_direct_info
- struct sv_direct_timecode

Defines

- #define SV_DIRECT_FLAG_DISCARD
- #define SV_DIRECT_FLAG_FIELD

Functions

- int sv_direct_bind_buffer (sv_handle *sv, sv_direct_handle *dh, int bufferindex, char *addr, int size)

- int sv_direct_bind_opengl (sv_handle *sv, sv_direct_handle *dh, int bufferindex, GLuint texture)
- int sv_direct_bind_timecode (sv_handle *sv, sv_direct_handle *dh, int bufferindex, sv_direct_timecode *ptc)
- int sv_direct_display (sv_handle *sv, sv_direct_handle *dh, int bufferindex, int flags, sv_direct_bufferinfo *pinfo)
- int sv_direct_free (sv_handle *sv, sv_direct_handle *dh)
- int sv_direct_init (sv_handle *sv, sv_direct_handle **pdh, char *mode, int jack, int buffercount, int flags)
- int sv_direct_record (sv_handle *sv, sv_direct_handle *dh, int bufferindex, int flags, sv_direct_bufferinfo *pinfo)
- int sv_direct_status (sv_handle *sv, sv_direct_handle *dh, sv_direct_info *pinfo)
- int sv_direct_sync (sv_handle *sv, sv_direct_handle *dh, int bufferindex, int flags)
- int sv_direct_unbind (sv_handle *sv, sv_direct_handle *dh, int bufferindex)

Define Documentation

#define SV_DIRECT_FLAG_DISCARD

This flag can only be passed to the functions *sv_direct_display()* and *sv_direct_record()*.

For the function *sv_direct_record()* it discards all already recorded buffers and lets the function wait for the next buffer to be ready.

For the function *sv_direct_display()* it discards all already queued buffers and lets the function queue the current buffer of this call immediately. The new buffer will be scheduled for the earliest possible output and the function will return immediately.

The effect of this flag differs slightly for record and display directions: For record it discards buffers recorded by the hardware and for display it discards buffers that the caller has previously passed to the driver.

Discarding all buffers in record direction means that the function *sv_direct_record()* will block until the next vertical sync. In display direction it means that the function *sv_direct_display()* is able to take a new buffer from the user immediately without blocking.

#define SV_DIRECT_FLAG_FIELD

This flag can only be passed to the function *sv_direct_init()*.

It enables the Direct API to handle frames of an interlaced raster on a per field basis. Each field is then handled by a separate call of either of the functions *sv_direct_display()* or *sv_direct_record()*.

Note:

Currently this operation mode is not available when the storage is configured to `SV_MODE_STORAGE_FRAME`. Storing the fields in separate buffers is mandatory for a field-based operation when using the Direct API.

Function Documentation

int sv_direct_bind_buffer (sv_handle * *sv*, sv_direct_handle * *dh*, int *bufferindex*, char * *addr*, int *size*)

This function is one of the binding functions for video. It binds a specific buffer address and size within the application's memory space to the given buffer index.

Each buffer index can only have one binding for video. Before using any of the *sv_direct_bind_<xxx>()* binding functions for video again, you have to call the function *sv_direct_unbind()* first. Otherwise the error code `SV_ERROR_DIRECT_BUFFER_ALREADY_BOUND` will be returned.

The buffer address which is bound here has to be valid and accessible when performing a call to the function *sv_direct_display()* or *sv_direct_record()*.

Parameters:

sv – Handle returned from the function *sv_open()*.

dh – Handle to the Direct API instance returned from the function *sv_direct_init()*.

bufferindex – The buffer index to which the buffer address and size should be bound.

addr – Buffer address which should be bound to the given buffer index.

size – Buffer size which should be bound to the given buffer index.

Returns:

If the function succeeds, it returns `SV_OK`. Otherwise it will return the error code `SV_ERROR_<xxx>`.

int sv_direct_bind_opengl (sv_handle * *sv*, sv_direct_handle * *dh*, int *bufferindex*, GLuint *texture*)

This function is one of the binding functions for video. It binds a specific OpenGL texture to the given buffer index. To use this function you have to initialize the Direct API instance either with `"NVIDIA/OPENGL"` or `"AMD/OPENGL"` in the function *sv_direct_init()*.

Each buffer index can only have one binding for video. Before using any of the *sv_direct_bind_<xxx>()* binding functions for video again, you have to call the function *sv_direct_unbind()* first. Otherwise the error code `SV_ERROR_DIRECT_BUFFER_ALREADY_BOUND` will be returned.

The OpenGL texture which is bound here has to be valid and accessible when performing a call to the function *sv_direct_display()* or *sv_direct_record()*.

Parameters:

sv – Handle returned from the function *sv_open()*.

dh – Handle to the Direct API instance returned from the function *sv_direct_init()*.

bufferindex – The buffer index to which the OpenGL texture should be bound.

texture – The OpenGL texture which should be bound to the given buffer index.

Returns:

If the function succeeds, it returns `SV_OK`. Otherwise it will return the error code `SV_ERROR_<xxx>`.

int sv_direct_bind_timecode (sv_handle * *sv*, sv_direct_handle * *dh*, int *bufferindex*, sv_direct_timecode * *ptc*)

This function binds a specific *sv_direct_timecode* structure to the given buffer index.

Each buffer index can only have one timecode binding. Before using this function again, you have to call the function *sv_direct_unbind()* first. Otherwise the error code `SV_ERROR_DIRECT_TIMECODE_ALREADY_BOUND` will be returned.

The *sv_direct_timecode* structure which is bound here has to be valid and accessible when performing a call to the function *sv_direct_display()* or *sv_direct_record()*. Before passing the `ptc` parameter its value `ptc->size` has to be initialized to `sizeof(sv_direct_timecode)` by the caller.

Calling this function is optional and only needed if a record or play-out of timecode is required.

Parameters:

sv – Handle returned from the function *sv_open()*.

dh – Handle to the Direct API instance returned from the function *sv_direct_init()*.

bufferindex – The buffer index to which the timecode structure should be bound.

ptc – Pointer to the *sv_direct_timecode* structure.

Returns:

If the function succeeds, it returns `SV_OK`. Otherwise it will return the error code `SV_ERROR_<xxx>`.

Note:

Due to technical reasons there is a static timecode shift compared to video on an output instance of the Direct API: In interlaced rasters each timecode is two fields late. In progressive rasters each timecode is two frames late. It can be counteracted by passing a timecode for buffer `n` in buffer `n-1` for interlaced rasters and in buffer `n-2` for progressive rasters.

int sv_direct_display (sv_handle * *sv*, sv_direct_handle * *dh*, int *bufferindex*, int *flags*, sv_direct_bufferinfo * *pinfo*)

As soon as an output instance is initialized with the function *sv_direct_init()*, a play-out is already started and it waits for new buffers to be queued. This function transfers and schedules the video and timecode data of the objects of the specified buffer index for play-out.

If there is already a buffer queued, this function will block until the next vertical sync. Then the queued buffer will be displayed.

Before passing the `pinfo` parameter its value `pinfo->size` has to be initialized to `sizeof(sv_direct_bufferinfo)` by the caller.

The values returned in the `pinfo` structure are basically the vertical sync tick and internal microseconds clock values at which the buffer will be scanned out. To have as much time as possible for GPU processing, it is possible to call this function very shortly (about 2 ms) before the vertical sync at which the buffer actually should be scanned out. If it is called too late, the buffer will be dropped thus avoiding tearing-artifacts. The values `pinfo->dma.clock_go_*` and `pinfo->dma.clock_ready_*` specify the absolute times (microseconds) when the DMA to the DVS device was started and when it was ready. Comparing these values with `pinfo->clock_*` may give the caller an indication how much later to call this function without causing any drops. Generally speaking `pinfo->dma.clock_go_*` has to occur before `pinfo->clock_*` while the remaining DMA transfer (until `pinfo->dma.clock_ready_*`) may overlap into the next vertical sync slot.

Parameters:

sv – Handle returned from the function *sv_open()*.

dh – Handle to the Direct API instance returned from the function *sv_direct_init()*.

bufferindex – The buffer index from which the play-out data should be transferred.

flags – `SV_DIRECT_FLAG_<xxx>` flags that are used only for this specific buffer index.

pinfo – Pointer to the *sv_direct_bufferinfo* structure (optional).

Returns:

If the function succeeds, it returns `SV_OK`. Otherwise it will return the error code `SV_ERROR_<xxx>`. The error `SV_ERROR_VSYNCPASSED` occurs when calling this function too late (too short before or even after the vertical sync).

int sv_direct_free (sv_handle * *sv*, sv_direct_handle * *dh*)

This function closes and frees a Direct API instance. After this call the *dh* handle will be invalid.

Parameters:

sv – Handle returned from the function *sv_open()*.

dh – Handle to the Direct API instance returned from the function *sv_direct_init()*.

Returns:

If the function succeeds, it returns `SV_OK`. Otherwise it will return the error code `SV_ERROR_<xxx>`.

Note:

This function must be called before switching the video raster. If this is not the case, the subsequent behavior cannot be predicted.

int sv_direct_init (sv_handle * *sv*, sv_direct_handle ** *pdh*, char * *mode*, int *jack*, int *buffercount*, int *flags*)

This function initializes the Direct API for in- or output video operations. It returns a *dh* handle which has to be passed to subsequent Direct API function calls.

One Direct API instance has to be opened for each input or output jack. The Direct API allocates *buffercount* independent buffers to be used in each of the Direct API instances. It is possible to use only a single buffer for the lowest possible latency in your application.

The following different operation modes are available for the Direct API. The operation mode has to be specified as a character string for the *mode* parameter:

- `"DVS"` – Uses the Direct API as a low-latency I/O interface for the regular DVSOEM library. This functionality is also available in any of the other DVSOEM libraries of the Direct API.
- `"NVIDIA/OPENGL"` – Activates the usage of the NVidia GPUDirect interface.
- `"AMD/OPENGL"` – Activates the usage of the AMD FirePro SDI-Link interface.

Parameters:

sv – Handle returned from the function *sv_open()*.

pdh – Pointer returning the handle to the Direct API instance.

mode – Operation mode of the Direct API instance (for possible values see above).

jack – The number of the jack where this Direct API instance should operate.

buffercount – Maximum number of independent buffers.

flags – Base `SV_DIRECT_FLAG_<xxx>` flags that are used until the Direct API instance is finally closed.

Returns:

If the function succeeds, it returns `SV_OK`. Otherwise it will return the error code `SV_ERROR_<xxx>`.

Note:

While a Direct API instance is open, a switching of the video raster may lead to unpredictable behavior. It has to be closed with the function *sv_direct_free()* before switching the video raster for a particular jack.

int sv_direct_record (sv_handle * *sv*, sv_direct_handle * *dh*, int *bufferindex*, int *flags*, sv_direct_bufferinfo * *pinfo*)

As soon as an input instance is initialized with the function *sv_direct_init()*, a record is already started. This function transfers the latest recorded video and timecode data to the objects of a specified buffer index. It will block until the next vertical sync in case there is no recorded buffer available.

Before passing the `pinfo` parameter its value `pinfo->size` has to be initialized to `sizeof(sv_direct_bufferinfo)` by the caller.

The values returned in the `pinfo` structure are basically the vertical sync tick and internal microseconds clock values at which the buffer has started recording. To have as much time as possible for GPU processing, this function should be called slightly before the vertical sync at which the buffer actually finishes recording. If it is called too late (beyond the next vertical sync), the buffer will be dropped. The values `pinfo->dma.clock_go_*` and `pinfo->dma.clock_ready_*` specify the absolute times (microseconds) when the DMA from the DVS device was started and when it was ready. Comparing these values with `pinfo->clock_*` may give you an indication whether this function should have been called earlier.

Parameters:

sv – Handle returned from the function *sv_open()*.

dh – Handle to the Direct API instance returned from the function *sv_direct_init()*.

bufferindex – The buffer index to which the recorded data should be transferred.

flags – `SV_DIRECT_FLAG_<xxx>` flags that are used only for this specific buffer index.

pinfo – Pointer to the *sv_direct_bufferinfo* structure (optional).

Returns:

If the function succeeds, it returns `SV_OK`. Otherwise it will return the error code `SV_ERROR_<xxx>`. Especially the errors `SV_ERROR_NOCARRIER`, `SV_ERROR_INPUT_VIDEO_DETECTING`, `SV_ERROR_INPUT_VIDEO_NOSIGNAL`, `SV_ERROR_INPUT_VIDEO_RASTER`, `SV_ERROR_INPUT_KEY_NOSIGNAL`, and `SV_ERROR_INPUT_KEY_RASTER` may occur in case there is a signal loss during operation. The error `SV_ERROR_VSYNCPASSED` occurs when calling this function too late (too short before or even after the vertical sync).

int sv_direct_status (sv_handle * *sv*, sv_direct_handle * *dh*, sv_direct_info * *pinfo*)

This function queries the number of available and dropped buffers of a particular Direct API instance. Knowing the amount of available buffers can be used to avoid a blocking of the functions *sv_direct_display()* and *sv_direct_record()*. Additionally, this function delivers the current tick and clock values.

Before passing the `pinfo` parameter its value `pinfo->size` has to be initialized to `sizeof(sv_direct_info)` by the caller.

Parameters:

sv – Handle returned from the function *sv_open()*.

dh – Handle to the Direct API instance returned from the function *sv_direct_init()*.

pinfo – Pointer to the *sv_direct_info* structure.

Returns:

If the function succeeds, it returns `SV_OK`. Otherwise it will return the error code `SV_ERROR_<xxx>`.

int sv_direct_sync (sv_handle * *sv*, sv_direct_handle * *dh*, int *bufferindex*, int *flags*)

This function is used to synchronize the video data of the specified buffer index with the GPU buffer object also bound to this buffer index. To use this function you have to initialize the Direct API instance either with `"NVIDIA/OPENGL"` or `"AMD/OPENGL"` in the function *sv_direct_init()*. The function cannot be called for native video data which has been bound via the function *sv_direct_bind_buffer()*.

For an input instance of the Direct API this function has to be called after the function *sv_direct_record()* and the GPU buffer object must not be accessed by the GPU before this function returns. For an output instance of the Direct API this function has to be called after the GPU has finished processing the GPU buffer object and before calling the function *sv_direct_display()*.

Parameters:

sv – Handle returned from the function *sv_open()*.

dh – Handle to the Direct API instance returned from the function *sv_direct_init()*.

bufferindex – The buffer index which should be synchronized with the GPU.

flags – `SV_DIRECT_FLAG_<xxx>` flags that are used only for this specific buffer index. Currently there are no flags defined.

Returns:

If the function succeeds, it returns `SV_OK`. Otherwise it will return the error code `SV_ERROR_<xxx>`.

int sv_direct_unbind (sv_handle * *sv*, sv_direct_handle * *dh*, int *bufferindex*)

This function unbinds everything that has been previously bound via any of the functions *sv_direct_bind_<xxx>()*.

Parameters:

sv – Handle returned from the function *sv_open()*.

dh – Handle to the Direct API instance returned from the function *sv_direct_init()*.

bufferindex – The buffer index where all previous bindings should be unbound.

Returns:

If the function succeeds, it returns `SV_OK`. Otherwise it will return the error code `SV_ERROR_<xxx>`.

API – Jack API

Detailed Description

The Jack API is an extension for the FIFO API to use independent I/O streams and/or multiple channels. For each jack one FIFO has to be used.

A jack can be seen as a pipeline for video either incoming or outgoing from the DVS video board. All currently supported video boards provide at least one input pipeline and one output pipeline that can run simultaneously. These two pipelines are the default jacks. However, whether they are totally independent from each other, for example, to be called in different video rasters, depends on the capabilities of the DVS video board product. As a rule older video board products do not provide independent I/Os, whereas most newer DVS video boards offer this feature. With the define SV_QUERY_FEATURE you can query your DVS video board product about the supported features, such as independent I/O or multiple channels.

The following table lists the available pipelines (jacks) and whether they can run independently:

DVS Video Board Product	Input Pipeline	Output Pipeline	Independent I/O
Centaurus II	2 single link or 1 dual link	2 single link or 1 dual link	Yes
Centaurus II LT	1 single link	1 single link or 1 dual link	Yes
Atomix and Atomix LT	2, either single link or dual link (with 3 Gbit/s SDI)	2, either single link or dual link (with 3 Gbit/s SDI)	Yes

The ports to be used for in- or output can be configured as allowed by the DVS video board.

Independent I/O

If your DVS video device supports independent in- and outputs, you can run the available in- and output pipelines totally independent from one another, and thus, for example, in different video rasters and/or color modes.

The following example demonstrates how to configure jacks in different formats:

```
  // Global open of DVS video board
  sv = sv_open("PCI,card:0");

  // Configure single-link output A (jack ID 0)
  res = sv_jack_option_set(sv, 0, SV_OPTION_VIDEOMODE, SV_MODE_SMPTE274_29I |
SV_MODE_COLOR_YUV422);
  res = sv_jack_option_set(sv, 0, SV_OPTION_IOMODE, SV_IOMODE_YUV422);

  // Configure single-link input A (jack ID 1)
  res = sv_jack_option_set(sv, 1, SV_OPTION_VIDEOMODE, SV_MODE_PAL |
SV_MODE_COLOR_YUV422);
  res = sv_jack_option_set(sv, 1, SV_OPTION_IOMODE, SV_IOMODE_YUV422);

  // Perform further configuration work
  ...
```

The DVS SDK uses fixed IDs (indexes) for the jacks:

Single Link:

I/O Port (Jack)	I/O Channel	Jack ID
SDI Out A	0	0
SDI In A	0	1
SDI Out B	1	2
SDI In B	1	3

Dual Link:

I/O Port (Jack)	I/O Channel	Jack ID
SDI Out A/B	0	0
SDI In A/B	0	1

Multi-channel Operation Mode

In multi-channel operation mode you can run more than the two default jacks mentioned above. To get the DVS video board into this mode you have to activate it by calling the define `SV_OPTION_MULTICHANNEL`. Additionally, to test the multi-channel operation mode you can use the *svram* example program with `svram multichannel on` (see chapter Example Projects Overview).

After an activation you will have all jacks that your DVS video board product supports (see table above) at your disposal:

```
  // Continuation of the example above

  // Enable multi-channel mode for input
  res = sv_option_set(sv, SV_OPTION_MULTICHANNEL, SV_MULTICHANNEL_ON);

  // Configure single-link output B (jack ID 2)
  res = sv_jack_option_set(sv, 2, SV_OPTION_VIDEOMODE, SV_MODE_SMPTE274_29I |
SV_MODE_COLOR_YUV422);
  res = sv_jack_option_set(sv, 2, SV_OPTION_IOMODE, SV_IOMODE_YUV422);

  // Configure single-link input B (jack ID 3)
  res = sv_jack_option_set(sv, 3, SV_OPTION_VIDEOMODE, SV_MODE_PAL |
SV_MODE_COLOR_YUV422);
  res = sv_jack_option_set(sv, 3, SV_OPTION_IOMODE, SV_IOMODE_YUV422);

  // Perform further configuration work
  ...
```

Then you can continue with the global handle and use the available jack functions by setting the jack IDs:

```
  // Start THREADs
  ...

  // THREAD output A
  res = sv_fifo_init(sv, &pfifo_out_a, 0, ...);
  ...
  // THREAD output B
  res = sv_fifo_init(sv, &pfifo_out_b, 2, ...);
  ...

  // THREAD input A
  res = sv_fifo_init(sv, &pfifo_in_a, 1, ...);
  ...
  // THREAD input B
  res = sv_fifo_init(sv, &pfifo_in_b, 3, ...);
  ...
```

Multi-channel and Global Functions

In multi-channel operation mode you normally cannot use any of the global functions of the DVS SDK, such as the function *sv_videomode()*, due to the fact that they do not provide any jack parameters. If calling these functions with the multi-channel operation mode activated, they will address all jacks at the same time.

However, there is a possibility to apply global functions to specific jacks. For this you have to close the global handle of the DVS video board and open it again by calling a specific I/O channel (a pair of input and output jacks) with, for example, `sv_open("PCI,card=0,channel=0")` (see also the function *sv_open()*). Afterwards all global SDK functions will use this I/O channel.

The following table shows how to address the channels and IDs that should be used for the respective jacks:

I/O Port (Jack)	I/O Channel	ID of Local Jack
SDI Out A	0	0
SDI In A	0	1
SDI Out B	1	0
SDI In B	1	1

For an example about this take a look into the *counter* example program (see chapter Example Projects Overview).

Multi-channel and Timecodes

Normally, all timecodes are included in the SDI signal and thus available for each I/O channel. However, there are two independent timecodes transmitted via LTC and/or RS-422. To assign these timecodes to another jack than the default jack use the define `SV_OPTION_ASSIGN_LTCA`, or, for the RS-422 timecode, the `SV_OPTION_RS422<xxx>` defines (see e.g. `SV_OPTION_RS422A`).

Defines

- #define SV_OPTION_MULTICHANNEL

Functions

- int sv_jack_memorysetup (sv_handle *sv, int bquery, sv_jack_memoryinfo **info, int njacks, int *pjacks, int flags)
- int sv_jack_option_get (sv_handle *sv, int jack, int option, int *pvalue)
- int sv_jack_option_set (sv_handle *sv, int jack, int option, int value)
- int sv_jack_query (sv_handle *sv, int jack, int query, int param, int *pvalue)
- int sv_jack_status (sv_handle *sv, int jack, sv_jack_info *pinfo)

Define Documentation

#define SV_OPTION_MULTICHANNEL

This define can be used to activate the multi-channel operation mode. When set, you can configure further jacks beyond the two default jacks. Then separate FIFOs can be run on these jacks.

To assign audio channels to I/O channels use the define SV_OPTION_AUDIOAESROUTING.

Values:

- SV_MULTICHANNEL_OFF – Turns off the multi-channel operation mode.
- SV_MULTICHANNEL_ON – Turns on the multi-channel operation mode.
- SV_MULTICHANNEL_DEFAULT – Switches to the default multi-channel operation mode: On Centaurus II the default is 'off' and on Atomix the default is 'on'.
- SV_MULTICHANNEL_INPUT – Turns the multi-channel operation mode on for an input only (Centaurus II only).
- SV_MULTICHANNEL_OUTPUT – Turns the multi-channel operation mode on for an output only (Centaurus II only).

Flags:

- SV_MULTICHANNEL_FLAG_ALLOW_DUALLINK – Allows the requested multi-channel operation mode even when the board is operating in a dual-link I/O mode. Use this flag in conjunction with the SV_MULTICHANNEL_<xxx> values. This flag is available on Centaurus II only.

Note:

By using the multi-channel operation mode on Centaurus II in conjunction with the define SV_OPTION_ALPHAMIXER you can mix (merge) two images available in the video board storage.

On Centaurus II a standard multi-channel operation mode is normally only possible in YUV422 I/O mode, otherwise this call will return SV_ERROR_WRONGMODE. The reason for this is that the two links which are used for dual link are reconfigured as two independent and separate YUV422 links.

However, there is one case on Centaurus II where it may be necessary to make the video board run in a dual-link I/O mode: When using the alpha mixer (SV_OPTION_ALPHAMIXER), both channels go into the mixer hardware separately but have to be sent out as one dual-link SDI stream. This can be achieved with the flag SV_MULTICHANNEL_FLAG_ALLOW_DUALLINK.

See also:

The defines SV_OPTION_AUDIOAESROUTING.

Function Documentation

int sv_jack_memorysetup (sv_handle * *sv*, int *bquery*, sv_jack_memoryinfo ** *info*, int *njacks*, int * *pjacks*, int *flags*)

This function configures the memory utilization for all requested jacks in a single step. Before the first FIFO is opened, the driver has to be told how many jacks the application will use and how much memory to assign to each jack.

There is no automatic memory assignment as only your application can know, how many jacks it requires and how much memory is needed for each of them. The amount of memory is relative to the amount of frames that can be used by a FIFO. It defines the maximum FIFO depth.

This function can only be called as long as no FIFO is opened.

When using this function to set up the memory usage, you may deposit an *sv_jack_memoryinfo* structure for each jack (see header file *dvs_clib.h* for further information about this structure). Set at least one of the elements in the substructure *usage*. If you do not know a specific parameter value, simply set it to zero (0). If you set multiple parameters within the *usage* substructure, the first valid value will be taken for the calculation, while overriding any of the

following. All calculations take place based on the currently configured video and audio mode of the jack.

The elements in the substructure *limit* can be used to limit the size of a frame to a smaller value below the currently adjusted mode setting of the jack. This is useful only when you know that you are using the define SV_FIFO_FLAG_STORAGEMODE and you do not want to waste memory.

When setting the function's *info* parameter to NULL, the memory will simply be divided into equally sized ranges for each requested jack.

When this function is called in query mode, all *info* structure elements are filled by the driver based on the current settings. This is also done when the function is not called in query mode, to give an immediate feedback about the new values.

Parameters:

sv – Handle returned from the function *sv_open()*.

bquery – If set to TRUE, this function will only query the current memory information.

info – Array of pointers to multiple *sv_jack_memoryinfo* structures. Each structure describes the memory usage for each of the requested jacks.

njacks – Number of jacks that the caller wants to use. This parameter also defines the array size of the *info* parameter.

pjacks – Number of jacks that are involved in the current memory setup.

flags – Optional flags (currently not used, set to zero (0)).

Returns:

If the function succeeds, it returns SV_OK. Otherwise it will return an error code that describes the error situation best.

Note:

If this function is not called, the driver will by default assume two jacks (the default jacks) where each jack holds half the memory.

int sv_jack_option_get (sv_handle * *sv*, int *jack*, int *option*, int * *pvalue*)

This function retrieves an SV_OPTION_<xxx> value from the specified jack.

Parameters:

sv – Handle returned from the function *sv_open()*.

jack – Jack ID/index.

option – SV_OPTION_<xxx> define. For possible defines see the corresponding chapters in this reference guide.

pvalue – Pointer to the value to be returned.

Returns:

If the function succeeds, it returns SV_OK. Otherwise it will return the error code SV_ERROR_<xxx>.

See also:

The function *sv_jack_option_set()*.

int sv_jack_option_set (sv_handle * *sv*, int *jack*, int *option*, int *value*)

This function sets an SV_OPTION_<xxx> value for the specified jack.

Parameters:

sv – Handle returned from the function *sv_open()*.

jack – Jack ID/index.

option – `SV_OPTION_<xxx>` define. For possible defines see the corresponding chapters in this reference guide.

value – Value to set for *option*.

Returns:

If the function succeeds, it returns `SV_OK`. Otherwise it will return the error code `SV_ERROR_<xxx>`.

See also:

The function *sv_jack_option_get()*.

int sv_jack_query (sv_handle * *sv*, int *jack*, int *query*, int *param*, int * *pvalue*)

This function retrieves an `SV_QUERY_<xxx>` value from the specified jack.

Parameters:

sv – Handle returned from the function *sv_open()*.

jack – Jack ID/index.

query – `SV_QUERY_<xxx>` define. For possible defines see the corresponding chapters in this reference guide.

param – For DVS internal use only.

pvalue – Pointer to the value to be returned.

Returns:

If the function succeeds, it returns `SV_OK`. Otherwise it will return the error code `SV_ERROR_<xxx>`.

int sv_jack_status (sv_handle * *sv*, int *jack*, sv_jack_info * *pinfo*)

This function retrieves status information about a jack.

Parameters:

sv – Handle returned from the function *sv_open()*.

jack – Jack ID/index (input parameter, set to `-1` if unknown).

pinfo – Pointer to the *sv_jack_info* structure.

Returns:

If the function succeeds, it returns `SV_OK`. Otherwise it will return the error code `SV_ERROR_<xxx>`.

API – The sv_option() Functions

Detailed Description

With the *sv_option_set()* and *sv_option_get()* functions it is possible to set and control most aspects of the DVS video device. The following describes the *sv_option()* functions in detail.

For possible SV_OPTION_<xxx> control codes please refer to the corresponding chapters in this reference guide (e.g. see chapter API – Control Functions for gamma control codes or API – Audio Functions for audio control codes).

Functions

- int sv_option (sv_handle *sv, int code, int value)
- int sv_option_get (sv_handle *sv, int option, int *pvalue)
- int sv_option_menu (sv_handle *sv, int menu, int submenu, int menulabel, char *plabel, int labelsize, int *pvalue, int *pmask, int *spare)
- int sv_option_set (sv_handle *sv, int option, int value)
- int sv_option_setat (sv_handle *sv, int option, int value, int when)

Function Documentation

int sv_option (sv_handle * *sv*, int *code*, int *value*)

This function sets an SV_OPTION_<xxx> value globally, i.e. for in- as well as output. In case you want to use independent I/O, you have to use the Jack API (see chapter API – Jack API). This function performs the same operation as the function *sv_option_set()*.

Parameters:

sv – Handle returned from the function *sv_open()*.

code – SV_OPTION_<xxx> define. For possible defines see the corresponding chapters in this reference guide.

value – Value to set for *code*.

Returns:

If the function succeeds, it returns SV_OK. Otherwise it will return the error code SV_ERROR_<xxx>.

See also:

The functions *sv_option_set()* and *sv_option_get()*.

int sv_option_get (sv_handle * *sv*, int *option*, int * *pvalue*)

This function retrieves an SV_OPTION_<xxx> value.

Parameters:

sv – Handle returned from the function *sv_open()*.

option – SV_OPTION_<xxx> define. For possible defines see the corresponding chapters in this reference guide.

pvalue – Pointer to the value to be returned.

Returns:

If the function succeeds, it returns `SV_OK`. Otherwise it will return the error code `SV_ERROR_<xxx>`.

See also:

The functions *sv_option()* and *sv_option_set()*.

int sv_option_menu (sv_handle * *sv*, int *menu*, int *submenu*, int *menulabel*, char * *plabel*, int *labelsize*, int * *pvalue*, int * *pmask*, int * *spare*)

Dynamic configuration menu support.

This function can be called multiple times while iterating the parameters *menu*, *submenu* and *menulabel*. With each call it gives back a readable string (label), an `SV_OPTION_<xxx>` value (*pvalue*) and a mask which has to be used with this specific `SV_OPTION_<xxx>`. To find out the number of elements in each level the value zero (`0`) will cause this function to return the maximum number of elements in *pvalue*. Also a top-level label is returned in *plabel* in this case. The counting for the *menu*, *submenu* and *menulabel* levels starts at one (`1`) in any case.

Parameters:

sv – Handle returned from the function *sv_open()*.

menu – Menu level.

submenu – Submenu level.

menulabel – Menu label level.

plabel – Pointer to the label text.

labelsize – Size of the buffer to receive the label text.

pvalue – Pointer to the value.

pmask – Pointer to the mask.

spare – Currently not used. It has to be set to zero (`0`).

Returns:

If the function succeeds, it returns `SV_OK`. Otherwise it will return the error code `SV_ERROR_<xxx>`.

int sv_option_set (sv_handle * *sv*, int *option*, int *value*)

This function sets an `SV_OPTION_<xxx>` value globally, i.e. for in- as well as output. In case you want to use independent I/O, you have to use the Jack API (see chapter API – Jack API). This function performs the same operation as the function *sv_option()*.

Parameters:

sv – Handle returned from the function *sv_open()*.

option – `SV_OPTION_<xxx>` define. For possible defines see the corresponding chapters in this reference guide.

value – Value to set for *option*.

Returns:

If the function succeeds, it returns `SV_OK`. Otherwise it will return the error code `SV_ERROR_<xxx>`.

See also:

The functions *sv_option()* and *sv_option_get()*.

int sv_option_setat (sv_handle * *sv*, int *option*, int *value*, int *when*)

This function sets a timecode related SV_OPTION_<xxx> value at a specific tick in the future. It can be used to set specific timecode values for each tick. The *sv_option_setat()* function is the most flexible approach to handle timecodes properly when doing pulldown operations and running at speeds different from one (1). To flush the timecode queue use the function *sv_option_set()* with SV_OPTION_FLUSH_TIMECODE.

This function is limited to the following values for SV_OPTION_<xxx> (in alphabetical order):

- SV_OPTION_AFILM_TC
- SV_OPTION_AFILM_UB
- SV_OPTION_APROD_TC
- SV_OPTION_APROD_UB
- SV_OPTION_DLTC_TC
- SV_OPTION_DLTC_UB
- SV_OPTION_DVITC_TC
- SV_OPTION_DVITC_UB
- SV_OPTION_FILM_TC
- SV_OPTION_FILM_UB
- SV_OPTION_GPI
- SV_OPTION_LTC_TC
- SV_OPTION_LTC_UB
- SV_OPTION_PROD_TC
- SV_OPTION_PROD_UB
- SV_OPTION_VITC_TC
- SV_OPTION_VITC_UB
- SV_OPTION_VTR_INFO
- SV_OPTION_VTR_INFO2
- SV_OPTION_VTR_INFO3
- SV_OPTION_VTR_TC
- SV_OPTION_VTR_UB

Parameters:

sv – Handle returned from the function *sv_open()*.

option – SV_OPTION_<xxx> define. See list above.

value – Value to set for *option*.

when – Tick at which the option/value pair should be set.

Returns:

If the function succeeds, it returns SV_OK. Otherwise it will return the error code SV_ERROR_<xxx>.

Note:

This function is demonstrated in the *dpxio* example program (see chapter Example Projects Overview).

API – The sv_query() Function

Detailed Description

With the *sv_query()* function it is possible to retrieve information about the current status of the DVS video device. The following describes the *sv_query()* function in detail.

For possible `SV_QUERY_<xxx>` control codes please refer to the corresponding chapters in this reference guide (e.g. see chapter API – Control Functions for gamma control codes or API – Audio Functions for audio control codes).

Functions

- int sv_query (sv_handle *sv, int cmd, int par, int *presult)

Function Documentation

int sv_query (sv_handle * *sv*, int *cmd*, int *par*, int * *presult*)

This function queries various board settings and retrieves information about the DVS video device.

Parameters:

sv – Handle returned from the function *sv_open()*.

cmd – `SV_QUERY_<xxx>` define. For possible defines see the corresponding chapters in this reference guide.

par – Optional parameter for *cmd*.

presult – Pointer to the integer that receives the return value.

Returns:

If the function succeeds, it returns `SV_OK`. Otherwise it will return the error code `SV_ERROR_<xxx>`.

API – Control Functions

Detailed Description

This chapter describes various control functions to control, for example, zooming and panning or pulldown.

Defines

- #define SV_OPTION_RS422A
- #define SV_OPTION_RS422B
- #define SV_QUERY_PULLDOWN
- #define SV_QUERY_XPANNING
- #define SV_QUERY_XZOOM
- #define SV_QUERY_YPANNING
- #define SV_QUERY_YZOOM
- #define SV_QUERY_ZOOMFLAGS

Functions

- int sv_lut (sv_handle *sv, int command, int *ptable, int lutid)
- int sv_matrix (sv_handle *sv, int matrixmode, sv_matrixinfo *pmatrix)
- int sv_matrixex (sv_handle *sv, int matrixtype, sv_matrixexinfo *pmatrix, sv_matrixexinfo *pquery)
- int sv_zoom (sv_handle *sv, int xzoom, int yzoom, int xpanning, int ypanning, int flags)

Define Documentation

#define SV_OPTION_RS422A

This option call independently configures the physical pin-out, the logical task and the assigned I/O channel (when in a multi-channel environment) of the RS-422 port A.

Possible parameters are:

- `SV_RS422_PINOUT_DEFAULT` – Sets this port to its default pin-out setting which is `SV_RS422_PINOUT_NORMAL`.
- `SV_RS422_PINOUT_NORMAL` – 1:1 pin-out.
- `SV_RS422_PINOUT_SWAPPED` – Reverse pin-out, meaning that the pins 2/8 and 3/7 on the port are swapped.
- `SV_RS422_PINOUT_MASTER` – Available for Atomix only: Switches this port to the master pin-out (remote out), i.e. it performs the same as `SV_RS422_PINOUT_NORMAL`.
- `SV_RS422_PINOUT_SLAVE` – Available for Atomix only: Switches this port to the slave pin-out (remote in), i.e. it performs the same as `SV_RS422_PINOUT_SWAPPED`.
- `SV_RS422_TASK_DEFAULT` – Sets this port to its default task setting which is `SV_RS422_TASK_MASTER`.
- `SV_RS422_TASK_NONE` – No task is assigned to this port, i.e. this port is switched off.
- `SV_RS422_TASK_MASTER` – This port is handled by the master task in the driver.

- SV_RS422_TASK_SLAVE – This port is handled by the slave task in the driver.
- SV_RS422_IOCHANNEL_MASK – Mask for the I/O channel value range.
- SV_RS422_IOCHANNEL_GET(<value>) – Macro that will extract the assigned I/O channel from the value returned by *sv_option_get()*. For example, call sv_option_get(sv, SV_OPTION_RS422A, &value) to get the value of RS-422 port A and then SV_RS422_IOCHANNEL_GET(value) to receive the I/O channel that is assigned to it.
- SV_RS422_IOCHANNEL_SET(<iochannel>). – Macro that will assign the indicated RS-422 task to an I/O channel.

Note:

On Atomix you can use the defines SV_RS422_PINOUT_MASTER and SV_RS422_PINOUT_SLAVE as replacements for SV_RS422_PINOUT_NORMAL and SV_RS422_PINOUT_SWAPPED.

When the port has been already opened by the function *sv_rs422_open()*, this option call will return SV_ERROR_ALREADY_OPENED.

When assigning an RS-422 port to the second I/O channel (i.e. 1) or higher, this port can only be used when activating SV_OPTION_MULTICHANNEL as well.

When using multiple calls of SV_OPTION_RS422<xxx>, only one master task per I/O channel is allowed. When trying to create a second master task, this option call will return SV_ERROR_PARAMETER. Nevertheless, it is possible to create multiple slave tasks.

See also:

The chapter API – Jack API and the define SV_OPTION_MULTICHANNEL.

#define SV_OPTION_RS422B

This option call independently configures the physical pin-out, the logical task and the assigned I/O channel (when in a multi-channel environment) of the RS-422 port B.

For a complete parameter list please refer to SV_OPTION_RS422A. Only the following two parameters differ from their meaning given under SV_OPTION_RS422A:

- SV_RS422_PINOUT_DEFAULT – Sets this port to its default pin-out setting which is SV_RS422_PINOUT_SWAPPED.
- SV_RS422_TASK_DEFAULT – Sets this port to its default task setting which is SV_RS422_TASK_SLAVE.

Note:

For further information please refer to SV_OPTION_RS422A.

#define SV_QUERY_PULLDOWN

This define returns the pulldown phase of the material currently recorded or played out.

#define SV_QUERY_XPANNING

This define returns the setting of the x-panning value. See the function *sv_zoom()*.

#define SV_QUERY_XZOOM

This define returns the setting of the x-zoom value. See the function *sv_zoom()*.

#define SV_QUERY_YPANNING

This define returns the setting of the y-panning value. See the function *sv_zoom()*.

#define SV_QUERY_YZOOM

This define returns the setting of the y-zoom value. See the function *sv_zoom()*.

#define SV_QUERY_ZOOMFLAGS

This define returns the setting of the zoom flags value. See the function *sv_zoom()*.

Function Documentation

int sv_lut (sv_handle * *sv*, int *command*, int * *ptable*, int *lutid*)

This function programs the look-up table for a color correction.

Parameters:

sv – Handle returned from the function *sv_open()*.

command – Look-up table to be set (`SV_LUT_<xxx>`, see the file *`dvs_clib.h`* for all possible defines).

ptable – Pointer to the integer table that contains the look-up table data. This is an integer array containing 1024 elements.

lutid – Sets the ID of the look-up table: zero (`0`) for an output LUT, one (`1`) for an input LUT.

Returns:

If the function succeeds, it returns `SV_OK`. Otherwise it will return the error code `SV_ERROR_<xxx>`.

int sv_matrix (sv_handle * *sv*, int *matrixmode*, sv_matrixinfo * *pmatrix*)

This function changes the default matrix used in the hardware for a color space conversion between YUV and RGB and vice versa. This function does not allow to set the matrix in- and out-offsets when specifying a custom matrix. To set the matrix in- and out-offsets you have to use the function *sv_matrixex()*.

The matrix coefficients have the following layout in the matrix array:

0:r2y 1:g2y 2:b2y

3:r2u 4:g2u 5:b2u

6:r2v 7:g2v 8:b2v

9:alpha

All values are fixed point float. A common divisor for all custom coefficients has to be specified in the structure *sv_matrixinfo*. The range for this divisor is limited to `0<divisor<0x100000`.

Parameters:

sv – Handle returned from the function *sv_open()*.

matrixmode – `SV_MATRIX_<xxx>` defines setting the matrix mode/type. See list below.

pmatrix – Pointer to the structure *sv_matrixinfo*.

Parameters for *matrixmode*:

- `SV_MATRIX_DEFAULT` – Uses the raster default matrix. The matrix will be selected according to the color space set for the storage mode and the I/O mode.
- `SV_MATRIX_CUSTOM` – Uses the custom matrix in *pmatrix*.
- `SV_MATRIX_QUERY` – Returns the currently used matrix.

- SV_MATRIX_IDENTITY – 1:1 conversion, i.e. no conversion takes place.
- SV_MATRIX_CCIR601 – RGB to SD YUV, or vice versa.
- SV_MATRIX_CCIR601CGR – RGB full to SD YUV head, or vice versa.
- SV_MATRIX_CCIR601INV – RGB head to SD YUV full, or vice versa.
- SV_MATRIX_SMPTE274 – RGB to HD YUV, or vice versa.
- SV_MATRIX_SMPTE274CGR – RGB full to HD YUV head, or vice versa.
- SV_MATRIX_SMPTE274INV – RGB head to HD YUV full, or vice versa.
- SV_MATRIX_CCIR709 – Same as SV_MATRIX_SMPTE274.
- SV_MATRIX_CCIR709CGR – Same as SV_MATRIX_SMPTE274CGR.
- SV_MATRIX_601TO274 – SD YUV to HD YUV.
- SV_MATRIX_601FTO274H – SD YUV full to HD YUV head.
- SV_MATRIX_601HTO274F – SD YUV head to HD YUV full.
- SV_MATRIX_274TO601 – HD YUV to SD YUV.
- SV_MATRIX_274FTO601H – HD YUV full to SD YUV head.
- SV_MATRIX_274HTO601F – HD YUV head to SD YUV full.
- SV_MATRIX_RGBHEAD2FULL – RGB head to RGB full.
- SV_MATRIX_RGBFULL2HEAD – RGB full to RGB head.
- SV_MATRIX_YUVHEAD2FULL – YUV head to YUV full.
- SV_MATRIX_YUVFULL2HEAD – YUV full to YUV head.
- SV_MATRIX_HEAD2FULL – Same as SV_MATRIX_RGBHEAD2FULL. For YUV data use SV_MATRIX_YUVHEAD2FULL.
- SV_MATRIX_FULL2HEAD – Same as SV_MATRIX_RGBFULL2HEAD. For YUV data use SV_MATRIX_YUVFULL2HEAD.

Parameters for *matrixmode* (Flags):

- SV_MATRIX_FLAG_CGRMATRIX – Only evaluated for the custom matrix. Sets the CGR range matrix, i.e. the default value range for RGB to full and for YUV to the restricted value range.
- SV_MATRIX_FLAG_SETINPUTFILTER – Programs the input filter next to a matrix as well. Needs *pmatrix->inputfilter*. See list below.
- SV_MATRIX_FLAG_SETOUTPUTFILTER – Programs the output filter next to a matrix as well. Needs *pmatrix->outputfilter*. See list below.
- SV_MATRIX_FLAG_FORCEMATRIX – Forces a programming of the matrix when the I/O mode and storage mode are in the same color space, e.g. when converting from YUV to YUV or RGB to RGB. Otherwise no color conversion will be performed (1:1 conversion).

Parameters for *pmatrix->inputfilter*:

The input filter defines the video 422-to-444 filter, i.e. how many taps the decimation and interpolation filter provides:

- SV_INPUTFILTER_DEFAULT – Selects the default filter (usually SV_INPUTFILTER_17TAPS).
- SV_INPUTFILTER_5TAPS – 5 taps interpolation or decimation filter width.
- SV_INPUTFILTER_9TAPS – 9 taps interpolation or decimation filter width.
- SV_INPUTFILTER_13TAPS – 13 taps interpolation or decimation filter width.
- SV_INPUTFILTER_17TAPS – 17 taps interpolation or decimation filter width.

Parameters for *pmatrix->outputfilter*:

The output filter defines the video 444-to-422 filter, i.e. how many taps the decimation and interpolation filter provides:

- SV_OUTPUTFILTER_DEFAULT – Selects the default filter (usually SV_OUTPUTFILTER_17TAPS).
- SV_OUTPUTFILTER_5TAPS – 5 taps interpolation or decimation filter width.
- SV_OUTPUTFILTER_9TAPS – 9 taps interpolation or decimation filter width.
- SV_OUTPUTFILTER_13TAPS – 13 taps interpolation or decimation filter width.
- SV_OUTPUTFILTER_17TAPS – 17 taps interpolation or decimation filter width.

Returns:

If the function succeeds, it returns SV_OK. Otherwise it will return the error code SV_ERROR_<xxx>.

See also:

The functions *sv_fifo_matrix()* and *sv_matrixex()*.

Example:

```
void example_printmatrix(sv_handle * sv);
{
  sv_matrix matrix;
  int res;
  int i,j;

  res = sv_matrix(sv, SV_MATRIX_QUERY, &matrix);
  if(res == SV_OK) {
    if(matrix.divisor != 0) {
      printf("matrix:\n");
        for(i = 0; i < 3; i++) {
          for(j = 0; j < 3; j++) {
            printf("%1.5f ", ((double)matrix.matrix[i*3+j]/matrix.divisor));
          }
          printf("\n");
        }
        printf("key %1.5f\n", ((double)matrix.matrix[9]/matrix.divisor));
        printf("dematrix:\n");
        for(i = 0; i < 3; i++) {
          for(j = 0; j < 3; j++) {
            printf("%1.5f ", ((double)matrix.dematrix[i*3+j]/matrix.divisor));
          }
          printf("\n");
        }
        printf("key %1.5f\n", ((double)matrix.dematrix[9]/matrix.divisor));
      } else {
        printf("\tdivisor == 0\n");
      }
    }
  }
}
```

int sv_matrixex (sv_handle * *sv*, int *matrixtype*, sv_matrixexinfo * *pmatrix*, sv_matrixexinfo * *pquery*)

This function changes the default matrix used in the hardware for a color space conversion between YUV and RGB and vice versa. Compared to the function *sv_matrix()*, this function also allows to set the matrix in- and out-offsets when specifying a custom matrix.

The matrix coefficients have the following layout in the matrix array (please note that the order of the coefficients differs from the ones specified in *sv_matrix()*):

0:g2y 1:b2y 2:r2y

3:g2u 4:b2u 5:r2u

6:g2v 7:b2v 8:r2v

9:alpha

10:inoffset_r 11:inoffset_g 12:inoffset_b 13:inoffset_alpha

14:outoffset_y 15:outoffset_u 16:outoffset_v 17:outoffset_alpha

All values are fixed point float. A common divisor for all custom coefficients has to be specified in the structure *sv_matrixexinfo*. The range for this divisor is limited to `0<divisor<0x100000`. An offset of one (`1`) (depending on the divisor) is interpreted as the full value range (e.g. 1024 in 10 bit video modes).

Parameters:

sv – Handle returned from the function *sv_open()*.

matrixtype – `SV_MATRIX_<xxx>` defines setting the matrix mode/type. For a list of these defines see the function *sv_matrix()*.

pmatrix – Pointer to the structure *sv_matrixexinfo*.

pquery – Returns the current matrix settings (can be `NULL`).

Parameters for *matrixtype* (Flags):

- `SV_MATRIX_FLAG_FORCEMATRIX` – Forces a programming of the matrix when the I/O mode and storage mode are in the same color space, e.g. when converting from YUV to YUV or RGB to RGB. Otherwise no color conversion will be performed (1:1 conversion).

Returns:

If the function succeeds, it returns `SV_OK`. Otherwise it will return the error code `SV_ERROR_<xxx>`.

Note:

When using this function, an in- and output filter can be set with the define `SV_OPTION_INPUTFILTER` or `SV_OPTION_OUTPUTFILTER` respectively.

See also:

The functions *sv_fifo_matrix()* and *sv_matrix()*.

int sv_zoom (sv_handle * *sv*, int *xzoom*, int *yzoom*, int *xpanning*, int *ypanning*, int *flags*)

This function performs a zooming and panning on the output images. There is no direct connection to the FIFO API. Only integer factor zooming (`1`, `2`, `4`, `8`, `16`, `32`) and integer pixel panning are supported. A set zooming or panning will be used with the next vertical sync and the zooming will be performed by simply doubling or cutting away pixels without any filtering.

Parameters:

sv – Handle returned from the function *sv_open()*.

xzoom – Zooming factor in horizontal direction. Can be set independently for each direction.

yzoom – Zooming factor in vertical direction. Can be set independently for each direction.

xpanning – Horizontal panning factor.

ypanning – Vertical panning factor.

flags – By default a zooming is done in the native mode of the video raster, i.e. in progressive and segmented frame rasters zooming is performed progressive, in interlaced rasters it is done interlaced. See list below.

Parameters for *flags*:

- `SV_ZOOMFLAGS_PROGRESSIVE` – Sets the device to a progressive zooming.
- `SV_ZOOMFLAGS_INTERLACED` – Sets the device to an interlaced zooming.
- `SV_ZOOMFLAGS_FIXEDFLOAT` – The values *xzoom*, *yzoom*, *xpanning*, and *ypanning* are defined as fixed float values, `floatvalue = ((double)x) / 0x10000`.

Returns:

If the function succeeds, it returns `SV_OK`. Otherwise it will return the error code `SV_ERROR_<xxx>`.

Note:

Define only one of the values `SV_ZOOMFLAGS_PROGRESSIVE` or `SV_ZOOMFLAGS_INTERLACED`. If both are set, a progressive zooming will be performed.

This function is available on Centaurus II only.

API – Status Functions

Detailed Description

This chapter details basic functions to set up or query the status of a DVS video device.

Defines

- #define SV_QUERY_DEVTYPE
- #define SV_QUERY_FEATURE
- #define SV_QUERY_FEATURE_AUDIOCHANNELS
- #define SV_QUERY_VALUE_AVAILABLE
- #define SV_QUERY_VERSION_DRIVER
- #define SV_QUERY_VERSION_DVSOEM

Functions

- int sv_raster_status (sv_handle *sv, int rasterid, sv_rasterheader *raster, int rastersize, int *nrasters, int spare)
- int sv_status (sv_handle *sv, sv_info *info)
- int sv_storage_status (sv_handle *sv, int cookie, sv_storageinfo *psiin, sv_storageinfo *psiout, int psioutsize, int flags)
- int sv_version_status (sv_handle *sv, sv_version *pversion, int versionsize, int deviceid, int moduleid, int spare)

Define Documentation

#define SV_QUERY_DEVTYPE

This define returns the device type (`SV_DEVTYPE_<xxx>`). Possible returns are:

- `SV_DEVTYPE_ATOMIX` – Atomix.
- `SV_DEVTYPE_ATOMIXLT` – Atomix LT.
- `SV_DEVTYPE_CENTAURUS` – Centaurus II.

Note:

In case you want to query the serial number of a DVS video board, you can use the function *sv_getlicence()* or the define `SV_QUERY_SERIALNUMBER`. The serial numbers of the supported DVS video board product (first digits) are listed in section Supported DVS Video Board Products.

#define SV_QUERY_FEATURE

This define returns a bit mask that describes the features available for the DVS video device. Possible returns are:

- `SV_FEATURE_CAPTURE` – The function *sv_capture()* is supported.
- `SV_FEATURE_DUALLINK` – Dual-link operation is supported.

- SV_FEATURE_INDEPENDENT_IO – Input and output support different rasters and storage formats (independent I/O).
- SV_FEATURE_KEYCHANNEL – Key channel operation is supported.
- SV_FEATURE_LUTSUPPORT – Loadable LUTs are supported.
- SV_FEATURE_MIXERPROCESSING – The mixing of channels coming from memory is supported (see e.g. the define SV_FIFO_FLAG_VIDEO_B).
- SV_FEATURE_MULTIJACK – Support of more than two default jacks (see chapter API – Jack API).
- SV_FEATURE_RASTERLIST – The generation of a raster list is supported.
- SV_FEATURE_ZOOMSUPPORT – Zooming and panning is supported (see e.g. the function *sv_zoom()*).

#define SV_QUERY_FEATURE_AUDIOCHANNELS

This define returns the number of stereo audio channel pairs licensed on your DVS video board product.

#define SV_QUERY_VALUE_AVAILABLE

This define returns TRUE if the specified SV_OPTION_<xxx> value is available.

#define SV_QUERY_VERSION_DRIVER

This query returns the driver version.

#define SV_QUERY_VERSION_DVSOEM

This query returns the DVSOEM library version.

Function Documentation

int sv_raster_status (sv_handle * *sv*, int *rasterid*, sv_rasterheader * *raster*, int *rastersize*, int * *nrasters*, int *spare*)

This function returns information either about a specific video raster and fills them into the structure *sv_rasterheader*, or about all available rasters and their internal index numbers in the raster table.

Parameters:

sv – Handle returned from the function *sv_open()*.

rasterid – If this parameter is set to a value of -1, the function returns the number of all available rasters. In case this parameter contains a specific internal raster identifier (index), the function fills information about this raster into the structure *sv_rasterheader*. Please note that you cannot pass a raster define (SV_MODE_<xxx>) in this parameter directly, because the function works with the internal raster indexes only (see example below).

raster – Structure that will be filled with information about the raster. See the structure *sv_rasterheader* in the file *dvs_clib.h*.

rastersize – Size of the structure *sv_rasterheader*.

nrasters – Number of available rasters.

spare – Currently not used. It has to be set to zero (0).

Returns:

If the function succeeds, it returns `SV_OK`. Otherwise it will return the error code `SV_ERROR_<xxx>`.

Note:

To get the raster index of the currently set raster of the DVS video device you can use the query `SV_QUERY_RASTERID`.

Example:

```
  int example_dumprasters(sv_handle * sv)
  {
    int res;

    // Query the maximum raster count
    res = sv_raster_status(sv, -1, &current, sizeof(current), &nrasters, 0);

    // Iterate over the complete raster list and print the available information
    for(int raster_index = 0; (res == SV_OK) && (raster_index < nrasters);
raster_index++) {
      res = sv_raster_status(sv, raster_index, &raster, sizeof(raster), NULL, 0);
      if(res == SV_OK) {
        printf("RASTERLIST %2d \"%-32s\"\n", raster.index, raster.name);
      }
    }

    return res;
  }
```

int sv_status (sv_handle * *sv*, sv_info * *info*)

This function retrieves various information from the DVS video device, for example, about its currently set video mode parameters, its settings as a VTR master as well as various other configuration settings. Some parts of the dynamic information (such as inpoint and outpoint) are updated on a frame basis on the video device only. Reading these parameters is possible only on a frame basis as well and will be retrieved from the last frame that was processed completely.

Parameters:

sv – Handle returned from the function *sv_open()*.

info – Pointer to the *sv_info* structure.

Returns:

If the function succeeds, it returns `SV_OK`. Otherwise it will return the error code `SV_ERROR_<xxx>`.

Example:

```
  int example_displayxysize(sv_handle * sv)
  {
    sv_info info;
    int     res;

    res = sv_status(sv, &info);
    if(res == SV_OK) {
      printf("Current video size %dx%d\n", info.xsize, info.ysize);
    } else {
      printf("Error: sv_status() failed %d '%s'\n", res, sv_geterrortext(res));
    }

    return res;
  }
```

int sv_storage_status (sv_handle * *sv*, int *cookie*, sv_storageinfo * *psiin*, sv_storageinfo * *psiout*, int *psioutsize*, int *flags*)

This function returns memory layout information, i.e. gets information about the current storage settings of the DVS video device and fills these into the structure *sv_storageinfo*.

Parameters:

sv – Handle returned from the function *sv_open()*.

cookie – Default is zero (0). Other values only in combination with *flags* (see *flags* below).

psiin – Input of the structure *sv_storageinfo*. Has to be used in combination with *flags* only (see *flags* below). Default is NULL.

psiout – Output of the structure *sv_storageinfo*.

psioutsize – Size of storage pointed to by *psiout*.

flags – Optional flags. See list below.

Parameters for *flags*:

- SV_STORAGEINFO_COOKIEISMODE – The cookie variable is interpreted as a video mode. Without this flag the current video mode of the device will be used.
- SV_STORAGEINFO_COOKIEISJACK – The cookie variable is interpreted as a jack. Without this flag the last set video mode (e.g. via the function *sv_videomode()*) for the DVS video device will be used.
- SV_STORAGEINFO_INPUT_XSIZE – *psiin->xsize* is used for *psiout*.
- SV_STORAGEINFO_INPUT_YSIZE – *psiin->ysize* is used for *psiout*.
- SV_STORAGEINFO_INPUT_NBITS – *psiin->nbits* is used for *psiout*.

Returns:

If the function succeeds, it returns SV_OK. Otherwise it will return the error code SV_ERROR_<xxx>.

int sv_version_status (sv_handle * *sv*, sv_version * *pversion*, int *versionsize*, int *deviceid*, int *moduleid*, int *spare*)

This function returns version information about different parts of the DVS video device, i.e. gets information about the device's versions and fills these into the structure *sv_version*. This function is mainly used for diagnostic purposes.

Parameters:

sv – Handle returned from the function *sv_open()*.

pversion – Pointer to the structure *sv_version*.

versionsize – Size of the structure *sv_version*.

deviceid – Device number where the information should be retrieved from. Has to be set to zero (0).

moduleid – Module number where the information should be retrieved from. All modules together describe the setup of your system and detail information such as driver version, board version, flash version, etc.

spare – Currently not used. It has to be set to zero (0).

Returns:

If the function succeeds, it returns SV_OK. Otherwise it will return the error code SV_ERROR_<xxx>.

Example:

```
void example_showversion(sv_handle * sv)
{
  sv_version version;
  int res = SV_OK;
  int device;
  int module, modulecount;
  char tmp[64];

  device = 0;
  modulecount = 1;
```

```
    for(module = 0; (res == SV_OK) && (module < modulecount); module++) {
      res = sv_version_status(sv, &version, sizeof(version), device, module, 0);
      if(res == SV_OK) {
        if(module == 0) {
          modulecount = version.modulecount;
        }
        sprintf(tmp, "%d.%d.%d.%d", version.release.v.major,
version.release.v.minor, version.release.v.patch, version.release.v.fix);
        strcat(version.module, ":");
        printf ("%-15s %10s ", version.module, tmp);

        if(version.date.date || version.time.time) {
          printf("%04x/%02x/%02x %02x:%02x ",
            version.date.d.yyyy, version.date.d.mm, version.date.d.dd,
            version.time.t.hh, version.time.t.mm);
        }
        printf("%s\n", version.comment);
      }
    }
  }
```

API – RS-422 High-level API (Master)

Detailed Description

This chapter describes various defines and functions for the VTR master control functionality of the DVS SDK, i.e. when your application controls an external VTR (remote control via RS-422).

To configure the RS-422 ports use the option calls `SV_OPTION_RS422A` and `SV_OPTION_RS422B`.

Defines

- #define SV_OPTION_VTRMASTER_EDITLAG
- #define SV_OPTION_VTRMASTER_FLAGS
- #define SV_OPTION_VTRMASTER_POSTROLL
- #define SV_OPTION_VTRMASTER_PREROLL
- #define SV_OPTION_VTRMASTER_TCTYPE
- #define SV_OPTION_VTRMASTER_TOLERANCE

Functions

- int sv_asc2tc (sv_handle *sv, char *str, int *ptc)
- int sv_tc2asc (sv_handle *sv, int timecode, char *str, int len)
- int sv_vtrcontrol (sv_handle *sv, int binput, int init, int tc, int nframes, int *pwhen, int *pvtrtc, int flags)
- int sv_vtrmaster (sv_handle *sv, int opcode, int value)
- int sv_vtrmaster_raw (sv_handle *sv, char *cmdstr, char *replystr)

Define Documentation

#define SV_OPTION_VTRMASTER_EDITLAG

This define sets the edit lag of the VTR master. The unit is in frames.

#define SV_OPTION_VTRMASTER_FLAGS

This define sets the VTR master flags bit mask:

- `SV_MASTER_FLAG_AUTOEDIT` – Performs autoedit operations instead of the default edit-on/edit-off commands. When using *sv_vtrmaster()*, it does not affect `SV_MASTER_AUTOEDIT` and `SV_MASTER_AUTOEDITONOFF`.
- `SV_MASTER_FLAG_FORCEDROPFRAME` – Forces the interpretation of a VTR timecode as a drop-frame timecode, i.e. uses the drop-frame timecode calculation at 30 Hz even if the VTR does not say so.
- `SV_MASTER_FLAG_EMULATESTEPCMD` – Emulates RS-422 step commands (214/224) with 'move to' commands.

#define SV_OPTION_VTRMASTER_POSTROLL

This define sets the postroll time after an edit operation on the VTR master. The unit is in seconds.

#define SV_OPTION_VTRMASTER_PREROLL

This define sets the preroll time prior to an edit operation on the VTR master. The unit is in seconds.

#define SV_OPTION_VTRMASTER_TCTYPE

This define allows you to set the timecode type that will be returned by the VTR in timecode replies. It performs the same operation as the option code `SV_MASTER_TIMECODE_<xxx>` of the function *sv_vtrmaster()*:

- `SV_MASTER_TIMECODE_VITC` – Asks for the VITC timecode.
- `SV_MASTER_TIMECODE_LTC` – Asks for the LTC timecode.
- `SV_MASTER_TIMECODE_AUTO` – The VTR chooses if it should return VITC or LTC timecode.
- `SV_MASTER_TIMECODE_TIMER1` – Asks for timer1 timecode.
- `SV_MASTER_TIMECODE_TIMER2` – Asks for timer2 timecode.

#define SV_OPTION_VTRMASTER_TOLERANCE

This define sets the edit tolerance of the VTR master (e.g. how to react to delayed response times or identical timecodes). It affects eventual timeouts: For example, if the tolerance is exceeded, the VTR master may abort the operation. It performs the same operation as the option code `SV_MASTER_TOLERANCE_<xxx>` of the function *sv_vtrmaster()*:

- `SV_MASTER_TOLERANCE_NONE` – Almost no tolerance.
- `SV_MASTER_TOLERANCE_NORMAL` – Normal tolerance.
- `SV_MASTER_TOLERANCE_LARGE` – The tolerance is high.
- `SV_MASTER_TOLERANCE_ROUGH` – The tolerance is very high.

Function Documentation

int sv_asc2tc (sv_handle * *sv*, char * *str*, int * *ptc*)

This function converts an ASCII representation into the timecode format.

Parameters:

sv – Handle returned from the function *sv_open()*.

str – ASCII timecode to be converted.

ptc – Pointer to the returned timecode.

Returns:

If the function succeeds, it returns `SV_OK`. Otherwise it will return the error code `SV_ERROR_<xxx>`.

See also:

The function *sv_tc2asc()*.

Examples:

```
0:0:0:0   -> 00:00:00:00
0:0:0.0   -> 00:00:00.00
1:23:4    -> 00:01:23:04
-22       -> 00:00:00:22
-1        -> 00:00:00:01
0         -> 00:00:00:00
9         -> 00:00:00:09
99        -> 00:00:00:99
101       -> 00:00:01:01
12345678 -> 12:34:56:78
```

int sv_tc2asc (sv_handle * *sv*, int *timecode*, char * *str*, int *len*)

This function converts a timecode into an ASCII representation. It decodes drop-frame timecode and field bits as well.

Parameters:

sv – Handle returned from the function *sv_open()*.

timecode – Timecode to be converted.

str – Pointer to the string that receives the ASCII timecode.

len – Size of the buffer pointed to by *str*.

Returns:

If the function succeeds, it returns `SV_OK`. Otherwise it will return the error code `SV_ERROR_<xxx>`.

See also:

The function *sv_asc2tc()*.

Example:

```
int example_displaytimecode(sv_handle * sv, int tc)
{
  char tmpbuffer[40];
  int res;

  res = sv_tc2asc(sv, tc, tmpbuffer, sizeof(tmpbuffer));
  if(res != SV_OK) {
    printf("Error: sv_tc2asc() failed = %d '%s'\n", res, sv_geterrortext(res));
    return FALSE;
  } else {
    printf("The current timecode is %s", tmpbuffer);
  }

  return TRUE;
}
```

int sv_vtrcontrol (sv_handle * *sv*, int *binput*, int *init*, int *tc*, int *nframes*, int * *pwhen*, int * *pvtrtc*, int *flags*)

This function is used for the two purposes as indicated in the following. However, the executions of both will work in the given order only:

1. Initializing a VTR operation with the first function call.
2. Querying the VTR status with subsequent function calls. The main purpose of subsequent calls is to receive the driver tick (timestamp) for a timed FIFO start (see description of the parameter *pwhen* below).

Parameters:

sv – Handle returned from the function *sv_open()*.

binput – This parameter is part of the control command sent to the VTR and defines the type of the VTR operation. If *binput* is `TRUE`, the VTR goes into play mode; if it is `FALSE`, the VTR goes into record mode. This parameter is only evaluated if the *init* parameter is `TRUE`.

However, you should keep the initial value of *binput* in all subsequent *sv_vtrcontrol()* calls. This will help to avoid possible incompatibilities of your programs with later SDK versions.

init – This parameter is used to indicate whether the current function call is the first or a subsequent call. If it is `TRUE`, the driver sends the VTR control command which is based on the values of *binput*, *tc* and *nframes*. If it is `FALSE`, nothing is sent to the VTR, thus leaving it in the current operation mode. So, only when calling the function for the first time, set *init* to `TRUE`. In all subsequent calls, *init* must be `FALSE`, otherwise the VTR will get a new control command each time. Whether this leads to erroneous behavior or not depends on the VTR.

tc – This parameter is part of the control command sent to the VTR and defines the inpoint of the VTR operation, i.e. the timecode where the VTR operation should start.

nframes – This parameter is part of the control command sent to the VTR and defines the number of frames that the VTR operation should last. After reaching *nframes* the VTR ends the operation, performs the postroll and stops.

pwhen – After initializing the VTR operation with the first *sv_vtrcontrol()* call, the video board driver uses the VTR's preroll time to negotiate the edit tick where the VTR operation will actually start. *pwhen* returns this edit tick as soon as it is available. It can then be used to start a timed FIFO (function *sv_fifo_getbuffer()* in conjunction with the flag `SV_FIFO_FLAG_TIMEDOPERATION`; see also chapter API – FIFO API). Before this edit tick is available *pwhen* will return zero (`0`). The caller should be polling the function *sv_vtrcontrol()* until receiving an edit tick which is not zero (`0`).

pvtrtc – Returns the current VTR timecode regardless of the mode that the VTR is in.

flags – Additional flags to control the behavior of this function. See list below. The `SV_VTRCONTROL_MODE_<xxx>` flags only have an effect when *binput* is `FALSE`.

Parameters for *flags*:

- `SV_VTRCONTROL_MODE_DEFAULT` – Default edit or play-out behavior (value zero (`0`)).
- `SV_VTRCONTROL_MODE_AUTOEDIT` – This flag performs an autoedit explicitly.
- `SV_VTRCONTROL_MODE_REVIEW` – This flag performs a review explicitly.
- `SV_VTRCONTROL_MODE_PREVIEW` – This flag performs a preview explicitly.
- `SV_VTRCONTROL_MODE_MASK` – Mask for possible operation mode flags.

Returns:

If the function succeeds, it returns `SV_OK`. Otherwise it will return the error code `SV_ERROR_<xxx>`.

Note:

This function is useful in conjunction with the FIFO API only.

Additionally, it overrides any `SV_MASTER_INPOINT`, `SV_MASTER_OUTPOINT` or `SV_MASTER_NFRAMES` values previously set via the function *sv_vtrmaster()*.

The example program *dpxio* shows how to use this function (see chapter Example Projects Overview).

int sv_vtrmaster (sv_handle * *sv*, int *opcode*, int *value*)

This function issues a master command to control a VTR via an RS-422 interface. It takes command defines for a more intuitive use instead of raw RS-422 commands. For a control with raw commands use the function *sv_vtrmaster_raw()*.

Parameters:

sv – Handle returned from the function *sv_open()*.

opcode – `SV_MASTER_<xxx>` define. See lists below.

value – Parameter depending on *opcode*.

Parameters for *opcode* (Setup Commands):

- SV_MASTER_DISOFFSET – Sets the display offset in frames.
- SV_MASTER_EDITLAG – Sets the VTR edit lag. This value depends on the capabilities of the controlled VTR. Refer to the VTR manual or its configurations to get this value.
- SV_MASTER_FORCEDROPFRAME – Forces a drop-frame timecode in 30 Hz, even if the drop-frame bit is not set.
- SV_MASTER_INPOINT – Sets the inpoint for the edit operation.
- SV_MASTER_OUTPOINT – Sets the outpoint for the edit operation. Use either this or SV_MASTER_NFRAMES.
- SV_MASTER_NFRAMES – Sets the number of frames that the edit operation should last. Use either this or SV_MASTER_OUTPOINT.
- SV_MASTER_POSTROLL – Sets the postroll time (in timecode) when to stop the VTR after an edit operation.
- SV_MASTER_PREROLL – Sets the preroll time (in timecode) when to start the VTR before an edit operation takes place.
- SV_MASTER_PRESET – Sets the channels for a record on the VTR during subsequent edit operations. The following settings are possible:
  - SV_MASTER_PRESET_VIDEO – Video.
  - SV_MASTER_PRESET_AUDIO1 – Analog audio mono channel 1.
  - SV_MASTER_PRESET_AUDIO2 – Analog audio mono channel 2.
  - SV_MASTER_PRESET_AUDIO3 – Analog audio mono channel 3, also used for LTC timecode control.
  - SV_MASTER_PRESET_AUDIO4 – Analog audio mono channel 4.
  - SV_MASTER_PRESET_DIGAUDIO1 – Digital audio mono channel 1.
  - SV_MASTER_PRESET_DIGAUDIO2 – Digital audio mono channel 2.
  - SV_MASTER_PRESET_DIGAUDIO3 – Digital audio mono channel 3.
  - SV_MASTER_PRESET_DIGAUDIO4 – Digital audio mono channel 4.
  - SV_MASTER_PRESET_DIGAUDIO5 – Digital audio mono channel 5.
  - SV_MASTER_PRESET_DIGAUDIO6 – Digital audio mono channel 6.
  - SV_MASTER_PRESET_DIGAUDIO7 – Digital audio mono channel 7.
  - SV_MASTER_PRESET_DIGAUDIO8 – Digital audio mono channel 8.
  - SV_MASTER_PRESET_AUDIOMASK – Mask for all analog audio channels.
  - SV_MASTER_PRESET_DIGAUDIOMASK – Mask for all digital audio channels.
  - SV_MASTER_PRESET_ASSEMBLE – Enables the assemble mode. The assemble mode is used to overwrite data such as timecode on the VTR.
- SV_MASTER_RECOFFSET – Sets a record offset in frames.
- SV_MASTER_TIMECODE – Sets the timecode type that will be returned by the VTR in timecode replies. For possible types see the define SV_OPTION_VTRMASTER_TCTYPE.
- SV_MASTER_TOLERANCE – Sets the edit tolerance of the VTR master. It affects eventual timeouts. For possible types see the define SV_OPTION_VTRMASTER_TOLERANCE.
- SV_MASTER_EDITFIELD_START – Sets the start field for the editing. The start field is the first field of the editing, i.e. it is included in the editing. Possible values are one (1) for field 1 or two (2) for field 2. The driver's default is the first field (1).
- SV_MASTER_EDITFIELD_END – Sets the end field for the editing. The end field is the first field that will not be edited anymore. Possible values are one (1) for field 1 or two (2) for field 2. The driver's default is the first field (1).

Parameters for *opcode* (Control Commands):

In brackets you can find the corresponding raw Sony 9-pin remote protocol commands.

- SV_MASTER_EJECT – Issues an eject command (20f).
- SV_MASTER_FORWARD – Issues a forward command (210).
- SV_MASTER_GOTO – Issues a cue up with data command (231).
- SV_MASTER_JOG – Issues a jog command (211/221). The value is a fixed point speed, speed * 0x10000.
- SV_MASTER_LIVE – Switches on the live mode (EE) (261).
- SV_MASTER_MOVETO – Moves to the specified position on the tape (using shuttle/jog).
- SV_MASTER_PAUSE – Sets the play-out speed to zero (0), i.e. SV_MASTER_JOG (211) with speed zero (0).
- SV_MASTER_PLAY – Issues a play command (201).
- SV_MASTER_RECORD – Issues a record command (202).
- SV_MASTER_REWIND – Issues a rewind command (220).
- SV_MASTER_SHUTTLE – Issues a shuttle command (211/221). The value is a fixed point speed, speed * 0x10000.
- SV_MASTER_STANDBY – Switches on the standby mode (205).
- SV_MASTER_STBOFF – Disables the standby mode.
- SV_MASTER_STEP – Issues a step command (214/224). The value is the number of frames to step (positive as well as negative). This command is not supported by all VTRs. If it is not, use the flag SV_MASTER_FLAG_EMULATESTEPCMD of the define SV_OPTION_VTRMASTER_FLAGS to emulate step commands.
- SV_MASTER_STOP – Issues a stop command (200).
- SV_MASTER_VAR – Issues a var command (211/221). The value is a fixed point speed, speed * 0x10000.
- SV_MASTER_AUTOEDITONOFF – Starts an autoedit operation using edit-on/-off commands (265/264).
- SV_MASTER_AUTOEDIT – Starts an autoedit operation using the autoedit function of the VTR (242).
- SV_MASTER_PREVIEW – Starts an autoedit operation using the preview function of the VTR (240).
- SV_MASTER_REVIEW – Starts an autoedit operation using the review function of the VTR (241).

Parameters for *opcode* (Flags):

- SV_MASTER_FLAG – For possible settings see the define SV_OPTION_VTRMASTER_FLAGS.

Returns:

If the function succeeds, it returns SV_OK. Otherwise it will return the error code SV_ERROR_<xxx>.

See also:

The function *sv_vtrmaster_raw()*.

int sv_vtrmaster_raw (sv_handle * *sv*, char * *cmdstr*, char * *replystr*)

This function issues a master command to control a VTR via the RS-422 interface. It takes the raw Sony 9-pin remote protocol commands in an abbreviated form. Some of them can be found in the description of the function *sv_vtrmaster()* which can be used to work with symbolic commands instead of raw RS-422 commands.

Parameters:

sv – Handle returned from the function *sv_open()*.

cmdstr – Defines the kind of operation. This parameter takes the raw RS-422 9-pin protocol commands in an abbreviated ASCII form, i.e. the check sum and byte size are calculated automatically.

replystr – Reply for command, same format as *cmdstr*. In case you do not need this, set it to `NULL`.

Returns:

If the function succeeds, it returns `SV_OK`. Otherwise it will return the error code `SV_ERROR_<xxx>`.

See also:

The function *sv_vtrmaster()*.

Example:

```
  int example_cueup(sv_handle * sv)
  {
    int res;

    // Stop: Family command 2 + Command 00 (no additional data)
    res = sv_vtrmaster_raw(sv, "200", NULL);
    if(res != SV_OK) {
      printf("Error: sv_vtrmaster_raw() failed = %d '%s'", res,
sv_geterrortext(res));
    }

    if(res == SV_OK) {

      // Wait 1 second
      sv_usleep(1000000);

      // Cue up to 00:10:20:05: Family command 2 + Command 31 + Data
      res = sv_vtrmaster_raw(sv, "231 05 20 10 00", NULL);
      if(res != SV_OK) {
        printf("Error: sv_vtrmaster_raw() failed = %d '%s'", res,
sv_geterrortext(res));
      }
    }

    return res;
  }
```

API – RS-422 High-level API (Slave)

Detailed Description

The functions described in this chapter allow you to implement a VTR emulation with the handling of the low-level CRC and communications performed in the driver.

To configure the RS-422 ports use the option calls SV_OPTION_RS422A and SV_OPTION_RS422B.

Functions

- int sv_slaveinfo_get (sv_handle *sv, sv_slaveinfo *slaveinfo, sv_overlapped *poverlapped)
- int sv_slaveinfo_set (sv_handle *sv, sv_slaveinfo *slaveinfo)

Function Documentation

int sv_slaveinfo_get (sv_handle * *sv*, sv_slaveinfo * *slaveinfo*, sv_overlapped * *poverlapped*)

This function returns information about the connected master from the driver. It is mainly used to get the received RS-422 commands.

Parameters:

sv – Handle returned from the function *sv_open()*.

slaveinfo – Pointer to the structure *sv_slaveinfo*.

poverlapped – Currently not used. It has to be set to NULL.

Returns:

If the function succeeds, it returns SV_OK. Otherwise it will return the error code SV_ERROR_<xxx>.

int sv_slaveinfo_set (sv_handle * *sv*, sv_slaveinfo * *slaveinfo*)

This function specifies the information to be returned by the driver and sent to the connected master.

Parameters:

sv – Handle returned from the function *sv_open()*.

slaveinfo – Pointer to the structure *sv_slaveinfo*. For possible values for the structure element *flags* see list below.

Parameters for *sv_slaveinfo.flags*:

- SV_SLAVEINFO_USESTATUSANDTC – This flag forces the usage of the status and timecode detailed in the structure *sv_slaveinfo*. If it is not set, the status and timecode provided by the FIFO API will be used.

Returns:

If the function succeeds, it returns SV_OK. Otherwise it will return the error code SV_ERROR_<xxx>.

API – RS-422 Low-level API

Detailed Description

The functions described in this chapter allow you to access the serial remote ports of a DVS video device on a lower level where you can configure them to your personal needs.

You can use these functions to access the main ports A and B, for example, in case they should provide a different configuration than the one automatically set with the default RS-422 functions.

Note:

Pin-outs of the RS-422 ports can be found in the installation guide of your DVS video board product.

Supported Baud Rates

- 9600
- 19200
- 38400 (default)
- 57600

It is recommended to use only a baud rate of 38400 with other standard RS-422 devices (e.g. VTRs) because normally they do not support a higher speed.

Functions

- int sv_rs422_close (sv_handle *sv, int device)
- int sv_rs422_open (sv_handle *sv, int device, int baudrate, int flags)
- int sv_rs422_rw (sv_handle *sv, int device, int bwrite, char *buffer, int buffersize, int *pbytecount, int flags)

Function Documentation

int sv_rs422_close (sv_handle * *sv*, int *device*)

This function closes a serial RS-422 port.

Parameters:

sv – Handle returned from the function *sv_open()*.

device – Selects the serial port. See the function *sv_rs422_open()* for valid values.

Returns:

If the function succeeds, it returns `SV_OK`. Otherwise it will return the error code `SV_ERROR_<xxx>`.

int sv_rs422_open (sv_handle * *sv*, int *device*, int *baudrate*, int *flags*)

This function opens a serial RS-422 port.

Parameters:

sv – Handle returned from the function *sv_open()*.

device – Selects the serial port. See list below.

baudrate – Baud rate of the RS-422 connection.

flags – Additional flags to control the behavior of this function. See list below.

Values for *device*:

- `0` – Main RS-422 port A.
- `1` – Main RS-422 port B.

Parameters for *flags*:

- `SV_RS422_OPENFLAG_SWAPPINOUT` – Uses a reverse pin-out (obsolete).
- `SV_RS422_OPENFLAG_MASTERPINOUT` – Uses the master pin-out (remote out).
- `SV_RS422_OPENFLAG_SLAVEPINOUT` – Uses the slave pin-out (remote in).

Returns:

If the function succeeds, it returns `SV_OK`. Otherwise it will return the error code `SV_ERROR_<xxx>`.

Note:

Only one *flags* parameter should be used in each function call.

On Centaurus II the initial pin-out of port A after driver loading is master and of port B is slave.

On Atomix the initial pin-out for all ports after driver loading is master.

For an example see the *`rs422test`* example program (for further information about this program see chapter Example Projects Overview).

int sv_rs422_rw (sv_handle * *sv*, int *device*, int *bwrite*, char * *buffer*, int *buffersize*, int * *pbytecount*, int *flags*)

This function reads data from or writes data to a serial RS-422 port.

This function reads and writes RS-422 data on a per-byte basis, meaning that it is not guaranteed that you get a complete RS-422 command in one read-call. A read-call may return less or even more than one RS-422 command. This function does not interpret the RS-422 commands, it simply reads from or writes to the RS-422 ports.

Parameters:

sv – Handle returned from the function *sv_open()*.

device – Selects the serial port. See the function *sv_rs422_open()* for valid values.

bwrite – For a write-call it has to be set to `TRUE`, for a read-call it must be `FALSE`.

buffer – Data buffer.

buffersize – Either the size of the buffer for a read-call or the number of bytes to be written with a write-call.

pbytecount – Number of bytes read or written.

flags – Currently not used. It has to be set to zero (`0`).

Returns:

If the function succeeds, it returns `SV_OK`. Otherwise it will return the error code `SV_ERROR_<xxx>`.

API – Timecode

Detailed Description

This chapter details defines and functions to control timecode related features of the DVS device.

Defines

- #define SV_OPTION_AFILM_TC
- #define SV_OPTION_AFILM_UB
- #define SV_OPTION_ANCCOMPLETE
- #define SV_OPTION_ANCGENERATOR
- #define SV_OPTION_ANCGENERATOR_RP165
- #define SV_OPTION_ANCREADER
- #define SV_OPTION_ANCUSER_DID
- #define SV_OPTION_ANCUSER_FLAGS
- #define SV_OPTION_ANCUSER_LINENR
- #define SV_OPTION_ANCUSER_SDID
- #define SV_OPTION_APROD_TC
- #define SV_OPTION_APROD_UB
- #define SV_OPTION_ASSIGN_LTCA
- #define SV_OPTION_DLTC_TC
- #define SV_OPTION_DLTC_UB
- #define SV_OPTION_DVITC_TC
- #define SV_OPTION_DVITC_UB
- #define SV_OPTION_FILM_TC
- #define SV_OPTION_FILM_UB
- #define SV_OPTION_FLUSH_TIMECODE
- #define SV_OPTION_LTC_TC
- #define SV_OPTION_LTC_UB
- #define SV_OPTION_LTCDELAY
- #define SV_OPTION_LTCDROPFRAME
- #define SV_OPTION_LTCFILTER
- #define SV_OPTION_LTCOFFSET
- #define SV_OPTION_LTCSOURCE
- #define SV_OPTION_PROD_TC
- #define SV_OPTION_PROD_UB
- #define SV_OPTION_VITC_TC
- #define SV_OPTION_VITC_UB
- #define SV_OPTION_VITCLINE
- #define SV_OPTION_VITCREADERLINE
- #define SV_OPTION_VTR_INFO
- #define SV_OPTION_VTR_INFO2
- #define SV_OPTION_VTR_INFO3
- #define SV_OPTION_VTR_TC

- #define SV_OPTION_VTR_UB
- #define SV_QUERY_AFILM_TC
- #define SV_QUERY_AFILM_UB
- #define SV_QUERY_ANC_MAXHANCLINENR
- #define SV_QUERY_ANC_MAXVANCLINENR
- #define SV_QUERY_ANC_MINLINENR
- #define SV_QUERY_APROD_TC
- #define SV_QUERY_APROD_UB
- #define SV_QUERY_DLTC_TC
- #define SV_QUERY_DLTC_UB
- #define SV_QUERY_DVITC_TC
- #define SV_QUERY_DVITC_UB
- #define SV_QUERY_FILM_TC
- #define SV_QUERY_FILM_UB
- #define SV_QUERY_LTCAVAILABLE
- #define SV_QUERY_LTCDROPFRAME
- #define SV_QUERY_LTCFILTER
- #define SV_QUERY_LTCOFFSET
- #define SV_QUERY_LTCSOURCE
- #define SV_QUERY_LTCTIMECODE
- #define SV_QUERY_LTCUSERBYTES
- #define SV_QUERY_PROD_TC
- #define SV_QUERY_PROD_UB
- #define SV_QUERY_VALIDTIMECODE
- #define SV_QUERY_VITCLINE
- #define SV_QUERY_VITCREADERLINE
- #define SV_QUERY_VITCTIMECODE
- #define SV_QUERY_VITCUSERBYTES

Functions

- int sv_timecode_feedback (sv_handle *sv, sv_timecode_info *input, sv_timecode_info *output)

Define Documentation

#define SV_OPTION_AFILM_TC

This define sets the analog RP201 film timecode. It can only be used together with the function *sv_option_setat()*.

See also:

The define SV_QUERY_AFILM_TC.

#define SV_OPTION_AFILM_UB

This define sets the analog RP201 film user bytes. It can only be used together with the function *sv_option_setat()*.

See also:

The define SV_QUERY_AFILM_UB.

#define SV_OPTION_ANCCOMPLETE

This option call activates or deactivates an extended ANC data handling. There are several different settings available for this call:

- SV_ANCCOMPLETE_OFF – Deactivates the extended ANC data handling.
- SV_ANCCOMPLETE_ON – Activates the extended ANC data handling to be used by the function *sv_fifo_anc()*.
- SV_ANCCOMPLETE_STREAMER – Activates the ANC streamer mode, i.e. the complete VANC and HANC are passed on unmodified. See also the flags below.

When the ANC data handling is set to SV_ANCCOMPLETE_ON, you can read or write ANC data from or to most parts of the video signal, i.e. most parts in the blanking interval of the active image will be used to include data. To get the valid minimum and maximum lines please call the defines SV_QUERY_ANC_MINLINENR, SV_QUERY_ANC_MAXVANCLINENR and SV_QUERY_ANC_MAXHANCLINENR. This setting has to be used together with the function *sv_fifo_anc()*.

When the ANC data handling is set to SV_ANCCOMPLETE_STREAMER, the ANC data are not handled as separate packets, but as one complete data buffer. It will pass the ANC data without any modifications. To work on the ANC data the ANC data buffer can be read or written via a regular FIFO buffer that is handled by the functions *sv_fifo_getbuffer()* and *sv_fifo_putbuffer()* (see the define SV_FIFO_FLAG_ANC).

When using this option call, you have to reinitialize the video mode using the function *sv_videomode()* afterwards. This also applies when the desired video mode has been set already before calling SV_OPTION_ANCCOMPLETE. If the video mode initialization is not performed, a proper operation cannot be ensured.

Flags:

- SV_ANCCOMPLETE_FLAG_FORCE_SWITCHINGLINE – Can be used together with SV_ANCCOMPLETE_STREAMER. The switching line is ignored during record in streamer mode. You can activate the capturing of the switching line with this flag.

Note:

The SV_ANCCOMPLETE_STREAMER setting is supported on Centaurus II only.

The SV_ANCCOMPLETE_STREAMER setting works in SDTV rasters only. To stream the ANC data in other rasters we recommend to use the setting SV_ANCCOMPLETE_ON in conjunction with the function *sv_fifo_anc()* (as described above).

#define SV_OPTION_ANCGENERATOR

This define details what the ANC embedder should insert.

- SV_ANCDATA_DEFAULT – Uses the default value. Default is normally the RP188 timecode.
- SV_ANCDATA_DISABLE – No ANC data will be embedded, i.e. it is disabled completely.
- SV_ANCDATA_USERDEF – Uses the user-defined ANC data. To be used mainly with the FIFO API.
- SV_ANCDATA_RP188 – Sends out RP188 DVITC/DLTC timecode.
- SV_ANCDATA_RP201 – Sends out RP201 film timecode (superset of RP188 with no DLTC).
- SV_ANCDATA_RP196 – Sends out RP196 timecode.
- SV_ANCDATA_RP196LTC – Sends out RP196 DLTC timecode
- SV_ANCDATA_FLAG_NOLTC – Disables DLTC generation.
- SV_ANCDATA_FLAG_NOSMPTE352 – Disables SMPTE352 generation.

#define SV_OPTION_ANCGENERATOR_RP165

If this define is set, the DVS video board will generate a checksum according to RP165 and place it at a certain position in the SDI signal. An already available RP165 checksum will be overwritten.

Note:

This feature is available on Centaurus II only and requires a special firmware. If called with a firmware where it is not supported, this define will return `SV_ERROR_WRONG_HARDWARE`.

#define SV_OPTION_ANCREADER

This define details the timecode type that the ANC reader should use. For possible values see the define `SV_OPTION_ANCGENERATOR`.

#define SV_OPTION_ANCUSER_DID

This define sets the DID for the ANC data to be inserted or captured in a user-defined mode.

#define SV_OPTION_ANCUSER_FLAGS

This define sets additional flags in a user-defined mode:

- `0` – Sends the ANC user data out in HANC (default).
- `SV_ANCUSER_FLAG_VANC` – Sends the ANC user data out in VANC.

#define SV_OPTION_ANCUSER_LINENR

This define determines the line number where the ANC data should be inserted in a user-defined mode. It should be set first, followed by the defines `SV_OPTION_ANCUSER_DID`, `SV_OPTION_ANCUSER_SDID` and `SV_OPTION_ANCUSER_FLAGS`.

#define SV_OPTION_ANCUSER_SDID

This define sets the SDID for the ANC data to insert/capture in a user-defined mode.

#define SV_OPTION_APROD_TC

This define sets the analog RP201 production timecode. It can only be used together with the function *sv_option_setat()*.

See also:

The define `SV_QUERY_APROD_TC`.

#define SV_OPTION_APROD_UB

This define sets the analog RP201 production user bytes. It can only be used together with the function *sv_option_setat()*.

See also:

The define `SV_QUERY_APROD_UB`.

#define SV_OPTION_ASSIGN_LTCA

This define assigns the LTC timecode of the first LTC (LTC A) to another jack than the default jack zero (`0`). It will be valid for the respective I/O channel, i.e. for the pair of input and output jacks. To use it the multi-channel operation mode has to be activated.

A second LTC (LTC B) is, for example, available on the Atomix Breakout Box 4 x SDI I/O+A. It is always assigned to the second I/O channel.

Note:

This feature is available on Centaurus II and Atomix.

See also:

The define SV_OPTION_MULTICHANNEL.

#define SV_OPTION_DLTC_TC

This define sets the DLTC (RP196/RP188/RP201) timecode. It can only be used together with the function *sv_option_setat()*.

See also:

The define SV_QUERY_DLTC_TC.

#define SV_OPTION_DLTC_UB

This define sets the DLTC (RP196/RP188/RP201) user bytes. It can only be used together with the function *sv_option_setat()*.

See also:

The define SV_QUERY_DLTC_UB.

#define SV_OPTION_DVITC_TC

This define sets the DVITC (RP196/RP188/RP201) timecode. It can only be used together with the function *sv_option_setat()*.

See also:

The define SV_QUERY_DVITC_TC.

#define SV_OPTION_DVITC_UB

This define sets the DVITC (RP196/RP188/RP201) user bytes. It can only be used together with the function *sv_option_setat()*.

See also:

The define SV_QUERY_DVITC_UB.

#define SV_OPTION_FILM_TC

This define sets the RP201 film timecode. It can only be used together with the function *sv_option_setat()*.

See also:

The define SV_QUERY_FILM_TC.

#define SV_OPTION_FILM_UB

This define sets the RP201 film user bytes. It can only be used together with the function *sv_option_setat()*.

See also:

The define SV_QUERY_FILM_UB.

#define SV_OPTION_FLUSH_TIMECODE

This define flushes all timecodes set via the function *sv_option_setat()*.

#define SV_OPTION_LTC_TC

This define sets the analog LTC timecode. It can only be used together with the function *sv_option_setat()*.

See also:

The define SV_QUERY_LTCTIMECODE.

#define SV_OPTION_LTC_UB

This define sets the analog LTC user bytes. It can only be used together with the function *sv_option_setat()*.

See also:

The define SV_QUERY_LTCUSERBYTES.

#define SV_OPTION_LTCDELAY

With this define you can set a delay for the incoming LTC timecode. The unit is in frames.

#define SV_OPTION_LTCDROPFRAME

This define enables or disables the drop-frame behavior in the LTC generator. It works in combination with the settings SV_LTCSOURCE_MASTER, SV_LTCSOURCE_INTERN and SV_LTCSOURCE_FREERUNNING:

- SV_LTCDROPFRAME_DEFAULT – Does not change the behavior (default).
- SV_LTCDROPFRAME_OFF – Disables the drop-frame timecode.
- SV_LTCDROPFRAME_ON – Enables the drop-frame timecode.

See also:

The define SV_OPTION_LTCSOURCE.

#define SV_OPTION_LTCFILTER

This define enables or disables the filtering of LTC timecode values that are invalid:

- SV_LTCFILTER_ENABLED – Enables the filtering.
- SV_LTCFILTER_DISABLED – Disables the filtering.

#define SV_OPTION_LTCOFFSET

This define sets an offset for the timecode sent out over the LTC connection. It can be used in conjunction with some settings of the define SV_OPTION_LTCSOURCE.

#define SV_OPTION_LTCSOURCE

This define sets the source for the timecode sent out over the LTC connection:

- SV_LTCSOURCE_DEFAULT – Uses the timecode value from the FIFO API.
- SV_LTCSOURCE_INTERN – Uses the internal timecode.
- SV_LTCSOURCE_MASTER – Uses the current timecode from the VTR master.

- SV_LTCSOURCE_FREERUNNING – Freerunning, can be reset with the define SV_OPTION_LTCOFFSET.
- SV_LTCSOURCE_LTCOFFSET – Sets an offset for the LTC timecode or resets the timecode in the freerunning state. When using this setting specify the value with the define SV_OPTION_LTCOFFSET.

#define SV_OPTION_PROD_TC

This define sets the RP201 production timecode. It can only be used together with the function *sv_option_setat()*.

See also:

The define SV_QUERY_PROD_TC.

#define SV_OPTION_PROD_UB

This define sets the RP201 production user bytes. It can only be used together with the function *sv_option_setat()*.

See also:

The define SV_QUERY_PROD_UB.

#define SV_OPTION_VITC_TC

This define sets the analog VITC timecode. It can only be used together with the function *sv_option_setat()*.

See also:

The define SV_QUERY_VITCTIMECODE.

#define SV_OPTION_VITC_UB

This define sets the analog VITC user bytes. It can only be used together with the function *sv_option_setat()*.

See also:

The define SV_QUERY_VITCUSERBYTES.

#define SV_OPTION_VITCLINE

This define sets the line number where to send out the analog VITC. Possible values can be the ones detailed below or a directly specified line number. When specifying a line number directly, only the appropriate lines of the respective raster where VITC is allowed can be set, i.e. in PAL the lines 8..22 and in NTSC 12..19.

Additionally, you can enable a three-line VITC when setting the flag SV_VITCLINE_ARP201 together with a value.

Values:

- SV_VITCLINE_DEFAULT – Outputs the VITC on the default line of the raster, i.e. in PAL on line 19 and in NTSC on line 14.
- SV_VITCLINE_DISABLED – Disables the VITC output.

Flags:

- SV_VITCLINE_ARP201 – Enables the full three-line VITC according to RP201.

Note:

Analog VITC is available in SDTV rasters only.

See also:

The `SV_QUERY_VITC<xxx>` defines.

#define SV_OPTION_VITCREADERLINE

This define enables or disables the VITC reader of the DVS video device, meaning it enables or disables the capturing of the analog VITC.

On most DVS video devices an enabled VITC reader will detect the line where the VITC is embedded automatically, and it will be captured accordingly.

Values:

- `SV_VITCLINE_DEFAULT` – Enables an autodetection of the VITC line on most DVS video devices.
- `SV_VITCLINE_DISABLED` – Disables the VITC reader.

Note:

Analog VITC is available in SDTV rasters only.

See also:

The define `SV_QUERY_VITCREADERLINE`.

#define SV_OPTION_VTR_INFO

This define has to be used when acting as a slave: It sets the first part of the VTR's status data (info bits). It can only be used together with the function *sv_option_setat()*.

#define SV_OPTION_VTR_INFO2

This define has to be used when acting as a slave: It sets the second part of the VTR's status data (info bits). It can only be used together with the function *sv_option_setat()*.

#define SV_OPTION_VTR_INFO3

This define has to be used when acting as a slave: It sets the third part of the VTR's status data (info bits). It can only be used together with the function *sv_option_setat()*.

#define SV_OPTION_VTR_TC

This define has to be used when acting as a slave: It sets the VTR timecode. It can only be used together with the function *sv_option_setat()*.

#define SV_OPTION_VTR_UB

This define has to be used when acting as a slave: It sets the VTR user bytes. It can only be used together with the function *sv_option_setat()*.

#define SV_QUERY_AFILM_TC

This query returns the current analog RP201 film timecode (see the define `SV_OPTION_AFILM_TC`). Additionally, the parameter *par* of the function *sv_query()* has to be set to `-1` to read this.

#define SV_QUERY_AFILM_UB

This query returns the current analog RP201 film user bytes (see the define SV_OPTION_AFILM_UB). Additionally, the parameter *par* of the function *sv_query()* has to be set to -1 to read this.

#define SV_QUERY_ANC_MAXHANCLINENR

This define returns the maximum line number to include HANC data in the current video stream.

#define SV_QUERY_ANC_MAXVANCLINENR

This define returns the maximum line number to include VANC data in the current video stream.

#define SV_QUERY_ANC_MINLINENR

This define returns the minimum line number to include ANC data in the current video stream (VANC and HANC).

#define SV_QUERY_APROD_TC

This query returns the current analog RP201 production timecode (see the define SV_OPTION_APROD_TC). Additionally, the parameter *par* of the function *sv_query()* has to be set to -1 to read this.

#define SV_QUERY_APROD_UB

This query returns the current analog RP201 production user bytes (see the define SV_OPTION_APROD_UB). Additionally, the parameter *par* of the function *sv_query()* has to be set to -1 to read this.

#define SV_QUERY_DLTC_TC

This query returns the current DLTC (RP196/RP188/RP201) timecode (see the define SV_OPTION_DLTC_TC). Additionally, the parameter *par* of the function *sv_query()* has to be set to -1 to read this.

#define SV_QUERY_DLTC_UB

This query returns the current DLTC (RP196/RP188/RP201) user bytes (see the define SV_OPTION_DLTC_UB). Additionally, the parameter *par* of the function *sv_query()* has to be set to -1 to read this.

#define SV_QUERY_DVITC_TC

This query returns the current DVITC (RP196/RP188/RP201) timecode (see the define SV_OPTION_DVITC_TC). Additionally, the parameter *par* of the function *sv_query()* has to be set to -1 to read this.

#define SV_QUERY_DVITC_UB

This query returns the current DVITC (RP196/RP188/RP201) user bytes (see the define SV_OPTION_DVITC_UB). Additionally, the parameter *par* of the function *sv_query()* has to be set to -1 to read this.

#define SV_QUERY_FILM_TC

This query returns the current RP201 film timecode (see the define SV_OPTION_FILM_TC). Additionally, the parameter *par* of the function *sv_query()* has to be set to -1 to read this.

#define SV_QUERY_FILM_UB

This query returns the current RP201 film user bytes (see the define SV_OPTION_FILM_UB). Additionally, the parameter *par* of the function *sv_query()* has to be set to -1 to read this.

#define SV_QUERY_LTCAVAILABLE

This define returns TRUE if the given I/O channel is able to send and receive LTC timecodes. The result of this query can be influenced with the define SV_OPTION_ASSIGN_LTCA.

#define SV_QUERY_LTCDROPFRAME

This query returns the setting of the LTC drop-frame timecode. See the define SV_OPTION_LTCDROPFRAME.

#define SV_QUERY_LTCFILTER

This query returns the setting of the LTC filtering. See the define SV_OPTION_LTCFILTER.

#define SV_QUERY_LTCOFFSET

This define returns the setting of the LTC offset. See the define SV_OPTION_LTCOFFSET.

#define SV_QUERY_LTCSOURCE

This define returns the setting of the LTC source. See the define SV_OPTION_LTCSOURCE.

#define SV_QUERY_LTCTIMECODE

This query returns the current analog LTC timecode (see the define SV_OPTION_LTC_TC). Additionally, the parameter *par* of the function *sv_query()* has to be set to -1 to read this.

#define SV_QUERY_LTCUSERBYTES

This query returns the current analog LTC user bytes (see the define SV_OPTION_LTC_UB). Additionally, the parameter *par* of the function *sv_query()* has to be set to -1 to read this.

#define SV_QUERY_PROD_TC

This query returns the current RP201 production timecode (see the define SV_OPTION_PROD_TC). Additionally, the parameter *par* of the function *sv_query()* has to be set to -1 to read this.

#define SV_QUERY_PROD_UB

This query returns the current RP201 production user bytes (see the define SV_OPTION_PROD_UB). Additionally, the parameter *par* of the function *sv_query()* has to be set to -1 to read this.

#define SV_QUERY_VALIDTIMECODE

This define returns a bit mask of valid timecodes.

#define SV_QUERY_VITCLINE

This define returns the number of the line where the VITC is sent out.

See also:

The define SV_OPTION_VITCLINE.

#define SV_QUERY_VITCREADERLINE

This define returns the number of the line where the VITC was found (automatic detection) or where it will be read.

See also:

The define SV_OPTION_VITCREADERLINE.

#define SV_QUERY_VITCTIMECODE

This query returns the current analog VITC timecode (see the define SV_OPTION_VITC_TC). Additionally, the parameter *par* of the function *sv_query()* has to be set to -1 to read this.

#define SV_QUERY_VITCUSERBYTES

This query returns the current analog VITC user bytes (see the define SV_OPTION_VITC_UB). Additionally, the parameter *par* of the function *sv_query()* has to be set to -1 to read this.

Function Documentation

int sv_timecode_feedback (sv_handle * *sv*, sv_timecode_info * *input*, sv_timecode_info * *output*)

This function retrieves the current timecodes directly from the DVS video device. If you only need one structure, you can fill the other pointer with zero (0).

Parameters:

sv – Handle returned from the function *sv_open()*.

input – Pointer to the structure *sv_timecode_info* that will be filled with all available input timecodes.

output – Pointer to the structure *sv_timecode_info* that will be filled with all available output timecodes.

Returns:

If the function succeeds, it returns SV_OK. Otherwise it will return the error code SV_ERROR_<xxx>.

API – GPI Functionality

Detailed Description

This chapter describes the defines and functions to access the GPI port of the DVS video device. The GPI can be set by using the FIFO API. Optionally the define SV_OPTION_GPI can be used to set the GPI output data. You can always read the GPI input with the function *sv_query()* in conjunction with the define SV_QUERY_GPI, i.e. with `sv_query(SV_QUERY_GPI)`.

The signal inputs and outputs of the GPI connector (9-pin D-Sub connector) are, for example, available at the front panel of a breakout box or the (optional) GPI slot panel.

Note:

A pin-out of the GPI port, if available, can be found in the installation guide of your DVS video board product.

Defines

- #define SV_OPTION_GPI
- #define SV_OPTION_GPIIN
- #define SV_OPTION_GPIOUT
- #define SV_QUERY_GPI
- #define SV_QUERY_GPIIN
- #define SV_QUERY_GPIOUT

Define Documentation

#define SV_OPTION_GPI

This define sets the output bits of the GPI port if the option call SV_OPTION_GPIOUT is set to `SV_GPIOUT_OPTIONGPI`.

#define SV_OPTION_GPIIN

This define configures the interpretation of incoming GPI data by the driver:

- `SV_GPIIN_IGNORE` – Ignore input. Data can be read back either in a FIFO or with the query SV_QUERY_GPI.

#define SV_OPTION_GPIOUT

This define configures the driver how to generate the outgoing GPI data:

- `SV_GPIOUT_DEFAULT` – Value, for example, taken from the FIFO API.
- `SV_GPIOUT_OPTIONGPI` – To set values use the define SV_OPTION_GPI.
- `SV_GPIOUT_INOUTPOINT` – When inpoint then `01`, when outpoint then `10`, else `00`.
- `SV_GPIOUT_PULLDOWN` – When phase A then `01`, else `00`.
- `SV_GPIOUT_PULLDOWNPHASE` – When phase A then `00`, when B then `01`, when C then `10`, when D then `11`.
- `SV_GPIOUT_REPEATED` – When current frame is repeated then `01`, else `00`.

#define SV_QUERY_GPI

This define returns the current GPI data. Additionally, the parameter *par* of the function *sv_query()* has to be set to `-1` to read this.

#define SV_QUERY_GPIIN

This define returns the setting of the GPI input. See the define `SV_OPTION_GPIIN`.

#define SV_QUERY_GPIOUT

This define returns the setting of the GPI output. See the define `SV_OPTION_GPIOUT`.

API – Proxy Capture

Detailed Description

The Proxy Capture API enables you to capture a downconverted version (proxy in SD) of the image at the output of the DVS video board. The demo program *proxy* provides examples that use these functions to grab the output image (see chapter Example Projects Overview). Please note that the downconverted image will be available in YUV422 8 bit only.

All SV_OPTION_PROXY_<xxx> settings detailed in the following affect the buffers returned by the *sv_capture()* function.

Defines

- #define SV_OPTION_PROXY_ASPECTRATIO
- #define SV_OPTION_PROXY_OPTIONS
- #define SV_OPTION_PROXY_OUTPUT
- #define SV_OPTION_PROXY_SYNCMODE
- #define SV_OPTION_PROXY_VIDEOMODE

Functions

- int sv_capture (sv_handle *sv, char *buffer, int buffersize, int lineoffset, int *pxsize, int *pysize, int *ptick, uint32 *pclockhigh, uint32 *pclocklow, int flags, sv_overlapped *poverlapped)
- int sv_capture_ready (sv_handle *sv, sv_overlapped *poverlapped, int *pxsize, int *pysize, int *ptick, uint32 *pclockhigh, uint32 *pclocklow)
- int sv_capture_status (sv_handle *sv, sv_capture_info *pinfo)
- int sv_captureex (sv_handle *sv, char *buffer, int buffersize, sv_capturebuffer *pcapture, int version, int flags, sv_overlapped *poverlapped)

Define Documentation

#define SV_OPTION_PROXY_ASPECTRATIO

This define sets the aspect ratio of the downconverted material. The value is expected as a fixed point float.

#define SV_OPTION_PROXY_OPTIONS

This define sets options related to the function *sv_capture()* and the downconverted output:

- SV_PROXY_OPTION_SDTVFULL – Sets the proxy output to give out full-sized SDTV material, otherwise it will be downscaled as well. This flag has an influence only when the general video output is set to an SDTV mode.

#define SV_OPTION_PROXY_OUTPUT

This define sets the output format for the proxy output:

- SV_PROXY_OUTPUT_UNDERSCAN – The complete image is shown with black around it.

- SV_PROXY_OUTPUT_LETTERBOX – A 16:9 image is shown with black bars on top and bottom.
- SV_PROXY_OUTPUT_CROPPED – A 16:9 image is cropped by a center cut (4:3).
- SV_PROXY_OUTPUT_ANAMORPH – Full screen anamorph output (4:3).

Note:

The *sv_capture()* function always returns the image buffer without any black borders.

#define SV_OPTION_PROXY_SYNCMODE

This define selects the sync mode of the proxy output:

- SV_PROXY_SYNC_INTERNAL – Sync mode is set to internal.
- SV_PROXY_SYNC_AUTO – Uses the sync configured for the SDI output. If this is not possible, the internal sync will be used.
- SV_PROXY_SYNC_GENLOCKED – Genlocked to an analog sync.

#define SV_OPTION_PROXY_VIDEOMODE

This define sets the video raster for the proxy output. It affects the video size of the buffers returned by the *sv_capture()* function as well:

- SV_MODE_PAL – PAL video raster.
- SV_MODE_NTSC – NTSC video raster.

Function Documentation

int sv_capture (sv_handle * *sv*, char * *buffer*, int *buffersize*, int *lineoffset*, int * *pxsize*, int * *pysize*, int * *ptick*, uint32 * *pclockhigh*, uint32 * *pclocklow*, int *flags*, sv_overlapped * *poverlapped*)

This function returns the last field/frame captured from the video output in a downscaled format. The format can be adjusted by various calls of SV_OPTION_PROXY_<xxx>.

This function is non-blocking. It returns the error code SV_ERROR_NOTREADY if there is no buffer available or if all buffers have been fetched by the caller. In such a case it is suggested to wait for the next vertical sync on the output pipeline (e.g. with the parameter SV_VSYNCWAIT_DISPLAY of the function *sv_vsyncwait()*).

Parameters:

sv – Handle returned from the function *sv_open()*.

buffer – Address to the memory buffer to receive the buffer.

buffersize – Size of the memory pointed to by *buffer*.

lineoffset – Line offset in the memory.

pxsize – Pointer to the integer which returns the x-size of the image.

pysize – Pointer to the integer which returns the y-size of the image.

ptick – Pointer to the integer which returns the tick when the image was captured.

pclockhigh – Pointer to the integer which returns the high 32 bits of the 64-bit clock when the image was captured.

pclocklow – Pointer to the integer which returns the low 32 bits of the 64-bit clock when the image was captured.

poverlapped – Either NULL or the pointer to the structure *sv_overlapped*. If set, you must use the function *sv_capture_ready()* to check the returned parameters.

flags – Currently not used. Has to be set to zero (0).

Returns:

If the function succeeds, it returns SV_OK. Otherwise it will return the error code SV_ERROR_<xxx>.

See also:

The function *sv_captureex()*.

int sv_capture_ready (sv_handle * *sv*, sv_overlapped * *poverlapped*, int * *pxsize*, int * *pysize*, int * *ptick*, uint32 * *pclockhigh*, uint32 * *pclocklow*)

In case you have called the function *sv_capture()* initially with an enabled *poverlapped*, you can pick up the buffer contents placed into *sv_capture()* with this function. It has to be called as soon as the overlapped event receives the signal that the function *sv_capture()* is ready.

Parameters:

sv – Handle returned from the function *sv_open()*.

poverlapped – Pointer to the structure *sv_overlapped*.

pxsize – Pointer to the integer that receives the x-size of the captured image.

pysize – Pointer to the integer that receives the y-size of the captured image.

ptick – Pointer to the integer that receives the display tick when the image was captured.

pclockhigh – Pointer to the integer that receives the clock of the MSBs (most significant bytes) when the image was captured.

pclocklow – Pointer to the integer that receives the clock of the LSBs (least significant bytes) when the image was captured.

Returns:

If the function succeeds, it returns SV_OK. Otherwise it will return the error code SV_ERROR_<xxx>.

int sv_capture_status (sv_handle * *sv*, sv_capture_info * *pinfo*)

This function retrieves the structure *sv_capture_info* which contains information about the downscaled buffer on the DVS video device memory. These information will be helpful during an *sv_capture()* operation.

Parameters:

sv – Handle returned from the function *sv_open()*.

pinfo – Contains information about the capture buffer format.

Returns:

If the function succeeds, it returns SV_OK. Otherwise it will return the error code SV_ERROR_<xxx>.

int sv_captureex (sv_handle * *sv*, char * *buffer*, int *buffersize*, sv_capturebuffer * *pcapture*, int *version*, int *flags*, sv_overlapped * *poverlapped*)

This function returns the last field/frame captured from the video output in a downscaled format. The format can be adjusted by various calls of SV_OPTION_PROXY_<xxx>. It works similar to the function *sv_capture()*, but provides in addition, depending on the version parameter, extended information for each returned field/frame via the structure *sv_capturebuffer*.

This function is non-blocking. It returns the error code SV_ERROR_NOTREADY if there is no buffer available or if all buffers have been fetched by the caller. In such a case it is suggested to

wait for the next vertical sync on the output pipeline (e.g. with the parameter `SV_VSYNCWAIT_DISPLAY` of the function *sv_vsyncwait()*).

Parameters:

sv – Handle returned from the function *sv_open()*.

buffer – Address to the memory buffer to receive the buffer.

buffersize – Size of the memory pointed to by *buffer*.

pcapture – Pointer to the structure *sv_capturebuffer*.

version – Determines the elements of the structure *sv_capturebuffer* that will be filled with data. If set to zero (`0`) or one (`1`), the default elements will be filled. If set to two (`2`), the default elements as well as additional timecode elements will be filled. Versions greater two are for DVS internal use only.

flags – Currently not used. Has to be set to zero (`0`).

poverlapped – Either `NULL` or the pointer to the structure *sv_overlapped*. If set, you must use the function *sv_capture_ready()* to check the returned parameters.

Returns:

If the function succeeds, it returns `SV_OK`. Otherwise it will return the error code `SV_ERROR_<xxx>`.

See also:

The function *sv_capture()*.

API – Hardware

Detailed Description

This chapter details defines and functions to control hardware related features of the DVS video device.

Defines

- #define SV_QUERY_DMAALIGNMENT
- #define SV_QUERY_FANSPEED
- #define SV_QUERY_HW_CARDOPTIONS
- #define SV_QUERY_HW_CARDVERSION
- #define SV_QUERY_HW_EPLDOPTIONS
- #define SV_QUERY_HW_EPLDVERSION
- #define SV_QUERY_HW_PCIELANES
- #define SV_QUERY_HW_PCISPEED
- #define SV_QUERY_HW_PCIWIDTH
- #define SV_QUERY_SERIALNUMBER
- #define SV_QUERY_TEMPERATURE
- #define SV_QUERY_VOLTAGE_12V0
- #define SV_QUERY_VOLTAGE_1V0
- #define SV_QUERY_VOLTAGE_1V2
- #define SV_QUERY_VOLTAGE_1V5
- #define SV_QUERY_VOLTAGE_1V8
- #define SV_QUERY_VOLTAGE_2V5
- #define SV_QUERY_VOLTAGE_3V3
- #define SV_QUERY_VOLTAGE_5V0

Define Documentation

#define SV_QUERY_DMAALIGNMENT

This define returns the minimum alignment needed for DMA transfers (see also chapter API – FIFO API).

#define SV_QUERY_FANSPEED

This define returns the speed of the fan on the DVS video board (unit is in r.p.m.).

#define SV_QUERY_HW_CARDOPTIONS

This define returns the options bit mask of the DVS video board.

#define SV_QUERY_HW_CARDVERSION

This define returns the hardware board version of the DVS video device.

#define SV_QUERY_HW_EPLDOPTIONS

This define returns the hardware EPLD options bit mask.

#define SV_QUERY_HW_EPLDVERSION

This define returns the hardware EPLD version.

#define SV_QUERY_HW_PCIELANES

This define returns the number of PCIe lanes currently available for the hardware. It can be used, for example, to verify that the hardware uses all PCIe lanes available and is working properly. In case of PCI or PCI-X boards it returns zero (0).

#define SV_QUERY_HW_PCISPEED

This define returns the speed of the PCI bus in megahertz (MHz).

#define SV_QUERY_HW_PCIWIDTH

This define returns the width of the PCI bus.

#define SV_QUERY_SERIALNUMBER

This define returns the device's serial number.

Note:

The serial numbers of the supported DVS video board products (first digits) are listed in the section Supported DVS Video Board Products.

#define SV_QUERY_TEMPERATURE

This define returns the measured board temperature in Celsius (value is fixed float).

#define SV_QUERY_VOLTAGE_12V0

This define returns the measured 12 V (value is fixed float).

#define SV_QUERY_VOLTAGE_1V0

This define returns the measured 1.0 V (value is fixed float).

#define SV_QUERY_VOLTAGE_1V2

This define returns the measured 1.2 V (value is fixed float).

#define SV_QUERY_VOLTAGE_1V5

This define returns the measured 1.5 V (value is fixed float).

#define SV_QUERY_VOLTAGE_1V8

This define returns the measured 1.8 V (value is fixed float).

#define SV_QUERY_VOLTAGE_2V5

This define returns the measured 2.5 V (value is fixed float).

#define SV_QUERY_VOLTAGE_3V3

This define returns the measured 3.3 V (value is fixed float).

#define SV_QUERY_VOLTAGE_5V0

This define returns the measured 5 V (value is fixed float).

API – Tracing

Detailed Description

This chapter describes the DVSOEM debug tracing.

With the debug version of the DVSOEM library (*dvsoemdbg.dll* or *libdvsoemdbg.a*) you can enable a tracing of *sv_<xxx>* calls in your application.

To use this under Windows download either the *dbgview* program (www.sysinternals.com), use the kernel debugger or see the output in Visual Studio. Make a backup of the file *dvsoem.dll* and replace its original with the file *dvsoemdbg.dll*. For this copy the *dvsoemdbg.dll* into the directory of the *dvsoem.dll* and rename it to 'dvsoem.dll'.

Under UNIX you have to use the file *libdvsoemdbg.a* for this instead of *libdvsoem.a*. The trace output will be sent to 'stdout'.

For each debug trace session you have to enable the debug output anew. This can be done either with the command `svram trace -1` for all outputs or with the define `SV_OPTION_TRACE` in your application directly.

Defines

- #define SV_OPTION_TRACE

Define Documentation

#define SV_OPTION_TRACE

This define enables the DVSOEM debug tracing.

API – Storage Functions

Detailed Description

This chapter provides a description of basic defines and functions to control the DVS video device. They work on the storage (memory) of the DVS video board and are not intended for real-time transfers to or from the system memory. Additionally, they should not be used in conjunction with the FIFO API.

Note:

The functions described in this chapter are obsolete. They were used in a no longer supported operation mode of the DVS SDK, the RAM-recorder mode. However, because most of them still work, they can be used in such a mode for testing purposes, for example, to check the basic functionality of the DVS video device.

Do not use these defines and functions together with the FIFO API.

Defines

- #define SV_OPTION_LOOPMODE
- #define SV_OPTION_REPEAT
- #define SV_OPTION_SLOWMOTION
- #define SV_OPTION_SPEED
- #define SV_OPTION_SPEEDBASE
- #define SV_QUERY_AUDIOSIZE
- #define SV_QUERY_AUDIOSIZE_FROMHOST
- #define SV_QUERY_AUDIOSIZE_TOHOST
- #define SV_QUERY_INTERLACEID_STORAGE
- #define SV_QUERY_INTERLACEID_VIDEO
- #define SV_QUERY_LOOPMODE
- #define SV_QUERY_PRESET
- #define SV_QUERY_REPEATMODE
- #define SV_QUERY_SLOWMOTION
- #define SV_QUERY_STREAMERSIZE

Functions

- int sv_black (sv_handle *sv)
- int sv_colorbar (sv_handle *sv)
- int sv_display (sv_handle *sv, char *memp, int memsize, int xsize, int ysize, int start, int nframes, int tc)
- int sv_goto (sv_handle *sv, int frame)
- int sv_host2sv (sv_handle *sv, char *buffer, int buffersize, int xsize, int ysize, int frame, int nframes, int mode)
- int sv_inpoint (sv_handle *sv, int frame)
- int sv_live (sv_handle *sv)
- int sv_outpoint (sv_handle *sv, int frame)
- int sv_position (sv_handle *sv, int frame, int field, int repeat, int flags)
- int sv_preset (sv_handle *sv, int preset)

- int sv_record (sv_handle *sv, char *memp, int memsize, int *pxsize, int *pysize, int start, int nframes, int tc)
- int sv_showinput (sv_handle *sv, int showinput, int spare)
- int sv_stop (sv_handle *sv)
- int sv_sv2host (sv_handle *sv, char *buffer, int buffersize, int xsize, int ysize, int frame, int nframes, int mode)

Define Documentation

#define SV_OPTION_LOOPMODE

This define sets the loop mode:

- `SV_LOOPMODE_FORWARD` – Infinite display forward.
- `SV_LOOPMODE_REVERSE` – Infinite display reverse.
- `SV_LOOPMODE_SHUTTLE` – Display forward and reverse between in- and outpoint.
- `SV_LOOPMODE_ONCE` – Display once and stop at outpoint.
- `SV_LOOPMODE_DEFAULT` – Default is `SV_LOOPMODE_INFINITE`.
- `SV_LOOPMODE_INFINITE` – Same as `SV_LOOPMODE_FORWARD`.

Note:

This define is obsolete but can be used for testing purposes (see the introduction to this chapter).

#define SV_OPTION_REPEAT

This define configures the display when the speed is zero (`0`):

- `SV_REPEAT_FRAME` – Displays the complete frame.
- `SV_REPEAT_FIELD1` – Displays field 1 only.
- `SV_REPEAT_FIELD2` – Displays field 2 only.
- `SV_REPEAT_CURRENT` – Displays the current field.
- `SV_REPEAT_DEFAULT` – Default is `SV_REPEAT_FRAME`.

Note:

This define is obsolete but can be used for testing purposes (see the introduction to this chapter).

#define SV_OPTION_SLOWMOTION

This define sets the slow motion mode, i.e. when the speed is smaller one (`1`):

- `SV_SLOWMOTION_FRAME` – Displays the complete frame.
- `SV_SLOWMOTION_FIELD` – Displays the current field.
- `SV_SLOWMOTION_FIELD1` – Displays field 1 only.
- `SV_SLOWMOTION_FIELD2` – Displays field 2 only.

Note:

This define is obsolete but can be used for testing purposes (see the introduction to this chapter).

#define SV_OPTION_SPEED

This define sets the speed. The denominator is by default `0x10000` but can be adjusted with the define `SV_OPTION_SPEEDBASE`.

Note:

This define is obsolete but can be used for testing purposes (see the introduction to this chapter).

#define SV_OPTION_SPEEDBASE

This define sets the base denominator. The denominator is by default `0x10000`.

Note:

This define is obsolete but can be used for testing purposes (see the introduction to this chapter).

#define SV_QUERY_AUDIOSIZE

This define returns the size of an audio buffer for the specified frame (for *sv_sv2host()* and *sv_host2sv()* transfers).

Note:

This define is obsolete but can be used for testing purposes (see the introduction to this chapter).

#define SV_QUERY_AUDIOSIZE_FROMHOST

Same as `SV_QUERY_AUDIOSIZE`.

Note:

This define is obsolete but can be used for testing purposes (see the introduction to this chapter).

#define SV_QUERY_AUDIOSIZE_TOHOST

Same as `SV_QUERY_AUDIOSIZE`.

Note:

This define is obsolete but can be used for testing purposes (see the introduction to this chapter).

#define SV_QUERY_INTERLACEID_STORAGE

This query returns the interlace-ID (scanning mode) of the storage:

- `1` – Progressive.
- `12` – Interlaced.
- `21` – Interlaced with field 2 on top.

Note:

This define is obsolete but can be used for testing purposes (see the introduction to this chapter).

See also:

The define `SV_QUERY_INTERLACEID_VIDEO`.

#define SV_QUERY_INTERLACEID_VIDEO

This query returns the interlace-ID (scanning mode) of the video signal. For possible return values see SV_QUERY_INTERLACEID_STORAGE.

Note:

This define is obsolete but can be used for testing purposes (see the introduction to this chapter).

#define SV_QUERY_LOOPMODE

This query returns the setting of SV_OPTION_LOOPMODE.

Note:

This define is obsolete but can be used for testing purposes (see the introduction to this chapter).

#define SV_QUERY_PRESET

This query returns the preset setting. See the function *sv_preset()*.

Note:

This define is obsolete but can be used for testing purposes (see the introduction to this chapter).

#define SV_QUERY_REPEATMODE

This define returns the setting of SV_OPTION_REPEAT.

Note:

This define is obsolete but can be used for testing purposes (see the introduction to this chapter).

#define SV_QUERY_SLOWMOTION

This define returns the setting of SV_OPTION_SLOWMOTION.

Note:

This define is obsolete but can be used for testing purposes (see the introduction to this chapter).

#define SV_QUERY_STREAMERSIZE

This define returns the size of one video frame.

Note:

This define is obsolete but can be used for testing purposes (see the introduction to this chapter).

Function Documentation

int sv_black (sv_handle * *sv*)

This function displays a black frame at the video output.

Parameters:

sv – Handle returned from the function *sv_open()*.

Returns:

If the function succeeds, it returns `SV_OK`. Otherwise it will return the error code `SV_ERROR_<xxx>`.

Note:

This define is obsolete but can be used for testing purposes (see the introduction to this chapter).

int sv_colorbar (sv_handle * *sv*)

This function displays a color bar at the video output.

Parameters:

sv – Handle returned from the function *sv_open()*.

Returns:

If the function succeeds, it returns `SV_OK`. Otherwise it will return the error code `SV_ERROR_<xxx>`.

Note:

This define is obsolete but can be used for testing purposes (see the introduction to this chapter).

int sv_display (sv_handle * *sv*, char * *memp*, int *memsize*, int *xsize*, int *ysize*, int *start*, int *nframes*, int *tc*)

This function starts a display operation of the material in the storage.

Parameters:

sv – Handle returned from the function *sv_open()*.

memp – Obsolete, must be `NULL`.

memsize – Obsolete, must be zero (`0`).

xsize – Obsolete, it will be ignored.

ysize – Obsolete, it will be ignored.

start – Indicates the frame number (start frame) where the display will start.

nframes – Sets the number of frames to be displayed.

tc – If not zero (`0`), an autoedit record will be performed on a connected external device (e.g. VTR) at the indicated position (timecode).

Returns:

If the function succeeds, it returns `SV_OK`. Otherwise it will return the error code `SV_ERROR_<xxx>`.

Note:

This define is obsolete but can be used for testing purposes (see the introduction to this chapter).

Example:

```
int example_display(sv_handle * sv, int start, int nframes)
{
  int res;

  res = sv_display(sv, buffer, 0, 0, 0, start, nframes, 0);
```

```
    if(res != SV_OK) {
      printf("Display example: sv_display() returned %d '%s'\n", res,
sv_geterrortext(res));
    }

    return res;
  }
```

int sv_goto (sv_handle * *sv*, int *frame*)

This function moves the current position to a certain frame and gives out a still picture (sets the speed to zero (0)). The still picture display is performed either with field or frame repetition, depending on the current setting for the repeat mode set with `sv_option(sv, SV_OPTION_REPEAT, SV_REPEAT_<xxx>)`.

Parameters:

sv – Handle returned from the function *sv_open()*.

frame – Number of the frame to jump to.

Returns:

If the function succeeds, it returns `SV_OK`. Otherwise it will return the error code `SV_ERROR_<xxx>`.

Note:

This define is obsolete but can be used for testing purposes (see the introduction to this chapter).

See also:

The function *sv_position()*.

int sv_host2sv (sv_handle * *sv*, char * *buffer*, int *buffersize*, int *xsize*, int *ysize*, int *frame*, int *nframes*, int *mode*)

This function transfers data from the system memory to the video device.

Parameters:

sv – Handle returned from the function *sv_open()*.

buffer – Pointer to the system memory area that shall be transferred.

buffersize – Buffer size of the data to be transferred to the video device. Must be greater than or equal to (>=) the size of one frame.

xsize – X-size corresponding to the current video raster for the video data. For audio data it must be 2 x 48000 / fps, for example, for 25 fps it is 3840 and for 29.97 Hz it alternates between 3200/3204. For pulldown clips the amount of data is 4004 or 1001 samples per frame.

ysize – Y-size corresponding to the current video raster for video data. This value is not evaluated for audio data.

frame – Frame number of the frame on the video device that shall be replaced by the transferred data. Must be a valid storage page.

nframes – Obsolete. Must be one (1).

mode – Combination of qualifiers that form the current raster. As qualifiers set a data type and data size. See lists below.

Parameters for *mode* (Data Types):

- `SV_TYPE_MONO` – Monochrome video (SDTV devices only).
- `SV_TYPE_YUV422` – YUV422 video.
- `SV_TYPE_YUV422A` – YUV422A video.

- SV_TYPE_YUV444 – YUV444 video.
- SV_TYPE_YUV444A – YUV444A video.
- SV_TYPE_RGB – RGB video.
- SV_TYPE_RGBA – RGBA video.
- SV_TYPE_AUDIO12 – Audio data from the first channel pair.
- SV_TYPE_AUDIO34 – Audio data from the second channel pair.
- SV_TYPE_AUDIO56 – Audio data from the third channel pair.
- SV_TYPE_AUDIO78 – Audio data from the fourth channel pair.
- SV_TYPE_AUDIO9a – Audio data from the fifth channel pair.
- SV_TYPE_AUDIObc – Audio data from the sixth channel pair.
- SV_TYPE_AUDIOde – Audio data from the seventh channel pair.
- SV_TYPE_AUDIOfg – Audio data from the eighth channel pair.
- SV_TYPE_KEY – Key data, monochrome.
- SV_TYPE_STREAMER – Streamer video data.

Parameters for *mode* (Data Sizes):

- SV_DATASIZE_8BIT – 8 bit.
- SV_DATASIZE_10BIT – 10 bit.
- SV_DATASIZE_16BIT_BIG – 16 bit, big endian.
- SV_DATASIZE_16BIT_LITTLE – 16 bit, little endian.
- SV_DATASIZE_32BIT_BIG – 32 bit, big endian.
- SV_DATASIZE_32BIT_LITTLE – 32 bit, little endian.
- SV_DATASIZE_MASK – Mask for the data size.

Returns:

If the function succeeds, it returns SV_OK. Otherwise it will return the error code SV_ERROR_<xxx>.

Note:

This define is obsolete but can be used for testing purposes (see the introduction to this chapter).

See also:

The function *sv_sv2host()*.

Example:

```
  int example_clearwithbuffer(sv_handle * handle, char * buffer, int buffersize,
int start, int nframes)
  {
    sv_info info;
    int count;

    res = sv_status(sv, &info);
    if(res != SV_OK) {
      printf("Error: sv_status() returned %d '%s'\n", res, sv_geterrortext(res));
    }

    for(count = 0; (count < nframes) && (res == SV_OK); count++) {
      res = sv_host2sv(sv, buffer, buffersize, info.setup.storagexsize,
info.setup.storageysize, start + count, 1, 0);
      if(res != SV_OK) {
        printf("Error: sv_host2sv() returned %d '%s'\n", res,
sv_geterrortext(res));
      }
    }

    return res;
  }
```

int sv_inpoint (sv_handle * *sv*, int *frame*)

This function sets the inpoint for subsequent *sv_display()* or *sv_record()* operations.

Parameters:

sv – Handle returned from the function *sv_open()*.

frame – Frame number of the frame that should be the new inpoint.

Returns:

If the function succeeds, it returns `SV_OK`. Otherwise it will return the error code `SV_ERROR_<xxx>`.

Note:

This define is obsolete but can be used for testing purposes (see the introduction to this chapter).

See also:

The function *sv_outpoint()*.

int sv_live (sv_handle * *sv*)

This function enables the live mode (EE).

Parameters:

sv – Handle returned from the function *sv_open()*.

Returns:

If the function succeeds, it returns `SV_OK`. Otherwise it will return the error code `SV_ERROR_<xxx>`.

Note:

This define is obsolete but can be used for testing purposes (see the introduction to this chapter).

See also:

The function *sv_showinput()*.

int sv_outpoint (sv_handle * *sv*, int *frame*)

This function sets the outpoint for subsequent *sv_display()* or *sv_record()* operations.

Parameters:

sv – Handle returned from the function *sv_open()*.

frame – Frame number of the frame that should be the new outpoint.

Returns:

If the function succeeds, it returns `SV_OK`. Otherwise it will return the error code `SV_ERROR_<xxx>`.

Note:

This define is obsolete but can be used for testing purposes (see the introduction to this chapter).

See also:

The function *sv_inpoint()*.

int sv_position (sv_handle * *sv*, int *frame*, int *field*, int *repeat*, int *flags*)

This function performs the same operation as the function *sv_goto()*, i.e. it moves the current position to a certain frame and performs a still picture display (sets the speed to zero (0)). However, it offers more options than *sv_goto()*.

Parameters:

sv – Handle returned from the function *sv_open()*.

frame – Number of the frame to jump to.

field – Number of the field to jump to.

repeat – Defines the repeat mode. For possible types see SV_OPTION_REPEAT.

flags – Optional. See list below.

Parameters for *flags*:

- SV_POSITION_FLAG_RELATIVE – The new position is relative to the actual one.
- SV_POSITION_FLAG_PAUSE – Goes to the position and pauses.
- SV_POSITION_FLAG_SPEEDONE – Goes to the position and sets the speed to one (1).

Returns:

If the function succeeds, it returns SV_OK. Otherwise it will return the error code SV_ERROR_<xxx>.

Note:

This define is obsolete but can be used for testing purposes (see the introduction to this chapter).

int sv_preset (sv_handle * *sv*, int *preset*)

This function selects the channels that shall be active during subsequent *sv_record()* operations.

Parameters:

sv – Handle returned from the function *sv_open()*.

preset – SV_PRESET_<xxx> defines. For a list see the function *sv_memory_frameinfo()*. Combine these values to enable the desired channels for a record.

Returns:

If the function succeeds, it returns SV_OK. Otherwise it will return the error code SV_ERROR_<xxx>.

Note:

This define is obsolete but can be used for testing purposes (see the introduction to this chapter).

int sv_record (sv_handle * *sv*, char * *memp*, int *memsize*, int * *pxsize*, int * *pysize*, int *start*, int *nframes*, int *tc*)

This function starts a record operation to the storage.

Parameters:

sv – Handle returned from the function *sv_open()*.

memp – Obsolete. Must be NULL.

memsize – Obsolete. Must be zero (0).

pxsize – Pointer to return the x-size. Can be NULL.

pysize – Pointer to return the y-size. Can be NULL.

start – Indicates the frame number (start frame) where the record will start.

nframes – Sets the number of frames to be recorded.

tc – If not zero (0), a play-out will be performed on a connected external device (e.g. VTR) at the indicated position (timecode).

Returns:

If the function succeeds, it returns SV_OK. Otherwise it will return the error code SV_ERROR_<xxx>.

Note:

This define is obsolete but can be used for testing purposes (see the introduction to this chapter).

int sv_showinput (sv_handle * *sv*, int *showinput*, int *spare*)

This function enables the live mode (EE).

Parameters:

sv – Handle returned from the function *sv_open()*.

showinput – SV_SHOWINPUT_<xxx>. See list below.

spare – Reserved for future use. It has to be set to zero (0).

Parameters for *showinput*:

- SV_SHOWINPUT_DEFAULT – The default input of the device will be selected.
- SV_SHOWINPUT_FRAMEBUFFERED – The live signal will be written to the memory first before sent to the output. This setting is the default setting.

Returns:

If the function succeeds, it returns SV_OK. Otherwise it will return the error code SV_ERROR_<xxx>.

Note:

This define is obsolete but can be used for testing purposes (see the introduction to this chapter).

See also:

The function *sv_live()*.

int sv_stop (sv_handle * *sv*)

This function stops any running record or display operation. It is a kind of an emergency break and will reinitialize any pending data transfer. Use this call if there has been a data rate overrun or any other cause for a program abort that leaves the communication between computer and video board in an invalid state.

Parameters:

sv – Handle returned from the function *sv_open()*.

Returns:

If the function succeeds, it returns SV_OK. Otherwise it will return the error code SV_ERROR_<xxx>.

Note:

This define is obsolete but can be used for testing purposes (see the introduction to this chapter).

int sv_sv2host (sv_handle * *sv*, char * *buffer*, int *buffersize*, int *xsize*, int *ysize*, int *frame*, int *nframes*, int *mode*)

This function transfers data from the video device to the system memory.

Parameters:

sv – Handle returned from the function *sv_open()*.

buffer – Pointer to the system memory area where the data shall be transferred to.

buffersize – Buffer size of the data to be transferred from the video device. Must be greater than or equal to (>=) the size of one frame.

xsize – X-size corresponding to the current video raster for the video data. For audio data it must be 2 x 48000 / fps, for example, for 25 fps it is 3840 and for 29.97 Hz it alternates between 3200/3204. For pulldown clips the amount of data is 4004 or 1001 samples per frame.

ysize – Y-size corresponding to the current video raster for the video data. This value is not evaluated for audio data.

frame – Frame number of the frame on the video device that shall be transferred. Must be a valid storage page.

nframes – Obsolete. Must be one (1).

mode – Combination of qualifiers (data type and data size) that form the current raster. For possible parameters see the function *sv_host2sv()*.

Returns:

If the function succeeds, it returns SV_OK. Otherwise it will return the error code SV_ERROR_<xxx>.

Note:

This define is obsolete but can be used for testing purposes (see the introduction to this chapter).

See also:

The function *sv_host2sv()*.

Obsolete Defines and Functions

Detailed Description

This chapter details obsolete defines and functions, i.e. defines and functions that were either substituted by newer ones or that have no useful functionality anymore.

Defines

- #define SV_OPTION_ASSIGN_LTC
- #define SV_OPTION_ASSIGN_VTR
- #define SV_OPTION_RS422A_PINOUT
- #define SV_OPTION_RS422B_PINOUT

Functions

- int sv_get_version (sv_handle *sv, sv_version *version, int id)
- int sv_memory_play (sv_handle *sv, int inpoint, int outpoint, double speed, int tc, int flags)
- int sv_memory_record (sv_handle *sv, int inpoint, int outpoint, double speed, int tc, int flags)
- char * sv_vtrerror (sv_handle *sv, int code)

Define Documentation

#define SV_OPTION_ASSIGN_LTC

This define is obsolete. Instead use the define SV_OPTION_ASSIGN_LTCA which performs the same operation but for the LTC A specifically.

#define SV_OPTION_ASSIGN_VTR

This define is obsolete. Instead use the defines SV_OPTION_RS422A and SV_OPTION_RS422B.

This define was used on Centaurus II to assign the RS-422 timecode to another jack than the default jack zero (0). To use this define the multi-channel operation mode had to be activated.

Note:

When setting any non-default RS-422 task assignments by using the defines SV_OPTION_RS422<xxx>, this define will return SV_ERROR_WRONGMODE until the default assignment is restored.

#define SV_OPTION_RS422A_PINOUT

Obsolete, use SV_OPTION_RS422A instead.

#define SV_OPTION_RS422B_PINOUT

Obsolete, use SV_OPTION_RS422B instead.

Function Documentation

int sv_get_version (sv_handle * *sv*, sv_version * *version*, int *id*)

This function is obsolete. Instead use the function *sv_version_status()*.

Parameters:

sv – Handle returned from the function *sv_open()*.

version – Not implemented.

id – Not implemented.

Returns:

Always returns `SV_ERROR_NOTIMPLEMENTED`.

int sv_memory_play (sv_handle * *sv*, int *inpoint*, int *outpoint*, double *speed*, int *tc*, int *flags*)

This function is obsolete. It was used to start a RAM-recorder display operation on the DVS video device using the specified inpoint/outpoint range on the video board. This function is similar to the function *sv_display()* with the difference that you can specify the display speed.

Parameters:

sv – Handle returned from the function *sv_open()*.

inpoint – First frame to be displayed.

outpoint – First frame not to be displayed.

speed – Display speed.

tc – If not zero (`0`), an autoedit record will be performed on a connected external device (e.g. VTR) at the indicated position (timecode).

flags – Not used. It has to be set to zero (`0`).

Returns:

If the function succeeds, it returns `SV_OK`. Otherwise it will return the error code `SV_ERROR_<xxx>`.

Note:

This function is not intended to be used together with the FIFO API.

int sv_memory_record (sv_handle * *sv*, int *inpoint*, int *outpoint*, double *speed*, int *tc*, int *flags*)

This function is obsolete. It was used to start a RAM-recorder record operation on the DVS video device. This function is similar to the function *sv_record()*.

Parameters:

sv – Handle returned from the function *sv_open()*.

inpoint – First frame to be recorded.

outpoint – First frame not to be recorded.

speed – Not used.

tc – If not zero (`0`), a play-out will be performed on a connected external device (e.g. VTR) at the indicated position (timecode).

flags – Not used. It has to be set to zero (`0`).

Returns:

If the function succeeds, it returns `SV_OK`. Otherwise it will return the error code `SV_ERROR_<xxx>`.

Note:

This function is not intended to be used together with the FIFO API.

char* sv_vtrerror (sv_handle * *sv*, int *code*)

This function is obsolete. Instead use the function *sv_geterrortext()*. It was used to return a pointer to a string describing the error code of the VTR.

Parameters:

sv – Handle returned from the function *sv_open()*.

code – Error code.

Returns:

ASCII representation (string) that describes the error code.

Info – Bit Formats

The following details information about the bit format of the various file formats that can be processed in your application via the DVS SDK. Each color component of a given pixel format corresponds to one of the below mentioned components A, B, C, and D. The following representations are in bytes always. The individual pixel formats are detailed in chapter Info – Pixel Formats.

8 bit

```
SV_MODE_NBIT_8B
AAAAAAAA BBBBBBBB CCCCCCCC ...
76543210 76543210 76543210 ...
```

10 bit (right aligned little endian)

`SV_MODE_NBIT_10B` (same as `SV_MODE_NBIT_10BRALE`)

```
AAAAAAAA BBBBBBAA CCCCBBBB 00CCCCCC ...
76543210 54321098 32109876 --987654 ...
```

10 bit DPX (left aligned big endian)

`SV_MODE_NBIT_10BDPX` (same as `SV_MODE_NBIT_10BLABE`)

```
AAAAAAAA AABBBBBB BBBBCCCC CCCCCC00 ...
98765432 10987654 32109876 543210-- ...
```

10 bit (right aligned big endian)

```
SV_MODE_NBIT_10BRABE
00CCCCCC CCCCBBBB BBBBBBAA AAAAAAAA ...
--987654 32109876 54321098 76543210 ...
```

10 bit (left aligned little endian)

```
SV_MODE_NBIT_10BLALE
CCCCCC00 BBBBCCCC AABBBBBB AAAAAAAA ...
543210-- 32109876 10987654 98765432 ...
```

10 bit DVS

`SV_MODE_NBIT_10BDVS` – Proprietary bit format, easy to handle in hardware

```
AAAAAAAA BBBBBBBB CCCCCCCC 00CCBBAA ...
98765432 98765432 98765432 --101010 ...
```

12 bit

```
SV_MODE_NBITS_12B
AAAAAAAA AAAABBBB BBBBBBBB CCCCCCCC CCCCAAAA AAAAAAAA BBBBBBBB
00000000 00000000 00000000 00000000 00001111 11111111 11111111
ba987654 3210ba98 76543210 ba987654 3210ba98 76543210 ba987654
BBBBCCCC CCCCCCCC ...
11111111 11111111 ...
3210ba98 76543210 ...
```

12 bit DPX

```
SV_MODE_NBITS_12BDPX
CCCCCCCC BBBBBBBB BBBBAAAA AAAAAAAA CCCCBBBB BBBBBBBB AAAAAAAA
00000000 00000000 00000000 00000000 11111111 11111111 11111111
```

```
76543210 ba987654 3210ba98 76543210 3210ba98 76543210 ba987654
AAAACCCC BBBBBBBB BBBBAAAA AAAAAAAA CCCCCCCC BBBBBBBB AAAAAAAA
11110000 22222222 22222222 22222222 11111111 33333333 33333333
3210ba98 ba987654 3210ba98 76543210 ba987654 76543210 ba987654
AAAACCCC CCCCCCCC BBBBAAAA AAAAAAAA CCCCCCCC CCCCBBBB AAAAAAAA
33332222 22222222 44444444 44444444 33333333 33333333 55555555
3210ba98 76543210 3210ba98 76543210 ba987654 3210ba98 ba987654
AAAACCCC CCCCCCCC BBBBBBBB AAAAAAAA CCCCCCCC CCCCBBBB BBBBBBBB
55554444 44444444 44444444 66666666 55555555 55555555 55555555
3210ba98 76543210 ba987654 76543210 ba987654 3210ba98 76543210
AAAACCCC CCCCCCCC BBBBBBBB BBBBAAAA CCCCCCCC CCCCBBBB BBBBBBBB
77776666 66666666 66666666 66666666 77777777 77777777 77777777
3210ba98 76543210 ba987654 3210ba98 ba987654 3210ba98 76543210
AAAAAAAA ...
77777777 ...
ba987654 ...
```

Info – Pixel Formats

This chapter details the pixel formats that can be processed in your application via the DVS SDK. The following representations show the color components instead of bytes. A component may have a specific bit depth and multiple components are normally grouped as shown in chapter Info – Bit Formats.

YUV422_UYVY

`SV_MODE_COLOR_YUV422_UYVY` (same as `SV_MODE_COLOR_YUV422`)

`U0Y0V0Y1 U2Y2V2Y3 ...`

This pixel format is similar to Windows UYVY.

YUV422_YUYV

`SV_MODE_COLOR_YUV422_YUYV`

`Y0U0Y1V0 Y2U2Y3V2 ...`

This pixel format is similar to Windows YUY2.

YUV422A

`SV_MODE_COLOR_YUV422A`

`U0Y0A0V0 Y1A1U2Y2 A2V2Y3A3 ...`

YUV444

`SV_MODE_COLOR_YUV444`

`U0Y0V0U1 Y1V1U2Y2 V2U3Y3V3 ...`

YUV444A

`SV_MODE_COLOR_YUV444A`

`U0Y0V0A0 U1Y1V1A1 U2Y2V2A2 ...`

RGB

`SV_MODE_COLOR_RGB_RGB`

`R0G0B0 R1G1B1 R2G2B2 ...`

BGR

`SV_MODE_COLOR_RGB_BGR` (same as `SV_MODE_COLOR_RGB`)

`B0G0R0 B1G1R1 B2G2R2 ...`

RGBA

`SV_MODE_COLOR_RGBA`

`R0G0B0A0 R1G1B1A1 R2G2B2A2 ...`

BGRA

`SV_MODE_COLOR_BGRA`

`B0G0R0A0 B1G1R1A1 B2G2R2A2 ...`

ARGB

`SV_MODE_COLOR_ARGB`

`A0R0G0B0 A1R1G1B1 A2R2G2B2 ...`

ABGR

`SV_MODE_COLOR_ABGR`

`A0B0G0R0 A1B1G1R1 A2B2G2R2 ...`

Info – Audio Formats

This chapter describes the audio configurations and formats that can be used with the DVS SDK.

Number of Audio Channels:

- `SV_MODE_AUDIO_NOAUDIO` – Configures none (zero) audio channels.
- `SV_MODE_AUDIO_8CHANNEL` – Configures eight stereo audio channels.

Audio Sample Size:

- `SV_MODE_AUDIOBITS_32` – Configures a sample size of 32 bit.

Info – Storage Formats

This chapter describes the video storage formats that can be used in the video buffers of the DVS video boards.

Field Storage:

The field storage mode is automatically used if the define `SV_MODE_STORAGE_FRAME` is not set. The frame/fields are stored line-wise in two separate buffers.

Frame Storage:

- `SV_MODE_STORAGE_FRAME` – In interlaced or segmented frames rasters both fields are stored in one buffer. This will also be the case when in progressive rasters, however, then the complete frame is stored in one buffer.

Bottom to Top:

- `SV_MODE_STORAGE_BOTTOM2TOP` – The lines of a video buffer are stored in swapped order, i.e. from bottom to top.

Info – Error Codes

This chapter provides a list of error codes that can be returned by SV functions together with a short description. However, return values of *sv_<xxx>()* functions might differ from the descriptions stated in the following. Instead of checking for a particular error code, we recommend to check a return value for being unequal to the constant `SV_OK` which is defined as zero (`0`). To convert the error code into a string you can use the function *sv_geterrortext()*.

Error Meaning:

- `SV_OK` (0) – Successful operation.
- `SV_ACTIVE` (-1) – Operation still active. Mainly used for overlapped DMAs.
- `SV_ERROR_ALREADY_RUNNING` (126) – You tried to start an operation that was already started.
- `SV_ERROR_ALREADY_OPENED` (155) – The resource is already opened.
- `SV_ERROR_ALREADY_CLOSED` (156) – The resource is already closed.
- `SV_ERROR_ANCONSWITCHINGLINE` (157) – You cannot put ANC data on the switching line.
- `SV_ERROR_ASYNCNOTFOUND` (143) – This asynchronous call is no longer available.
- `SV_ERROR_AUDIO_SEGMENT` (57) – Obsolete. The specified audio segment does not exist.
- `SV_ERROR_BUFFER_NOTALIGNED` (106) – The buffer does not provide the needed alignment.
- `SV_ERROR_BUFFER_NULL` (107) – The buffer does not point to anything.
- `SV_ERROR_BUFFER_TOLARGE` (108) – A buffer is too large.
- `SV_ERROR_BUFFERSIZE` (4) – The supplied buffer is too small.
- `SV_ERROR_CANCELED` (89) – The operation was cancelled.
- `SV_ERROR_CHECKWORD` (121) – The check word is wrong.
- `SV_ERROR_CLIP_BLOCKED` (137) – Obsolete. The selected clip is blocked because it is currently in use.
- `SV_ERROR_CLIP_INVALID` (138) – Obsolete. The selected clip is not recognized as valid for the DVS video device.
- `SV_ERROR_CLIP_NAMEEXISTS` (43) – Obsolete. The clip name already exists.
- `SV_ERROR_CLIP_NOENTRY` (44) – Obsolete. The clip directory is full.
- `SV_ERROR_CLIP_NOTCREATED` (164) – Obsolete. The clip cannot be created (possibly because of an unsupported format).
- `SV_ERROR_CLIP_NOTFOUND` (42) – Obsolete. The clip could not be found.
- `SV_ERROR_CLIP_OVERLAP` (45) – Obsolete. The clip would overlap with a subdirectory.
- `SV_ERROR_CLIP_PROTECTED` (170) – Obsolete. This clip is protected and cannot be deleted.
- `SV_ERROR_CLIP_TOOBIG` (165) – Obsolete. There is not enough free space to create this clip.
- `SV_ERROR_CLIPDIR_NAMEEXISTS` (47) – Obsolete. The directory name already exists.
- `SV_ERROR_CLIPDIR_NOENTRY` (48) – Obsolete. The directory field is full.
- `SV_ERROR_CLIPDIR_NOTEMPTY` (60) – Obsolete. The directory is not empty.
- `SV_ERROR_CLIPDIR_NOTFOUND` (46) – Obsolete. The directory could not be found.
- `SV_ERROR_CLIPDIR_OVERLAP` (49) – Obsolete. The directory would overlap with a clip or subdirectory.
- `SV_ERROR_CLIPDIR_NOTSELECT` (122) – Obsolete. A directory cannot be selected in this file system.
- `SV_ERROR_DATALOST` (9) – Obsolete. Data was lost during a transfer.

- SV_ERROR_DATARATE (150) – The data rate for this raster is too high.
- SV_ERROR_DEVICEINUSE (161) – The device is in use.
- SV_ERROR_DEVICENOTFOUND (162) – The device could not be found.
- SV_ERROR_DIRCREATE (178) – Obsolete. Could not create the directory.
- SV_ERROR_DIRECT_BUFFER_ALREADY_BOUND (230) – The buffer index used in the Direct API instance is already bound to an object.
- SV_ERROR_DIRECT_BUFFER_INDEX (229) – The buffer index used in the Direct API instance is not available.
- SV_ERROR_DIRECT_BUFFER_NOT_BOUND (231) – The buffer index used in the Direct API does not provide a binding.
- SV_ERROR_DIRECT_BUFFER_SYNC (232) – Either this Direct API instance cannot be synchronized or something is missing for a synchronization.
- SV_ERROR_DIRECT_CLOSED (237) – The Direct API instance is already closed.
- SV_ERROR_DIRECT_INVALID (228) – The handle to the Direct API instance is invalid.
- SV_ERROR_DIRECT_NULL (227) – The handle to the Direct API instance is NULL.
- SV_ERROR_DIRECT_OPENED (236) – The Direct API instance is already opened.
- SV_ERROR_DIRECT_STARTED (238) – The Direct API instance is already started.
- SV_ERROR_DIRECT_STOPPED (239) – The Direct API instance is stopped.
- SV_ERROR_DIRECT_STORAGEMODE_INVALID (240) – Wrong storage mode has been selected for the Direct API instance.
- SV_ERROR_DIRECT_TEXTURE_INVALID (235) – Wrong or no texture available for the Direct API instance.
- SV_ERROR_DIRECT_TIMECODE_ALREADY_BOUND (233) – A timecode binding is already bound to the Direct API instance.
- SV_ERROR_DIRECT_TIMECODE_NOT_BOUND (234) – No timecode binding has been bound to the Direct API instance.
- SV_ERROR_DISABLED (61) – The called function is disabled.
- SV_ERROR_DISKFORMAT (34) – Obsolete. Inappropriate disk formatting for selected video mode.
- SV_ERROR_DISPLAYONLY (93) – With the specified video raster only display operations are possible.
- SV_ERROR_DRIVER_BADPCIMAPPING (172) – The PCI mapping has an overlap.
- SV_ERROR_DRIVER_CONNECTIRQ (77) – The driver could not connect to an IRQ.
- SV_ERROR_DRIVER_HWCHECK (215) – A hardware malfunction has been detected. Please contact the DVS service department.
- SV_ERROR_DRIVER_HWPATH (92) – The driver hardware file(s) (*.pld) could not be found (wrong path or files missing).
- SV_ERROR_DRIVER_MALLOC (80) – Driver could not allocate critical memory.
- SV_ERROR_DRIVER_MAPIOSPACE (78) – Driver could not map on-board memory into kernel memory.
- SV_ERROR_DRIVER_MAPPEDSIZE (209) – Wrong internal state in driver detected. Please contact the support for DVS video board products.
- SV_ERROR_DRIVER_MEMORY (168) – Not all memory modules found.
- SV_ERROR_DRIVER_MEMORYINIT (216) – A hardware malfunction has been detected. Please contact the DVS service department.
- SV_ERROR_DRIVER_MEMORYMATCH (169) – Mounted memory modules do not match.
- SV_ERROR_DRIVER_MISMATCH (147) – Driver and library version mismatch detected.
- SV_ERROR_DRIVER_RESOURCES (79) – The driver did not get resources from the kernel.
- SV_ERROR_DTM_TIMEOUT (40) – Connection timeout between computer and video board.

- SV_ERROR_EPLD_CHIP (73) – An EPLD has the wrong chip ID.
- SV_ERROR_EPLD_MAGIC (71) – An EPLD has the wrong magic number.
- SV_ERROR_EPLD_NOTFOUND (123) – During driver loading the driver could not find the hardware files (*.pld).
- SV_ERROR_EPLD_PRODUCT (72) – An EPLD is from the wrong device.
- SV_ERROR_EPLD_VERSION (74) – An EPLD has the wrong version.
- SV_ERROR_FIFO_PUTBUFFER (85) – The FIFO getbuffer/putbuffer pair was called incorrectly.
- SV_ERROR_FIFO_STOPPED (190) – This command cannot be done while the FIFO is stopped.
- SV_ERROR_FIFO_TIMEOUT (84) – The FIFO timed out.
- SV_ERROR_FIFOCLOSED (154) – This command cannot be done while the FIFO is closed.
- SV_ERROR_FIFOOPENED (152) – This command cannot be done while the FIFO is opened.
- SV_ERROR_FILECLOSE (14) – Closing of file failed.
- SV_ERROR_FILECREATE (8) – Creation of file failed.
- SV_ERROR_FILEDIRECT (15) – Direct file access could not be set.
- SV_ERROR_FILEEXISTS (177) – The file already exists.
- SV_ERROR_FILEFORMAT (139) – The file format is not valid.
- SV_ERROR_FILEOPEN (7) – The opening of the file failed.
- SV_ERROR_FILEREAD (12) – The reading from a file failed.
- SV_ERROR_FILESEEK (16) – The seeking in a file failed.
- SV_ERROR_FILETRUNCATE (17) – Truncating of file failed.
- SV_ERROR_FILEWRITE (13) – The writing to a file failed.
- SV_ERROR_FIRMWARE (30) – Wrong firmware version detected that does not support the request.
- SV_ERROR_FLASH_ERASETIMEOUT (69) – During erasure of the flash chip a timeout occurred.
- SV_ERROR_FLASH_ERASEVERIFY (118) – Verifying of the flash erase failed.
- SV_ERROR_FLASH_VERIFY (70) – Verifying the flash chip after programming failed.
- SV_ERROR_FLASH_WRITE (163) – Flash write failed.
- SV_ERROR_FRAME_NOACCESS (51) – Frame is not accessible.
- SV_ERROR_HARDWARELOAD (59) – Hardware failed to load.
- SV_ERROR_HIGH_MEMORY (184) – The operation cannot be performed in high memory.
- SV_ERROR_INF_MISMATCH (194) – The driver file (*.inf) does not match the driver binary.
- SV_ERROR_INPUT_AUDIO_FREQUENCY (119) – Wrong audio frequency, i.e. an unexpected audio frequency is available at the input.
- SV_ERROR_INPUT_AUDIO_NOAESEBU (116) – Wrong audio format at the input, the expected audio signal should be AES/EBU audio.
- SV_ERROR_INPUT_AUDIO_NOAIV (120) – Wrong audio format at the input, the expected audio signal should be embedded audio (AIV).
- SV_ERROR_INPUT_KEY_NOSIGNAL (114) – No key signal detected at the input.
- SV_ERROR_INPUT_KEY_RASTER (115) – Key signal at the input has an unexpected raster.
- SV_ERROR_INPUT_VIDEO_DETECTING (186) – Video input detection not yet ready.
- SV_ERROR_INPUT_VIDEO_NOSIGNAL (112) – No video signal detected at the input.
- SV_ERROR_INPUT_VIDEO_RASTER (113) – Video signal at the input has an unexpected raster, i.e. the input raster does not match the device setup.
- SV_ERROR_INTERNALMAGIC (166) – An internal check of the library failed.

- SV_ERROR_IOCHANNEL_INVALID (208) – Wrong jack for the specified I/O channel selected.
- SV_ERROR_IOCTL_FAILED (83) – An ioctl operation failed.
- SV_ERROR_IOMODE (64) – Invalid I/O mode selected.
- SV_ERROR_JACK_ASSIGNMENT (197) – The channel is already assigned to another jack.
- SV_ERROR_JACK_INVALID (196) – Invalid jack name or index for this operation.
- SV_ERROR_JACK_NOBYPASS (201) – The jack does not have a bypass jack assigned.
- SV_ERROR_JACK_NOTASSIGNED (198) – No channels are assigned to this jack.
- SV_ERROR_JPEG2K_CODESTREAM (199) – Error in the JPEG2000 codestream.
- SV_ERROR_JPEG2K_DECODE (225) – Wrong internal chip state detected. Please contact the support for DVS video board products.
- SV_ERROR_LICENCE_12BITS (175) – You tried to use a 12-bit I/O mode without license.
- SV_ERROR_LICENCE_AUDIO (129) – You tried to use more audio channels than licensed.
- SV_ERROR_LICENCE_CUSTOMRASTER (185) – You tried to use a customized video raster without license.
- SV_ERROR_LICENCE_DUALLINK (134) – You tried to use dual link without license.
- SV_ERROR_LICENCE_DVI16 (207) – You tried to use DVI 16 bit without license.
- SV_ERROR_LICENCE_DVIINPUT (193) – You tried to use the DVI input without license.
- SV_ERROR_LICENCE_EUREKA (192) – You tried to use an Eureka raster without license.
- SV_ERROR_LICENCE_EXPIRED (210) – Your license has expired.
- SV_ERROR_LICENCE_FILM2K (136) – You tried to use a FILM2K raster without license.
- SV_ERROR_LICENCE_FILM2KPLUS (145) – You tried to use a FILM2Kplus feature without license.
- SV_ERROR_LICENCE_FILM4K (173) – You tried to use a FILM4K raster without license.
- SV_ERROR_LICENCE_HD360 (141) – You tried to use HD360 without license.
- SV_ERROR_LICENCE_HDTV (180) – You tried to use an HDTV raster without license.
- SV_ERROR_LICENCE_HIRES (181) – You tried to use a high-resolution raster without license.
- SV_ERROR_LICENCE_HSDL (144) – You tried to use an HSDL raster without license.
- SV_ERROR_LICENCE_HSDLRT (174) – Obsolete. You tried to use an HSDL real-time raster without license.
- SV_ERROR_LICENCE_JPEG2000CODEC4K (220) – You tried to use JPEG2000 4K without license.
- SV_ERROR_LICENCE_JPEG2000RAW (214) – You tried to use JPEG2000 RAW decompression without license.
- SV_ERROR_LICENCE_KEYCHANNEL (132) – You tried to use a key channel without license.
- SV_ERROR_LICENCE_LINKENCRYPT (211) – You tried to use link encryption without license.
- SV_ERROR_LICENCE_MIXER (133) – You tried to use the mixer without license.
- SV_ERROR_LICENCE_MULTICHANNEL (217) – You tried to use multi-channel without license.
- SV_ERROR_LICENCE_MULTIDEVICE (182) – You tried to use the multi-device feature without license.
- SV_ERROR_LICENCE_PHDTV (187) – You tried to use a PHDTV raster without license.
- SV_ERROR_LICENCE_RENDER (213) – You tried to use the Render API without license.
- SV_ERROR_LICENCE_RGB (131) – You tried to use RGB without license.
- SV_ERROR_LICENCE_SDTV (135) – You tried to use an SDTV raster without license.
- SV_ERROR_LICENCE_SDTVFF (191) – You tried to use an SDTV-FF raster without license.

- SV_ERROR_LICENCE_SLOWPAL (188) – You tried to use SLOW PAL without license.
- SV_ERROR_LICENCE_STEREO (218) – You tried to use stereo without license.
- SV_ERROR_LICENCE_STREAMER (130) – You tried to use the streamer mode without license.
- SV_ERROR_LICENCE_WATERMARK (221) – You tried to use watermarking without license.
- SV_ERROR_MALLOC (2) – Memory allocation failed.
- SV_ERROR_MALLOC_FRAGMENTED (183) – Memory allocation failed due to memory fragmentation.
- SV_ERROR_MASTER (22) – Master control failed.
- SV_ERROR_MEM_BUFFERSIZE (24) – The buffer size is too small.
- SV_ERROR_MEM_NULL (23) – The buffer is NULL.
- SV_ERROR_MISSING_SLAVE_TASK (222) – RS-422 configuration error: No slave task has been assigned to the current I/O channel.
- SV_ERROR_MMAPFAILED (87) – Memory mapping function failed.
- SV_ERROR_MULTICHANNEL_RASTER (212) – Wrong raster has been configured for an I/O channel in multi-channel operation mode.
- SV_ERROR_NOCARRIER (27) – No valid input signal detected.
- SV_ERROR_NODATA (200) – Internal. No data provided.
- SV_ERROR_NODRAM (29) – Obsolete. DRAM option is not available.
- SV_ERROR_NOGENLOCK (28) – Genlock option not available.
- SV_ERROR_NOHSWTRANSFER (111) – Obsolete. Host-to-software transfer has been disabled.
- SV_ERROR_NOINPUTANDOUTPUT (153) – In this mode you cannot do both input and output.
- SV_ERROR_NOLICENCE (50) – License code for this operation not available.
- SV_ERROR_NOTASYNCCALL (142) – Function cannot be called asynchronously.
- SV_ERROR_NOTAVAILABLE (149) – A value is not in the input stream.
- SV_ERROR_NOTDEBUGDRIVER (76) – The desired operation is supported by the debug driver only.
- SV_ERROR_NOTEXTSYNC (5) – The DVS video device is not in an external sync mode.
- SV_ERROR_NOTFORDDR (159) – Obsolete. The function is not supported by the DDR.
- SV_ERROR_NOTFRAMESTORAGE (109) – This can only be done in frame storage mode.
- SV_ERROR_NOTIMPLEMENTED (3) – The called function is not supported by the current operating system.
- SV_ERROR_NOTREADY (75) – The operation is not ready.
- SV_ERROR_NOTRUNNING (110) – A polled operation is no longer active.
- SV_ERROR_NOTSUPPORTED (41) – A feature that is not supported has been called.
- SV_ERROR_OBSOLETE (146) – An obsolete function has been called.
- SV_ERROR_OPENTYPE (167) – You have not opened this resource.
- SV_ERROR_PARAMETER (1) – A parameter is wrong.
- SV_ERROR_PARAMETER_NEGATIVE (124) – A negative parameter is not valid.
- SV_ERROR_PARAMETER_TOLARGE (125) – A parameter is too large.
- SV_ERROR_PARTITION_INVALID (56) – Obsolete. Invalid partition number.
- SV_ERROR_PARTITION_NOENTRY (52) – Obsolete. The partition table is full.
- SV_ERROR_PARTITION_NOSPACE (53) – Obsolete. Not enough free space available.
- SV_ERROR_PARTITION_NOTFOUND (55) – Obsolete. Partition not found.

- SV_ERROR_PARTITION_NOTLAST (54) – Obsolete. The desired operation is not possible because the selected partition is not the last one.
- SV_ERROR_POLL_TASK_ACTIVE (58) – Obsolete. Poll task is active.
- SV_ERROR_PROGRAM (35) – Unrecoverable internal program error (program is in an illegal internal state).
- SV_ERROR_QUANTLOSS (31) – Obsolete. AC quant values were lost during compression.
- SV_ERROR_RECORD (32) – The record operation failed.
- SV_ERROR_RESOURCENOTAVAIBLE (223) – SDI link mapping is wrong: The configured resource is not available in this mode.
- SV_ERROR_SAMPLINGFREQ (86) – Illegal sampling frequency specified.
- SV_ERROR_SCSI (20) – Obsolete. SCSI transfer error between computer and video board.
- SV_ERROR_SCSIDEVICE (37) – Obsolete. SCSI device not found.
- SV_ERROR_SCSIREAD (39) – Obsolete. Error during SCSI read.
- SV_ERROR_SCSIWRITE (38) – Obsolete. Error during SCSI write.
- SV_ERROR_SERIALNUMBER (176) – Serial number is missing.
- SV_ERROR_SLAVE (33) – Obsolete. The DVS video device is in remote control mode.
- SV_ERROR_SLEEPING (219) – The driver is still in sleep mode. You have to initialize (open) the DVS video board again.
- SV_ERROR_SVJ_FRAMENR (18) – Obsolete. The desired frame number is not in the file.
- SV_ERROR_SVJ_NULL (19) – Obsolete. The *sv_handle* pointer is invalid.
- SV_ERROR_SVMAGIC (11) – The *sv_handle* pointer's magic number is invalid.
- SV_ERROR_SVNULL (10) – The *sv_handle* pointer is NULL.
- SV_ERROR_SVOPENSTRING (160) – There is a syntax error in the *sv_open()* string.
- SV_ERROR_SYNC_CALCULATION (67) – The calculation of the output sync signal failed.
- SV_ERROR_SYNC_MISSING (195) – The sync signal is either bad or missing.
- SV_ERROR_SYNC_OUTPUT (68) – The specified sync output signal is not supported.
- SV_ERROR_SYNCDELAY (171) – Invalid sync H-/V-delay.
- SV_ERROR_SYNCMODE (63) – Invalid sync mode selected.
- SV_ERROR_TIMECODE (21) – Timecode format invalid. The valid format is `hh:mm:ss:ff`.
- SV_ERROR_TIMELINE (36) – Invalid timeline segment was specified.
- SV_ERROR_TIMEOUT (88) – The operation timed out.
- SV_ERROR_TOLERANCE (148) – The tolerance value has been exceeded.
- SV_ERROR_TOMANYAUDIOCHANNELS (128) – You tried to set too many audio channels.
- SV_ERROR_TRANSFER (26) – The transfer failed.
- SV_ERROR_TRANSFER_NOAUDIO (117) – Obsolete. There is no audio configured on the DVS video device.
- SV_ERROR_UNKNOWNFLASH (90) – The flash chip is not supported by the software.
- SV_ERROR_USERNOTALLOWED (179) – Obsolete. No privileges to interact with the DVS video device.
- SV_ERROR_VERSION (202) – API and driver version mismatch detected.
- SV_ERROR_VIDEO_RASTER_FILE (66) – During driver loading the driver could not find the raster definition files (*.ref).
- SV_ERROR_VIDEO_RASTER_TABLE (65) – During driver loading the driver could not initialize the video raster table.
- SV_ERROR_VIDEOMODE (6) – Unsupported video mode was specified.
- SV_ERROR_VIDEOPAGE (25) – Supplied frame number is out of range (video page does not exist).

- SV_ERROR_VSYNCFUTURE (82) – An operation was issued too early before it could start.
- SV_ERROR_VSYNCPASSED (81) – An operation was issued for a vertical sync that has already passed.
- SV_ERROR_VTR_EDIT (105) – Master control during VTR edit failed.
- SV_ERROR_VTR_GOTOERROR (98) – Operation on VTR did not complete.
- SV_ERROR_VTR_LOCAL (95) – The VTR is in local mode. Check the VTR and set it to remote control.
- SV_ERROR_VTR_NAK (97) – Received 'Not acknowledged' from VTR.
- SV_ERROR_VTR_NOACK (100) – Acknowledge from VTR is missing.
- SV_ERROR_VTR_NOSTATUS (99) – Status reply from VTR is missing.
- SV_ERROR_VTR_NOTIMECODE (101) – The VTR's timecode reply is wrong.
- SV_ERROR_VTR_NOTIMECODECHANGE (102) – Timecode did not change during VTR edit.
- SV_ERROR_VTR_OFFLINE (94) – There is no VTR connected.
- SV_ERROR_VTR_SERIAL (96) – Error from the serial driver.
- SV_ERROR_VTR_TCORDER (103) – Timecode order during VTR edit is wrong.
- SV_ERROR_VTR_TICKORDER (104) – Tick order during VTR edit is wrong.
- SV_ERROR_VTR_UNDEFINEDCOMMAND (140) – The VTR returns an undefined command.
- SV_ERROR_WRONG_BITDEPTH (158) – The selected bit depth or operation for this bit depth is not supported.
- SV_ERROR_WRONG_COLORMODE (91) – The selected color mode is not supported.
- SV_ERROR_WRONG_HARDWARE (62) – The hardware does not support the desired operation.
- SV_ERROR_WRONG_OS (127) – This function is not supported on the current operating system.
- SV_ERROR_WRONG_PCISPEED (189) – The DVS video device is running at a PCI speed that is not supported.
- SV_ERROR_WRONGMODE (151) – Currently this command is not possible.
- SV_ERROR_WRONGMODE_QUADMODE_DUALLINK_1GB5 (224) – Quad-mode raster in combination with dual-link I/O mode and 1.5 Gbit/s I/O speed not possible on this hardware configuration.

Example Projects Overview

The DVS SDK comes with some example projects that can be analyzed to understand the SDK programming. This chapter describes the function and purposes of these example projects. The projects are stored in the directory *sdk4.<x>.<y>.<z>/development/examples*. To actually run the example projects, you have to compile them first or run the pre-compiled programs that can be found in the *sdk4.<x>.<y>.<z>/{win32, linux_x86, ...}/bin* directory.

Most of the examples mentioned in the following demonstrate the usage of the FIFO API (see chapter API – FIFO API). They use functions from the video C library (*dvs_clib.h*) and the FIFO API (*dvs_fifo.h*).

Note:

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

The directory *common* contains materials (e.g. header files) that are used by several of the example programs.

bmpoutput

This program works under Windows only as it uses the Win32 API and GUI components. It shows the Windows desktop on the video output.

cmodetst

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.

directloop

This example program uses the Direct API (see API – Direct API) in its native operation mode to pass video data from input to output. Inbetween a copy in memory is executed, which is used in this example as a placeholder for other processing tasks that can be performed with the data in the buffer.

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 and how to reset FIFOs properly in case of signal loss.

dmaspeed

Determines the current host transfer speed.

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. As a very first approach this example program is in more detail described in chapter Example – dpxio.

dpxrender

This example demonstrates the usage of the Render API (see the reference guide to the DVS Render API Extension). You can use it to process image files (in this case DPX files) in hardware with different video processing operations provided by the Render API.

jackloop

This example program is similar to the `dmaloop` example program but shows how to use independent I/Os. It will use the input/output raster set for the board and give out, for example, an SD image in a 2K frame until aborted. The smaller input image is shown with its original size in the larger output image and will alter its position with each frame displayed.

logo

This example shows how to use an input and output FIFO simultaneously. It records the incoming video, inserts a logo into each frame and outputs the material again.

overlay

The `overlay` example program shows how to use the mixer functionality of the FIFO API. It will mix an input stream (bypass) with a black image performing a wipe from bottom to top. Additionally, it applies an extended data handling of ANC data.

preview

This program works under Windows only as it uses the Win32 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, then converted into RGB and displayed.

proxy

This program works under Windows only as it uses the Win32 API and GUI components. It shows the usage of the Proxy Capture API (see chapter API – Proxy Capture) and displays the current output signal in a downscaled format inside a desktop overlay.

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.

stereo_player

This example demonstrates the stereoscopic capabilities of the DVS SDK and the video boards. It uses the FIFO API to display two independent video streams. Additionally, it can be used with the Render API to display the images e.g. side by side (left/right or top/bottom) or interlaced.

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 information about how to use this program please refer to the user guide of the DVS SDK.

Example – dpxio

Detailed Description

This chapter shows the use of the FIFO API (see chapter API – FIFO API) to display and record video from/to DPX files and audio from/to an AIFF file.

Functions

- int dpxio_exec (dpxio_handle *dpxio, int framecount)
- void dpxio_set_timecodes (dpxio_handle *dpxio, int frame, int tick, int fieldcount)
- void dpxio_tracelog (dpxio_handle *dpxio, int start, int count)
- int main (int argc, char **argv)

Function Documentation

int dpxio_exec (dpxio_handle * *dpxio*, int *framecount*)

This function is the execution function of the example program. All FIFO calls are performed in this function.

Parameters:

dpxio – Application handle.

framecount – Number of frames to record/display.

Returns:

The number of frames that were actually recorded/displayed.

First, we want to check the video raster settings that are currently initialized to be able to set the size for the captured frame. For this we start by querying the hardware with the function *sv_storage_status()*.

In all error logging functions we use the function *sv_geterrortext()* to get a readable form of the SV error codes.

```
    res = sv_storage_status(dpxio->sv, dpxio->binput ? 1: 0, NULL, &storage,
sizeof(storage), SV_STORAGEINFO_COOKIEISJACK);
    if(res != SV_OK) {
      printf("ERROR: sv_storage_status() failed = %d '%s'\n", res,
sv_geterrortext(res));
      running = FALSE;
    }
```

For the input we use the size returned by the function *sv_storage_status()*, for the output the size of the read DPX frames will be used. We also set the color mode of the DPX frames to the proper format.

Pulldown

The pulldown flag is needed in the parameter *flagbase* for pulldown removal on input.

The function *sv_fifo_init()* is then called to initialize the input or output FIFO. For the input FIFO we may have set the FIFO flagbase as the record is done before the FIFO input get-/putbuffer pair.

```
      res = sv_fifo_init(dpxio->sv, &pfifo, dpxio->binput, TRUE, TRUE, flagbase, 0);
      if(res != SV_OK)  {
        printf("ERROR: sv_fifo_init(sv) failed = %d '%s'\n", res,
sv_geterrortext(res));
        running = FALSE;
      }
```

Memory Allocation

As a next step we allocate DMA buffers because it is good practice to page-align the buffers. This is not really needed, but due to the large sizes of video there will be no scatter/gather block for the DMA to start the transfer. All DMA buffers must at least be at aligned to the size returned from the driver for the board's minimum DMA alignment. You can query the minimum alignment required for your DVS video board with the define SV_QUERY_DMAALIGNMENT.

VTR Control

Before a VTR transfer is performed you should set the edit settings to appropriate values:

```
      res = sv_vtrmaster(dpxio->sv, SV_MASTER_EDITFIELD_START, 1);
      if((res != SV_OK) && (res != SV_ERROR_VTR_OFFLINE)) {
        printf("ERROR: Setting edit field start failed = %d '%s'\n", res,
sv_geterrortext(res));
        running = FALSE;
      }
      res = sv_vtrmaster(dpxio->sv, SV_MASTER_EDITFIELD_END,   1);
      if((res != SV_OK) && (res != SV_ERROR_VTR_OFFLINE)) {
        printf("ERROR: Setting edit field end failed = %d '%s'\n", res,
sv_geterrortext(res));
        running = FALSE;
      }
```

VTR control for the FIFO API is performed via the *sv_vtrcontrol()* function. The first call with a set *init* parameter should be done with the VTR timecode and the number of frames that should be edited. This call will instruct the driver to preroll the VTR and commence a play once the VTR has reached the preroll point. Slightly before the inpoint the VTR will be up to speed (if the preroll time was enough) and the function will return the tick (i.e. the parameter *when*) when the edit should commence. This can be fed into the structure *sv_fifo_bufferinfo* of the *sv_fifo_getbuffer()* function to start the FIFO with a timed operation which will start the input or output at the correct position.

These two functions are mostly needed for edits on the VTR. An edit from the VTR can also be implemented with these functions, or by just scanning the appropriate fields in the FIFO *pbuffer* structure during a permanent capture.

```
      res = sv_vtrcontrol(dpxio->sv, dpxio->binput, TRUE, dpxio->vtr.tc,
dpxio->vtr.nframes, &when, NULL, 0);
      while((res == SV_OK) && (when == 0)) {
        res = sv_vtrcontrol(dpxio->sv, dpxio->binput, FALSE, 0, 0, &when, &timecode,
0);
        sv_usleep(dpxio->sv, 50000);
      }
```

The function *sv_fifo_start()* starts the output of the FIFO, since for an output it is a good idea to prefill the FIFO with a couple of frames.

When a VTR control is performed together with slow disks, the prebuffering might take too long. In this case the FIFO may not be started before reaching *when* which will result eventually in a drop of all frames.

```
          res = sv_fifo_startex(dpxio->sv, pfifo, &tick, &clock_high, &clock_low,
NULL);
          if(res != SV_OK)  {
          printf("ERROR: sv_fifo_start(sv) failed = %d '%s'\n", res,
sv_geterrortext(res));
          running = FALSE;
          continue;
          }
```

The functions *sv_fifo_getbuffer()* and *sv_fifo_putbuffer()* are the main work horse of the FIFO API. The function *sv_fifo_getbuffer()* returns a buffer structure that can be filled by the user and afterwards returned to the driver using the *sv_fifo_putbuffer()* function. In the function *sv_fifo_putbuffer()* the DMA to transfer the frame to or from the hardware will be done.

```
      res = sv_fifo_getbuffer(dpxio->sv, pfifo, &pbuffer, pbufferinfo, fifoflags);
      if(res != SV_OK)  {
        printf("ERROR: sv_fifo_getbuffer(sv) failed = %d '%s'\n", res,
sv_geterrortext(res));
        running = FALSE;
        continue;
      }
```

For an output of the image the code to set the *pbuffer* values must be ready for transfer at this point. For an input the buffer where the image should be stored must be given into the API.

```
      res = sv_fifo_putbuffer(dpxio->sv, pfifo, pbuffer, &putbufferinfo);
      if(res != SV_OK) {
        printf("ERROR: sv_fifo_putbuffer(sv) failed = %d '%s'\n", res,
sv_geterrortext(res));
        running = FALSE;
      }
```

For an input the writing of the recorded image should be done here.

If you want to perform pulldown in conjunction with RP215, the `dpxio` example shows how to use the *sv_fifo_anc()* function to achieve this. In case pulldown is not needed but RP215 nonetheless, it shows how to use the *sv_fifo_ancdata()* function.

Setting the Storage Format

The format of the output FIFO can be dynamically changed as long as the FIFO memory size allows this. For all DVS video boards the FIFO can be changed in format from one vertical sync to the next without any other reinitialization. One example when this will be useful is the play-out of file system files that may change in format. Please note that this is not supported by the `dpxio` example program. To see an example for this it is suggested to take a look at the `cmodetst` program.

Some fields in the *sv_fifo_buffer* structure need to be set and the flag `SV_FIFO_FLAG_STORAGEMODE` has to be set in the *sv_fifo_getbuffer()* call. In the *sv_fifo_putbuffer()* call the required storage parameters have to be set.

```
          struct {
            ...
            struct {
              int storagemode;      // Image data storage mode.
              int xsize;            // Image data x-size.
```

```
              int ysize;            // Image data y-size.
              int xoffset;          // Image data x-offset from center.
              int yoffset;          // Image data y-offset from center.
              int dataoffset;       // Offset to the first pixel in the buffer.
              int lineoffset;       // Offset from line to line or zero (0) for
default.
            } storage;
            ...
          } sv_fifo_buffer;
```

The *sv_fifo_status()* function returns the number of total buffers, free buffers and whether any frames were dropped.

Wait

If you want to wait until all frames have been transmitted for an output FIFO call the function *sv_fifo_wait()*.

Closing

After finishing using the FIFO it should be closed. The FIFO is closed and freed for other usages with the function *sv_fifo_free()*. Once this function is called, the *pfifo* handle should be discarded and not used anymore.

void dpxio_set_timecodes (dpxio_handle * *dpxio*, int *frame*, int *tick*, int *fieldcount*)

This function is used internally by the example program and sets explicit timecode values for a tick.

Parameters:

dpxio – Application handle.

frame – Frame value to be used for timecode.

tick – Tick at which the specific timecode should appear.

fieldcount – Number of fields for which timecodes should be set.

void dpxio_tracelog (dpxio_handle * *dpxio*, int *start*, int *count*)

This function is used internally by the example program and dumps timing measurement results.

Parameters:

dpxio – Application handle.

start – First frame to dump.

count – Number of frames to dump.

int main (int *argc*, char ** *argv*)

The main function of the application.

Parameters:

argc – Argument count.

argv – Argument vector.

Returns:

Return code.

DVS SDK Data Structure Documentation

sv_direct_bufferinfo Struct Reference

Detailed Description

The following describes the structure *sv_direct_bufferinfo* which is used by the functions *sv_direct_display()* and *sv_direct_record()*.

```
typedef struct {
  int size;                 // Size of the structure (needs to be filled by the
                            // caller).
  int when;                 // Current tick.
  int clock_high;           // Current clock time (upper 32 bits).
  int clock_low;            // Current clock time (lower 32 bits).
  struct {
    int clock_distance;     // Distance from starting the DMA transfer (between video
                            // board and system memory) until the following vsync. This
                            // value can be seen as an indication if the respective
                            // sv_direct_display() or sv_direct_record() function has
                            // been called too late.
    int clock_go_high;      // Clock time of the starting of the DMA (upper 32 bits).
    int clock_go_low;       // Clock time of the starting of the DMA (lower 32 bits).
    int clock_ready_high;   // Clock time of the finishing of the DMA (upper 32 bits).
    int clock_ready_low;    // Clock time of the finishing of the DMA (upper 32 bits).
    int pad[8];             // Reserved for future use.
  } dma;
  int pad[16];              // Reserved for future use.
} sv_direct_bufferinfo;
```

sv_direct_info Struct Reference

Detailed Description

The following describes the structure *sv_direct_info* which is used by the function *sv_direct_status()*.

```
typedef struct {
  int size;                 // Size of the structure (needs to be filled by the
                            // caller).
  int available;            // For an input this element is the number of filled/
                            // recorded buffers that can be fetched by
                            // sv_direct_record() without blocking. For an output this
                            // element is the number of buffers that can be queued to
                            // the driver without blocking.
  int dropped;              // Number of buffers that were dropped since calling the
                            // function sv_direct_init().
  int when;                 // Buffer tick.
  int clock_high;           // Buffer clock (upper 32 bits).
  int clock_low;            // Buffer clock (lower 32 bits).
  int pad[16];              // Reserved for future use.
} sv_direct_info;
```

sv_direct_timecode Struct Reference

Detailed Description

The following describes the structure *sv_direct_timecode* which is used by the function *sv_direct_bind_timecode()*.

```
typedef struct {
  int size;                 // Size of the structure (needs to be filled by the
                            // caller).
  int ltc_tc;               // Analog LTC timecode without bit masking.
  int ltc_ub;               // Analog LTC user bytes.
  int vtr_tc;               // VTR timecode.
  int vtr_ub;               // VTR user bytes.
  int vitc_tc[2];           // Analog VITC timecode.
  int vitc_ub[2];           // Analog VITC user bytes.
  int film_tc[2];           // Analog film timecode.
  int film_ub[2];           // Analog film user bytes.
  int prod_tc[2];           // Analog production timecode.
  int prod_ub[2];           // Analog production user bytes.
  int dvitc_tc[2];          // Digital/ANC VITC timecode.
  int dvitc_ub[2];          // Digital/ANC VITC user bytes.
  int dfilm_tc[2];          // Digital/ANC film timecode (RP201).
  int dfilm_ub[2];          // Digital/ANC film user bytes (RP201).
  int dprod_tc[2];          // Digital/ANC production timecode (RP201).
  int dprod_ub[2];          // Digital/ANC production user bytes (RP201).
  int dltc_tc;              // Digital/ANC LTC timecode.
  int dltc_ub;              // Digital/ANC LTC user bytes.
  int gpi;                  // GPI information of the buffer.
  int pad[16];              // Reserved for future use.
} sv_direct_timecode;
```

sv_fifo_ancbuffer Struct Reference

Detailed Description

The following details the ANC data structure used by the function *sv_fifo_anc()*.

```
typedef struct {
  int            linenr;       // Line number of this ANC packet.
  int            did;          // Data ID of this ANC packet.
  int            sdid;         // Secondary data ID of this ANC packet.
  int            datasize;     // Data payload, i.e. the number of bytes in the data
                               // element.
  int            vanc;         // Position of this packet. Either VANC (1) or
                               // HANC (0).
  int            field;        // Field index (0 or 1) of this ANC packet.
  int            pad[6];       // Reserved for future use. Set to zero (0).
  unsigned char  data[256];    // Buffer for the data payload (element 'datasize').
} sv_fifo_ancbuffer;
```

sv_fifo_buffer Struct Reference

Detailed Description

The following details the FIFO buffer structure used, for example, by the functions *sv_fifo_getbuffer()* and *sv_fifo_putbuffer()*.

```
typedef struct {
  int version;              // Used internally only, i.e. do not change.
  int size;                 // Size of this structure in bytes. Do not change.
  int fifoid;               // FIFO/frame ID of the buffer. Do not change.
  int flags;                // Used internally only, i.e. do not change.
  struct {
    char * addr;            // Pointer for the DMA.
    int    size;            // Size of the DMA.
  } dma;                    // DMA substructure (not used when setting the define
                            // SV_FIFO_FLAG_NODMAADDR).
  struct {
    char * addr;            // Pointer/offset to the video data.
    int    size;            // Size of the video data.
  } video[2];               // Array containing two video fields.
  struct {
    char * addr[4];         // Pointer/offset to the audio data of the channels 1 to 4.
    int    size;            // Size of the audio buffer.
  } audio[2];               // Array containing the audio for two video fields.
  struct {
    int    ltc_tc;          // Analog LTC timecode without bit masking.
    int    ltc_ub;          // Analog LTC user bytes.
    int    vitc_tc;         // Analog VITC timecode.
    int    vitc_ub;         // Analog VITC user bytes.
    int    vtr_tick;        // Capture tick for VTR timecode and user bytes.
    int    vtr_tc;          // VTR timecode.
    int    vtr_ub;          // VTR user bytes.
    int    vtr_info3;       // VTR info bytes 8 to 11.
    int    vtr_info;        // VTR info bytes 0 to 3.
    int    vtr_info2;       // VTR info bytes 4 to 7.
    int    vitc_tc2;        // Analog VITC timecode of field 2.
    int    vitc_ub2;        // Analog VITC user bytes of field 2.
    int    pad[4];          // Reserved for future use.
  } timecode;               // Timecode substructure.
  struct {
    int tick;               // Tick when the frame was captured. Valid for an input
                            // only.
    int clock_high;         // Clock of the MSBs (most significant bytes) when a frame
                            // was captured. Valid for an input only.
    int clock_low;          // Clock of the LSBs (least significant bytes) when a frame
                            // was captured. Valid for an input only.
    int gpi;                // GPI information of the FIFO buffer.
    int aclock_high;        // Clock of the MSBs (most significant bytes) when audio was
                            // captured. Valid for an input only.
    int aclock_low;         // Clock of the LSBs (least significant bytes) when audio
                            // was captured. Valid for an input only.
    int pad[2];             // Reserved for future use.
  } control;                // Various buffer related values.
  struct {
    int cmd;                // The received VTR command. Valid for an input only.
    int length;             // Number of data bytes. Valid for an input only.
    unsigned char data[16];// Data bytes. Valid for an input only.
  } vtrcmd;                 // Substructure containing the incoming VTR commands.
  struct {
    char * addr[4];         // Pointer/offset to the audio data of the channels 5 to 8.
  } audio2[2];              // Array containing the audio for two video fields.
  struct {
    int storagemode;        // Image data storage mode.
    int xsize;              // Image data x-size.
    int ysize;              // Image data y-size.
    int xoffset;            // Image data x-offset from center.
    int yoffset;            // Image data y-offset from center.
```

```
    int dataoffset;       // Offset to the first pixel in the buffer.
    int lineoffset;       // Offset from line to line or zero (0) for default.
    int compression;      // Compression code of the video data.
    int encryption;       // Obsolete. Instead use sv_fifo_buffer.encryption.code.
    int matrixtype;       // Matrix type.
    int bufferid;         // Buffer ID from the Render API. For a normal FIFO
                          // operation it should be set to zero (0).
    int pad[4];           // Reserved for future use.
  } storage;              // Dynamic storage mode. Valid for an output only.
  struct {
    int dvitc_tc[2];      // Digital/ANC VITC timecode.
    int dvitc_ub[2];      // Digital/ANC VITC user bytes.
    int film_tc[2];       // Digital/ANC film timecode (RP201).
    int film_ub[2];       // Digital/ANC film user bytes (RP201).
    int prod_tc[2];       // Digital/ANC production timecode (RP201).
    int prod_ub[2];       // Digital/ANC production user bytes (RP201).
    int dltc_tc;          // Digital/ANC LTC timecode.
    int dltc_ub;          // Digital/ANC LTC user bytes.
    int closedcaption[2]; // Analog closed caption. Valid for an input only.
    int afilm_tc[2];      // Analog film timecode.
    int afilm_ub[2];      // Analog film user bytes.
    int aprod_tc[2];      // Analog production timecode.
    int aprod_ub[2];      // Analog production user bytes.
  } anctimecode;          // ANC timecode substructure.
  struct {
    char * addr;          // Pointer/offset to the video data channel B (second video
                          // image, see the define SV_FIFO_FLAG_VIDEO_B).
    int    size;          // Size of the video data.
  } video_b[2];           // Array containing two video fields (second video image,
                          // see the define SV_FIFO_FLAG_VIDEO_B).
  struct {
    char * addr;          // Pointer/offset to the ANC data.
    int    size;          // Size of the ANC data.
  } anc[2];               // Array containing two ANC fields.
  struct {
    int keyid;            // Decryption key ID.
    int payload;          // Amount of data (incl. plaintext and padding).
    int plaintext;        // Plaintext offset.
    int sourcelength;     // Original size of the non-encrypted data.
    int code;             // Encryption code of the video data.
    int pad[3];           // Reserved for future use.
  } encryption;           // Substructure for a decryption of video. It can be used
                          // with HydraCinema only.
  struct {
    int keyid;            // Decryption key ID.
    int payload;          // Amount of data (incl. plaintext and padding).
    int plaintext;        // Plaintext offset.
    int sourcelength;     // Original size of the non-encrypted data.
    int code;             // Encryption code of the audio data.
    int bits;             // Bit width of the decrypted data.
    int channels;         // Channel count of the decrypted data.
    int frequency;        // Frequency of the decrypted data.
    int littleendian;     // Endianness of the decrypted data. It is a boolean value
                          // and TRUE if the data is in little endian byte order.
    int bsigned;          // Signedness of the decrypted data. It is a boolean value
                          // and TRUE if the data must be interpreted as a signed
                          // integer.
  } encryption_audio;     // Substructure for a decryption of audio. It can be used
                          // with HydraCinema only.
  int pad[6];             // Reserved for future use.
} sv_fifo_buffer;
```

sv_fifo_bufferinfo Struct Reference

Detailed Description

The following details the structure of the buffer related timing information used by the functions *sv_fifo_getbuffer()* and *sv_fifo_putbuffer()*.

```
typedef struct {
  int version;                // Version of this structure.
  int size;                   // Size of the structure.
  int when;                   // Buffer tick.
  int clock_high;             // Buffer clock (upper 32 bits).
  int clock_low;              // Buffer clock (lower 32 bits).
  int clock_tolerance;        // Buffer clock operation tolerance.
  int padding[32-6];          // Reserved for future use.
} sv_fifo_bufferinfo;
```

sv_fifo_configinfo Struct Reference

Detailed Description

The following provides details about the structure *sv_fifo_configinfo* used by the function *sv_fifo_configstatus()* to return system parameters.

```
typedef struct {
  int     entries;            // Number of valid entries in this structure.
  int     dmaalignment;       // Needed alignment of a DMA buffer.
  int     nbuffers;           // Maximum number of buffers in this video mode.
  int     vbuffersize;        // Size of one video frame in the board memory plus an
                              // internal alignment. For the exact size use the
                              // function sv_storage_status().
  int     abuffersize;        // Size of the audio data in the board memory that
                              // corresponds to one frame of video.
  void *  unused1;            // No longer used.
  void *  unused2;            // No longer used.
  int     dmarect_xoffset;    // X-offset for the current DMA rectangle.
  int     dmarect_yoffset;    // Y-offset for the current DMA rectangle.
  int     dmarect_xsize;      // X-size for the current DMA rectangle.
  int     dmarect_ysize;      // Y-size for the current DMA rectangle.
  int     dmarect_lineoffset; // Line to line offset for the current DMA rectangle.
  int     field1offset;       // Offset to the start of field 1.
  int     field2offset;       // Offset to the start of field 2.
  int     ancbuffersize;      // Size of the ANC data in the board memory that
                              // corresponds to one frame of video.
  int     pad[64-15];         // Reserved for future use.
} sv_fifo_configinfo;
```

sv_fifo_info Struct Reference

Detailed Description

The following describes the structure *sv_fifo_info* which is used by the function *sv_fifo_status()*.

```
typedef struct {
  int nbuffers;           // Absolute FIFO depth. The real FIFO depth is:
                          // <nReal> = sv_fifo_info.nbuffers - 1.
  int availbuffers;       // For an input this element returns the number of filled
                          // buffers, while for an output it returns the number of
                          // free/empty buffers, i.e. for both the remaining
                          // sv_fifo_getbuffer() calls that can be used. You can
                          // calculate the reverse with:
                          // <n> = sv_fifo_info.nbuffers - sv_fifo_info.availbuffers.
  int tick;               // Current tick.
  int clock_high;         // Clock time of the last vertical sync (upper 32 bits).
  int clock_low;          // Clock time of the last vertical sync (lower 32 bits).
  int dropped;            // Number of frames that were dropped since calling the
                          // function sv_fifo_start().
  int clocknow_high;      // Current clock time (upper 32 bits).
  int clocknow_low;       // Current clock time (lower 32 bits).
  int waitdropped;        // For DVS internal use. Number of dropped waits for the
                          // vertical sync of the function sv_fifo_getbuffer().
  int waitnotintime;      // For DVS internal use. Number of times with waits for the
                          // vertical sync that occurred not in real-time.
  int audioinerror;       // Audio input error code.
  int videoinerror;       // Video input error code.
  int displaytick;        // Current display tick.
  int recordtick;         // Current record tick.
  int openprogram;        // Program which tried to open the device.
  int opentick;           // Tick of the time when the program tried to open the
                          // device.
  int pad26[8];           // Reserved for future use.
} sv_fifo_info;
```

Index

API - Audio Functions 10
API - Basic Functions 1
API - Control Functions 78
API - Direct API 61
API - FIFO API 34
API - GPI Functionality 111
API - Hardware 117
API - Jack API 68
API - Proxy Capture 113
API - RS-422 High-level API (Master) 90
API - RS-422 High-level API (Slave) 97
API - RS-422 Low-level API 98
API - Status Functions 85
API - Storage Functions 121
API - The sv_option() Functions 74
API - The sv_query() Function 77
API - Timecode 100
API - Tracing 120
API - Video Functions 15
conventions of manual iv
directapi
  sv_direct_bind_buffer 63
  sv_direct_bind_opengl 63
  sv_direct_bind_timecode 63
  sv_direct_display 64
  SV_DIRECT_FLAG_DISCARD 62
  SV_DIRECT_FLAG_FIELD 62
  sv_direct_free 65
  sv_direct_init 65
  sv_direct_record 66
  sv_direct_status 66
  sv_direct_sync 67
  sv_direct_unbind 67
dpxio_exec
  svdpxio 149
dpxio_set_timecodes
  svdpxio 152
dpxio_tracelog
  svdpxio 152
DVS SDK 4.<x> features v
DVS video board products v
dvssdktrace
  SV_OPTION_TRACE 120
Example - dpxio 149
Example Projects Overview 147
fifoapi
  sv_fifo 44
  sv_fifo_anc 44
  sv_fifo_ancdata 45
  sv_fifo_anclayout 46
  SV_FIFO_BUFFERINFO_VERSION_1 39
  sv_fifo_bypass 47
sv_fifo_configstatus 47
sv_fifo_dmarectangle 47
SV_FIFO_FLAG_ANC 39
SV_FIFO_FLAG_AUDIOINTERLEAVED 39
SV_FIFO_FLAG_AUDIOONLY 40
SV_FIFO_FLAG_CLOCKEDOPERATION 40
SV_FIFO_FLAG_DMARECTANGLE 40
SV_FIFO_FLAG_DONTBLOCK 40
SV_FIFO_FLAG_FIELD 40
SV_FIFO_FLAG_FLUSH 41
SV_FIFO_FLAG_NO_LIVE 41
SV_FIFO_FLAG_NODMA 41
SV_FIFO_FLAG_NODMAADDR 41
SV_FIFO_FLAG_PULLDOWN 42
SV_FIFO_FLAG_REPEAT_2TIMES 42
SV_FIFO_FLAG_REPEAT_3TIMES 42
SV_FIFO_FLAG_REPEAT_4TIMES 42
SV_FIFO_FLAG_REPEAT_MASK 42
SV_FIFO_FLAG_REPEAT_ONCE 42
SV_FIFO_FLAG_SETAUDIOSIZE 43
SV_FIFO_FLAG_STORAGEMODE 43
SV_FIFO_FLAG_STORAGENOAUTOCENTER 43
SV_FIFO_FLAG_TIMEDOPERATION 43
SV_FIFO_FLAG_VIDEO_B 43
SV_FIFO_FLAG_VIDEOONLY 43
SV_FIFO_FLAG_VSYNCWAIT 44
sv_fifo_free 48
sv_fifo_getbuffer 48
sv_fifo_init 49
sv_fifo_lut 50
sv_fifo_matrix 51
sv_fifo_putbuffer 52
sv_fifo_reset 52
sv_fifo_sanitycheck 53
sv_fifo_sanitylevel 53
sv_fifo_start 54
sv_fifo_startex 54
sv_fifo_status 55
sv_fifo_stop 55
sv_fifo_stopex 56
sv_fifo_vsyncwait 56
sv_fifo_wait 56
sv_memory_dma 57
sv_memory_dma_ready 57
sv_memory_dmaex 58
sv_memory_dmarect 58
sv_memory_dmax 59
sv_memory_frameinfo 59
SV_OPTION_DROPMODE 44
SV_OPTION_WATCHDOG_ACTION 44
SV_OPTION_WATCHDOG_TIMEOUT 44

Info - Audio Formats 138
Info - Bit Formats 135
Info - Error Codes 140
Info - Pixel Formats 137
Info - Storage Formats 139
jackapi
  sv_jack_memorysetup 71
  sv_jack_option_get 72
  sv_jack_option_set 72
  sv_jack_query 73
  sv_jack_status 73
  SV_OPTION_MULTICHANNEL 70
main
  svdpxio 152
new in DVS SDK 4.<x> v
obsolete
  sv_get_version 133
  sv_memory_play 133
  sv_memory_record 133
  SV_OPTION_ASSIGN_LTC 132
  SV_OPTION_ASSIGN_VTR 132
  SV_OPTION_RS422A_PINOUT 132
  SV_OPTION_RS422B_PINOUT 132
  sv_vtrerror 134
Obsolete Defines and Functions 132
SDK 4.<x> features v
supported DVS video board products v
supported video rasters vi
sv_asc2tc
  svvtrmaster 91
sv_black
  svstorage 124
sv_capture
  svcapture 114
sv_capture_ready
  svcapture 115
sv_capture_status
  svcapture 115
sv_captureex
  svcapture 115
sv_close
  svclib 2
sv_colorbar
  svstorage 125
sv_currenttime
  svclib 2
sv_debugprint
  svclib 3
sv_direct_bind_buffer
  directapi 63
sv_direct_bind_opengl
  directapi 63
sv_direct_bind_timecode
  directapi 63
sv_direct_bufferinfo 153
sv_direct_display
  directapi 64
SV_DIRECT_FLAG_DISCARD
  directapi 62
SV_DIRECT_FLAG_FIELD
  directapi 62
sv_direct_free
  directapi 65
sv_direct_info 154
sv_direct_init
  directapi 65
sv_direct_record
  directapi 66
sv_direct_status
  directapi 66
sv_direct_sync
  directapi 67
sv_direct_timecode 155
sv_direct_unbind
  directapi 67
sv_display
  svstorage 125
sv_errorprint
  svclib 3
sv_errorstring
  svclib 3
sv_fifo
  fifoapi 44
sv_fifo_anc
  fifoapi 44
sv_fifo_ancbuffer 156
sv_fifo_ancdata
  fifoapi 45
sv_fifo_anclayout
  fifoapi 46
sv_fifo_buffer 157
sv_fifo_bufferinfo 159
SV_FIFO_BUFFERINFO_VERSION_1
  fifoapi 39
sv_fifo_bypass
  fifoapi 47
sv_fifo_configinfo 160
sv_fifo_configstatus
  fifoapi 47
sv_fifo_dmarectangle
  fifoapi 47
SV_FIFO_FLAG_ANC
  fifoapi 39
SV_FIFO_FLAG_AUDIOINTERLEAVED
  fifoapi 39
SV_FIFO_FLAG_AUDIOONLY
  fifoapi 40
SV_FIFO_FLAG_CLOCKEDOPERATION
  fifoapi 40
SV_FIFO_FLAG_DMARECTANGLE
  fifoapi 40
SV_FIFO_FLAG_DONTBLOCK
  fifoapi 40
SV_FIFO_FLAG_FIELD
  fifoapi 40
SV_FIFO_FLAG_FLUSH

fifoapi 41
SV_FIFO_FLAG_NO_LIVE
fifoapi 41
SV_FIFO_FLAG_NODMA
fifoapi 41
SV_FIFO_FLAG_NODMAADDR
fifoapi 41
SV_FIFO_FLAG_PULLDOWN
fifoapi 42
SV_FIFO_FLAG_REPEAT_2TIMES
fifoapi 42
SV_FIFO_FLAG_REPEAT_3TIMES
fifoapi 42
SV_FIFO_FLAG_REPEAT_4TIMES
fifoapi 42
SV_FIFO_FLAG_REPEAT_MASK
fifoapi 42
SV_FIFO_FLAG_REPEAT_ONCE
fifoapi 42
SV_FIFO_FLAG_SETAUDIOSIZE
fifoapi 43
SV_FIFO_FLAG_STORAGEMODE
fifoapi 43
SV_FIFO_FLAG_STORAGENOAUTOCENTER
fifoapi 43
SV_FIFO_FLAG_TIMEDOPERATION
fifoapi 43
SV_FIFO_FLAG_VIDEO_B
fifoapi 43
SV_FIFO_FLAG_VIDEOONLY
fifoapi 43
SV_FIFO_FLAG_VSYNCWAIT
fifoapi 44
sv_fifo_free
fifoapi 48
sv_fifo_getbuffer
fifoapi 48
sv_fifo_info 161
sv_fifo_init
fifoapi 49
sv_fifo_lut
fifoapi 50
sv_fifo_matrix
fifoapi 51
sv_fifo_putbuffer
fifoapi 52
sv_fifo_reset
fifoapi 52
sv_fifo_sanitycheck
fifoapi 53
sv_fifo_sanitylevel
fifoapi 53
sv_fifo_start
fifoapi 54
sv_fifo_startex
fifoapi 54
sv_fifo_status
fifoapi 55
sv_fifo_stop
fifoapi 55
sv_fifo_stopex
fifoapi 56
sv_fifo_vsyncwait
fifoapi 56
sv_fifo_wait
fifoapi 56
sv_get_version
obsolete 133
sv_geterrortext
svclib 3
sv_getlicence
svclib 4
sv_goto
svstorage 126
sv_host2sv
svstorage 126
sv_inpoint
svstorage 128
sv_jack_memorysetup
jackapi 71
sv_jack_option_get
jackapi 72
sv_jack_option_set
jackapi 72
sv_jack_query
jackapi 73
sv_jack_status
jackapi 73
sv_licence
svclib 4
sv_licencebit2string
svclib 5
sv_licenceinfo
svclib 5
sv_live
svstorage 128
sv_lut
svcontrol 80
sv_matrix
svcontrol 80
sv_matrixex
svcontrol 82
sv_memory_dma
fifoapi 57
sv_memory_dma_ready
fifoapi 57
sv_memory_dmaex
fifoapi 58
sv_memory_dmarect
fifoapi 58
sv_memory_dmax
fifoapi 59
sv_memory_frameinfo
fifoapi 59
sv_memory_play
obsolete 133

sv_memory_record
obsolete 133
sv_open
svclib 5
sv_openex
svclib 6
sv_option
svoption 74
SV_OPTION_AFILM_TC
svtimecode 101
SV_OPTION_AFILM_UB
svtimecode 101
SV_OPTION_ALPHAGAIN
svvideo 17
SV_OPTION_ALPHAMIXER
svvideo 18
SV_OPTION_ALPHAOFFSET
svvideo 19
SV_OPTION_ANCCOMPLETE
svtimecode 102
SV_OPTION_ANCGENERATOR
svtimecode 102
SV_OPTION_ANCGENERATOR_RP165
svtimecode 103
SV_OPTION_ANCREADER
svtimecode 103
SV_OPTION_ANCUSER_DID
svtimecode 103
SV_OPTION_ANCUSER_FLAGS
svtimecode 103
SV_OPTION_ANCUSER_LINENR
svtimecode 103
SV_OPTION_ANCUSER_SDID
svtimecode 103
SV_OPTION_APROD_TC
svtimecode 103
SV_OPTION_APROD_UB
svtimecode 103
SV_OPTION_ASSIGN_LTC
obsolete 132
SV_OPTION_ASSIGN_LTCA
svtimecode 103
SV_OPTION_ASSIGN_VTR
obsolete 132
SV_OPTION_AUDIOAESROUTING
svaudio 10
SV_OPTION_AUDIOAESSOURCE
svaudio 11
SV_OPTION_AUDIOANALOGOUT
svaudio 11
SV_OPTION_AUDIOBITS
svaudio 12
SV_OPTION_AUDIOCHANNELS
svaudio 12
SV_OPTION_AUDIODRIFT_ADJUST
svaudio 12
SV_OPTION_AUDIOFREQ
svaudio 12
SV_OPTION_AUDIOINPUT
svaudio 12
SV_OPTION_AUDIOMAXAIV
svaudio 12
SV_OPTION_AUDIOMUTE
svaudio 12
SV_OPTION_AUDIONOFADING
svaudio 13
SV_OPTION_DEBUG
svclib 2
SV_OPTION_DETECTION_NO4K
svvideo 19
SV_OPTION_DETECTION_TOLERANCE
svvideo 19
SV_OPTION_DISABLESWITCHINGLINE
svvideo 19
SV_OPTION_DLTC_TC
svtimecode 104
SV_OPTION_DLTC_UB
svtimecode 104
SV_OPTION_DROPMODE
fifoapi 44
SV_OPTION_DVI_OUTPUT
svvideo 19
SV_OPTION_DVITC_TC
svtimecode 104
SV_OPTION_DVITC_UB
svtimecode 104
SV_OPTION_FIELD_DOMINANCE
svvideo 20
SV_OPTION_FILM_TC
svtimecode 104
SV_OPTION_FILM_UB
svtimecode 104
SV_OPTION_FLUSH_TIMECODE
svtimecode 105
sv_option_get
svoption 74
SV_OPTION_GPI
svgpi 111
SV_OPTION_GPIIN
svgpi 111
SV_OPTION_GPIOUT
svgpi 111
SV_OPTION_HDELAY
svvideo 20
SV_OPTION_HWWATCHDOG_ACTION
svvideo 20
SV_OPTION_HWWATCHDOG_REFRESH
svvideo 20
SV_OPTION_HWWATCHDOG_RELAY_DELAY
svvideo 21
SV_OPTION_HWWATCHDOG_TIMEOUT
svvideo 21
SV_OPTION_HWWATCHDOG_TRIGGER
svvideo 21
SV_OPTION_INPUTFILTER
svvideo 21

SV_OPTION_INPUTPORT
svvideo 21
SV_OPTION_IOMODE
svvideo 22
SV_OPTION_IOMODE_AUTODETECT
svvideo 22
SV_OPTION_IOSPEED
svvideo 23
SV_OPTION_LOOPMODE
svstorage 122
SV_OPTION_LTC_TC
svtimecode 105
SV_OPTION_LTC_UB
svtimecode 105
SV_OPTION_LTCDELAY
svtimecode 105
SV_OPTION_LTCDROPFRAME
svtimecode 105
SV_OPTION_LTCFILTER
svtimecode 105
SV_OPTION_LTCOFFSET
svtimecode 105
SV_OPTION_LTCSOURCE
svtimecode 105
SV_OPTION_MAINOUTPUT
svvideo 23
sv_option_menu
svoption 75
SV_OPTION_MULTICHANNEL
jackapi 70
SV_OPTION_NOP
svclib 2
SV_OPTION_OUTPUTFILTER
svvideo 23
SV_OPTION_OUTPUTPORT
svvideo 23
SV_OPTION_PROD_TC
svtimecode 106
SV_OPTION_PROD_UB
svtimecode 106
SV_OPTION_PROXY_ASPECTRATIO
svcapture 113
SV_OPTION_PROXY_OPTIONS
svcapture 113
SV_OPTION_PROXY_OUTPUT
svcapture 113
SV_OPTION_PROXY_SYNCMODE
svcapture 114
SV_OPTION_PROXY_VIDEOMODE
svcapture 114
SV_OPTION_PULLDOWN_STARTLTC
svvideo 23
SV_OPTION_PULLDOWN_STARTPHASE
svvideo 24
SV_OPTION_PULLDOWN_STARTVTRTC
svvideo 24
SV_OPTION_REPEAT
svstorage 122
SV_OPTION_RS422A
svcontrol 78
SV_OPTION_RS422A_PINOUT
obsolete 132
SV_OPTION_RS422B
svcontrol 79
SV_OPTION_RS422B_PINOUT
obsolete 132
sv_option_set
svoption 75
sv_option_setat
svoption 76
SV_OPTION_SLOWMOTION
svstorage 122
SV_OPTION_SPEED
svstorage 123
SV_OPTION_SPEEDBASE
svstorage 123
SV_OPTION_SWITCH_TOLERANCE
svvideo 24
SV_OPTION_SYNCMODE
svvideo 24
SV_OPTION_SYNCOUT
svvideo 25
SV_OPTION_SYNCOUTDELAY
svvideo 25
SV_OPTION_SYNCOUTVDELAY
svvideo 25
SV_OPTION_SYNCSELECT
svvideo 25
SV_OPTION_TRACE
dvssdktrace 120
SV_OPTION_VDELAY
svvideo 25
SV_OPTION_VIDEOMODE
svvideo 25
SV_OPTION_VITC_TC
svtimecode 106
SV_OPTION_VITC_UB
svtimecode 106
SV_OPTION_VITCLINE
svtimecode 106
SV_OPTION_VITCREADERLINE
svtimecode 107
SV_OPTION_VTR_INFO
svtimecode 107
SV_OPTION_VTR_INFO2
svtimecode 107
SV_OPTION_VTR_INFO3
svtimecode 107
SV_OPTION_VTR_TC
svtimecode 107
SV_OPTION_VTR_UB
svtimecode 107
SV_OPTION_VTRMASTER_EDITLAG
svvtrmaster 90
SV_OPTION_VTRMASTER_FLAGS
svvtrmaster 90

SV_OPTION_VTRMASTER_POSTROLL
svvtrmaster 91
SV_OPTION_VTRMASTER_PREROLL
svvtrmaster 91
SV_OPTION_VTRMASTER_TCTYPE
svvtrmaster 91
SV_OPTION_VTRMASTER_TOLERANCE
svvtrmaster 91
SV_OPTION_WATCHDOG_ACTION
fifoapi 44
SV_OPTION_WATCHDOG_TIMEOUT
fifoapi 44
SV_OPTION_WORDCLOCK
svaudio 13
sv_outpoint
svstorage 128
sv_position
svstorage 129
sv_preset
svstorage 129
sv_pulldown
svvideo 30
sv_query
svquery 77
SV_QUERY_AFILM_TC
svtimecode 107
SV_QUERY_AFILM_UB
svtimecode 108
SV_QUERY_ANC_MAXHANCLINENR
svtimecode 108
SV_QUERY_ANC_MAXVANCLINENR
svtimecode 108
SV_QUERY_ANC_MINLINENR
svtimecode 108
SV_QUERY_APROD_TC
svtimecode 108
SV_QUERY_APROD_UB
svtimecode 108
SV_QUERY_AUDIO_AESCHANNELS
svaudio 13
SV_QUERY_AUDIO_AIVCHANNELS
svaudio 13
SV_QUERY_AUDIO_MAXCHANNELS
svaudio 13
SV_QUERY_AUDIOBITS
svaudio 13
SV_QUERY_AUDIOCHANNELS
svaudio 13
SV_QUERY_AUDIOFREQ
svaudio 13
SV_QUERY_AUDIOINERROR
svaudio 13
SV_QUERY_AUDIOINPUT
svaudio 13
SV_QUERY_AUDIOMUTE
svaudio 13
SV_QUERY_AUDIOSIZE
svstorage 123
SV_QUERY_AUDIOSIZE_FROMHOST
svstorage 123
SV_QUERY_AUDIOSIZE_TOHOST
svstorage 123
SV_QUERY_CARRIER
svvideo 26
SV_QUERY_DEVTYPE
svstatus 85
SV_QUERY_DISPLAY_LINENR
svvideo 26
SV_QUERY_DLTC_TC
svtimecode 108
SV_QUERY_DLTC_UB
svtimecode 108
SV_QUERY_DMAALIGNMENT
svhardware 117
SV_QUERY_DVITC_TC
svtimecode 108
SV_QUERY_DVITC_UB
svtimecode 108
SV_QUERY_FANSPEED
svhardware 117
SV_QUERY_FEATURE
svstatus 85
SV_QUERY_FEATURE_AUDIOCHANNELS
svstatus 86
SV_QUERY_FILM_TC
svtimecode 109
SV_QUERY_FILM_UB
svtimecode 109
SV_QUERY_GENLOCK
svvideo 26
SV_QUERY_GPI
svgpi 112
SV_QUERY_GPIIN
svgpi 112
SV_QUERY_GPIOUT
svgpi 112
SV_QUERY_HDELAY
svvideo 26
SV_QUERY_HW_CARDOPTIONS
svhardware 117
SV_QUERY_HW_CARDVERSION
svhardware 117
SV_QUERY_HW_EPLDOPTIONS
svhardware 118
SV_QUERY_HW_EPLDVERSION
svhardware 118
SV_QUERY_HW_PCIELANES
svhardware 118
SV_QUERY_HW_PCISPEED
svhardware 118
SV_QUERY_HW_PCIWIDTH
svhardware 118
SV_QUERY_INPUTFILTER
svvideo 26
SV_QUERY_INPUTPORT
svvideo 26

SV_QUERY_INPUTRASTER
svvideo 26
SV_QUERY_INPUTRASTER_GENLOCK
svvideo 26
SV_QUERY_INPUTRASTER_GENLOCK_TYPE
svvideo 26
SV_QUERY_INPUTRASTER_SDIA
svvideo 26
SV_QUERY_INPUTRASTER_SDIB
svvideo 27
SV_QUERY_INPUTRASTER_SDIC
svvideo 27
SV_QUERY_INTERLACEID_STORAGE
svstorage 123
SV_QUERY_INTERLACEID_VIDEO
svstorage 124
SV_QUERY_IOCHANNELS
svvideo 27
SV_QUERY_IOLINK_MAPPING
svvideo 27
SV_QUERY_IOLINKS_INPUT
svvideo 27
SV_QUERY_IOLINKS_OUTPUT
svvideo 27
SV_QUERY_IOMODE
svvideo 27
SV_QUERY_IOMODEINERROR
svvideo 27
SV_QUERY_IOSPEED
svvideo 28
SV_QUERY_IOSPEED_SDIA
svvideo 28
SV_QUERY_IOSPEED_SDIB
svvideo 28
SV_QUERY_IOSPEED_SDIC
svvideo 28
SV_QUERY_IOSPEED_SDID
svvideo 28
SV_QUERY_LOOPMODE
svstorage 124
SV_QUERY_LTCAVAILABLE
svtimecode 109
SV_QUERY_LTCDROPFRAME
svtimecode 109
SV_QUERY_LTCFILTER
svtimecode 109
SV_QUERY_LTCOFFSET
svtimecode 109
SV_QUERY_LTCSOURCE
svtimecode 109
SV_QUERY_LTCTIMECODE
svtimecode 109
SV_QUERY_LTCUSERBYTES
svtimecode 109
SV_QUERY_MODE_AVAILABLE
svvideo 28
SV_QUERY_MODE_CURRENT
svvideo 28
SV_QUERY_OUTPUTFILTER
svvideo 28
SV_QUERY_OUTPUTPORT
svvideo 28
SV_QUERY_PRESET
svstorage 124
SV_QUERY_PROD_TC
svtimecode 109
SV_QUERY_PROD_UB
svtimecode 109
SV_QUERY_PULLDOWN
svcontrol 79
SV_QUERY_RASTER_DROPFRAME
svvideo 28
SV_QUERY_RASTER_FPS
svvideo 29
SV_QUERY_RASTER_INTERLACE
svvideo 29
SV_QUERY_RASTER_SEGMENTED
svvideo 29
SV_QUERY_RASTER_XSIZE
svvideo 29
SV_QUERY_RASTER_YSIZE
svvideo 29
SV_QUERY_RASTERID
svvideo 29
SV_QUERY_RECORD_LINENR
svvideo 29
SV_QUERY_REPEATMODE
svstorage 124
SV_QUERY_SERIALNUMBER
svhardware 118
SV_QUERY_SLOWMOTION
svstorage 124
SV_QUERY_SMPTE352
svvideo 29
SV_QUERY_STORAGE_XSIZE
svvideo 29
SV_QUERY_STORAGE_YSIZE
svvideo 29
SV_QUERY_STREAMERSIZE
svstorage 124
SV_QUERY_SYNCMODE
svvideo 30
SV_QUERY_SYNCOUT
svvideo 30
SV_QUERY_SYNCOUTDELAY
svvideo 30
SV_QUERY_SYNCOUTVDELAY
svvideo 30
SV_QUERY_SYNCSTATE
svvideo 30
SV_QUERY_TEMPERATURE
svhardware 118
SV_QUERY_TICK
svvideo 30
SV_QUERY_VALIDTIMECODE
svtimecode 110

SV_QUERY_VALUE_AVAILABLE
  svstatus 86
SV_QUERY_VDELAY
  svvideo 30
SV_QUERY_VERSION_DRIVER
  svstatus 86
SV_QUERY_VERSION_DVSOEM
  svstatus 86
SV_QUERY_VIDEOINERROR
  svvideo 30
SV_QUERY_VITCLINE
  svtimecode 110
SV_QUERY_VITCREADERLINE
  svtimecode 110
SV_QUERY_VITCTIMECODE
  svtimecode 110
SV_QUERY_VITCUSERBYTES
  svtimecode 110
SV_QUERY_VOLTAGE_12V0
  svhardware 118
SV_QUERY_VOLTAGE_1V0
  svhardware 118
SV_QUERY_VOLTAGE_1V2
  svhardware 118
SV_QUERY_VOLTAGE_1V5
  svhardware 118
SV_QUERY_VOLTAGE_1V8
  svhardware 118
SV_QUERY_VOLTAGE_2V5
  svhardware 119
SV_QUERY_VOLTAGE_3V3
  svhardware 119
SV_QUERY_VOLTAGE_5V0
  svhardware 119
SV_QUERY_WORDCLOCK
  svaudio 14
SV_QUERY_XPANNING
  svcontrol 79
SV_QUERY_XZOOM
  svcontrol 79
SV_QUERY_YPANNING
  svcontrol 79
SV_QUERY_YZOOM
  svcontrol 80
SV_QUERY_ZOOMFLAGS
  svcontrol 80
sv_raster_status
  svstatus 86
sv_record
  svstorage 129
sv_rs422_close
  svrs422 98
sv_rs422_open
  svrs422 98
sv_rs422_rw
  svrs422 99
sv_showinput
  svstorage 130
sv_slaveinfo_get
  svslaveinfo 97
sv_slaveinfo_set
  svslaveinfo 97
sv_status
  svstatus 87
sv_stop
  svstorage 130
sv_storage_status
  svstatus 87
sv_sv2host
  svstorage 131
sv_sync
  svvideo 31
sv_sync_output
  svvideo 32
sv_tc2asc
  svvtrmaster 92
sv_timecode_feedback
  svtimecode 110
sv_usleep
  svclib 7
sv_version_certify
  svclib 7
sv_version_check
  svclib 8
sv_version_check_firmware
  svclib 8
sv_version_status
  svstatus 88
sv_version_verify
  svclib 9
sv_videomode
  svvideo 33
sv_vsyncwait
  svvideo 33
sv_vtrcontrol
  svvtrmaster 92
sv_vtrerror
  obsolete 134
sv_vtrmaster
  svvtrmaster 93
sv_vtrmaster_raw
  svvtrmaster 95
sv_zoom
  svcontrol 83
svaudio
  SV_OPTION_AUDIOAESROUTING 10
  SV_OPTION_AUDIOAESSOURCE 11
  SV_OPTION_AUDIOANALOGOUT 11
  SV_OPTION_AUDIOBITS 12
  SV_OPTION_AUDIOCHANNELS 12
  SV_OPTION_AUDIODRIFT_ADJUST 12
  SV_OPTION_AUDIOFREQ 12
  SV_OPTION_AUDIOINPUT 12
  SV_OPTION_AUDIOMAXAIV 12
  SV_OPTION_AUDIOMUTE 12
  SV_OPTION_AUDIONOFADING 13

SV_OPTION_WORDCLOCK 13
SV_QUERY_AUDIO_AESCHANNELS 13
SV_QUERY_AUDIO_AIVCHANNELS 13
SV_QUERY_AUDIO_MAXCHANNELS 13
SV_QUERY_AUDIOBITS 13
SV_QUERY_AUDIOCHANNELS 13
SV_QUERY_AUDIOFREQ 13
SV_QUERY_AUDIOINERROR 13
SV_QUERY_AUDIOINPUT 13
SV_QUERY_AUDIOMUTE 13
SV_QUERY_WORDCLOCK 14
svcapture
sv_capture 114
sv_capture_ready 115
sv_capture_status 115
sv_captureex 115
SV_OPTION_PROXY_ASPECTRATIO 113
SV_OPTION_PROXY_OPTIONS 113
SV_OPTION_PROXY_OUTPUT 113
SV_OPTION_PROXY_SYNCMODE 114
SV_OPTION_PROXY_VIDEOMODE 114
svclib
sv_close 2
sv_currenttime 2
sv_debugprint 3
sv_errorprint 3
sv_errorstring 3
sv_geterrortext 3
sv_getlicence 4
sv_licence 4
sv_licencebit2string 5
sv_licenceinfo 5
sv_open 5
sv_openex 6
SV_OPTION_DEBUG 2
SV_OPTION_NOP 2
sv_usleep 7
sv_version_certify 7
sv_version_check 8
sv_version_check_firmware 8
sv_version_verify 9
svcontrol
sv_lut 80
sv_matrix 80
sv_matrixex 82
SV_OPTION_RS422A 78
SV_OPTION_RS422B 79
SV_QUERY_PULLDOWN 79
SV_QUERY_XPANNING 79
SV_QUERY_XZOOM 79
SV_QUERY_YPANNING 79
SV_QUERY_YZOOM 80
SV_QUERY_ZOOMFLAGS 80
sv_zoom 83
svdpxio
dpxio_exec 149
dpxio_set_timecodes 152
dpxio_tracelog 152
main 152
svgpi
SV_OPTION_GPI 111
SV_OPTION_GPIIN 111
SV_OPTION_GPIOUT 111
SV_QUERY_GPI 112
SV_QUERY_GPIIN 112
SV_QUERY_GPIOUT 112
svhardware
SV_QUERY_DMAALIGNMENT 117
SV_QUERY_FANSPEED 117
SV_QUERY_HW_CARDOPTIONS 117
SV_QUERY_HW_CARDVERSION 117
SV_QUERY_HW_EPLDOPTIONS 118
SV_QUERY_HW_EPLDVERSION 118
SV_QUERY_HW_PCIELANES 118
SV_QUERY_HW_PCISPEED 118
SV_QUERY_HW_PCIWIDTH 118
SV_QUERY_SERIALNUMBER 118
SV_QUERY_TEMPERATURE 118
SV_QUERY_VOLTAGE_12V0 118
SV_QUERY_VOLTAGE_1V0 118
SV_QUERY_VOLTAGE_1V2 118
SV_QUERY_VOLTAGE_1V5 118
SV_QUERY_VOLTAGE_1V8 118
SV_QUERY_VOLTAGE_2V5 119
SV_QUERY_VOLTAGE_3V3 119
SV_QUERY_VOLTAGE_5V0 119
svoption
sv_option 74
sv_option_get 74
sv_option_menu 75
sv_option_set 75
sv_option_setat 76
svquery
sv_query 77
svrs422
sv_rs422_close 98
sv_rs422_open 98
sv_rs422_rw 99
svslaveinfo
sv_slaveinfo_get 97
sv_slaveinfo_set 97
svstatus
SV_QUERY_DEVTYPE 85
SV_QUERY_FEATURE 85
SV_QUERY_FEATURE_AUDIOCHANNELS 86
SV_QUERY_VALUE_AVAILABLE 86
SV_QUERY_VERSION_DRIVER 86
SV_QUERY_VERSION_DVSOEM 86
sv_raster_status 86
sv_status 87
sv_storage_status 87
sv_version_status 88
svstorage
sv_black 124
sv_colorbar 125

sv_display 125
sv_goto 126
sv_host2sv 126
sv_inpoint 128
sv_live 128
SV_OPTION_LOOPMODE 122
SV_OPTION_REPEAT 122
SV_OPTION_SLOWMOTION 122
SV_OPTION_SPEED 123
SV_OPTION_SPEEDBASE 123
sv_outpoint 128
sv_position 129
sv_preset 129
SV_QUERY_AUDIOSIZE 123
SV_QUERY_AUDIOSIZE_FROMHOST 123
SV_QUERY_AUDIOSIZE_TOHOST 123
SV_QUERY_INTERLACEID_STORAGE 123
SV_QUERY_INTERLACEID_VIDEO 124
SV_QUERY_LOOPMODE 124
SV_QUERY_PRESET 124
SV_QUERY_REPEATMODE 124
SV_QUERY_SLOWMOTION 124
SV_QUERY_STREAMERSIZE 124
sv_record 129
sv_showinput 130
sv_stop 130
sv_sv2host 131
svtimecode
SV_OPTION_AFILM_TC 101
SV_OPTION_AFILM_UB 101
SV_OPTION_ANCCOMPLETE 102
SV_OPTION_ANCGENERATOR 102
SV_OPTION_ANCGENERATOR_RP165 103
SV_OPTION_ANCREADER 103
SV_OPTION_ANCUSER_DID 103
SV_OPTION_ANCUSER_FLAGS 103
SV_OPTION_ANCUSER_LINENR 103
SV_OPTION_ANCUSER_SDID 103
SV_OPTION_APROD_TC 103
SV_OPTION_APROD_UB 103
SV_OPTION_ASSIGN_LTCA 103
SV_OPTION_DLTC_TC 104
SV_OPTION_DLTC_UB 104
SV_OPTION_DVITC_TC 104
SV_OPTION_DVITC_UB 104
SV_OPTION_FILM_TC 104
SV_OPTION_FILM_UB 104
SV_OPTION_FLUSH_TIMECODE 105
SV_OPTION_LTC_TC 105
SV_OPTION_LTC_UB 105
SV_OPTION_LTCDELAY 105
SV_OPTION_LTCDROPFRAME 105
SV_OPTION_LTCFILTER 105
SV_OPTION_LTCOFFSET 105
SV_OPTION_LTCSOURCE 105
SV_OPTION_PROD_TC 106
SV_OPTION_PROD_UB 106
SV_OPTION_VITC_TC 106
SV_OPTION_VITC_UB 106
SV_OPTION_VITCLINE 106
SV_OPTION_VITCREADERLINE 107
SV_OPTION_VTR_INFO 107
SV_OPTION_VTR_INFO2 107
SV_OPTION_VTR_INFO3 107
SV_OPTION_VTR_TC 107
SV_OPTION_VTR_UB 107
SV_QUERY_AFILM_TC 107
SV_QUERY_AFILM_UB 108
SV_QUERY_ANC_MAXHANCLINENR 108
SV_QUERY_ANC_MAXVANCLINENR 108
SV_QUERY_ANC_MINLINENR 108
SV_QUERY_APROD_TC 108
SV_QUERY_APROD_UB 108
SV_QUERY_DLTC_TC 108
SV_QUERY_DLTC_UB 108
SV_QUERY_DVITC_TC 108
SV_QUERY_DVITC_UB 108
SV_QUERY_FILM_TC 109
SV_QUERY_FILM_UB 109
SV_QUERY_LTCAVAILABLE 109
SV_QUERY_LTCDROPFRAME 109
SV_QUERY_LTCFILTER 109
SV_QUERY_LTCOFFSET 109
SV_QUERY_LTCSOURCE 109
SV_QUERY_LTCTIMECODE 109
SV_QUERY_LTCUSERBYTES 109
SV_QUERY_PROD_TC 109
SV_QUERY_PROD_UB 109
SV_QUERY_VALIDTIMECODE 110
SV_QUERY_VITCLINE 110
SV_QUERY_VITCREADERLINE 110
SV_QUERY_VITCTIMECODE 110
SV_QUERY_VITCUSERBYTES 110
sv_timecode_feedback 110
svvideo
SV_OPTION_ALPHAGAIN 17
SV_OPTION_ALPHAMIXER 18
SV_OPTION_ALPHAOFFSET 19
SV_OPTION_DETECTION_NO4K 19
SV_OPTION_DETECTION_TOLERANCE 19
SV_OPTION_DISABLESWITCHINGLINE 19
SV_OPTION_DVI_OUTPUT 19
SV_OPTION_FIELD_DOMINANCE 20
SV_OPTION_HDELAY 20
SV_OPTION_HWWATCHDOG_ACTION 20
SV_OPTION_HWWATCHDOG_REFRESH 20
SV_OPTION_HWWATCHDOG_RELAY_DELAY 21
SV_OPTION_HWWATCHDOG_TIMEOUT 21
SV_OPTION_HWWATCHDOG_TRIGGER 21
SV_OPTION_INPUTFILTER 21
SV_OPTION_INPUTPORT 21
SV_OPTION_IOMODE 22
SV_OPTION_IOMODE_AUTODETECT 22
SV_OPTION_IOSPEED 23

SV_OPTION_MAINOUTPUT 23
SV_OPTION_OUTPUTFILTER 23
SV_OPTION_OUTPUTPORT 23
SV_OPTION_PULLDOWN_STARTLTC 23
SV_OPTION_PULLDOWN_STARTPHASE 24
SV_OPTION_PULLDOWN_STARTVTRTC 24
SV_OPTION_SWITCH_TOLERANCE 24
SV_OPTION_SYNCMODE 24
SV_OPTION_SYNCOUT 25
SV_OPTION_SYNCOUTDELAY 25
SV_OPTION_SYNCOUTVDELAY 25
SV_OPTION_SYNCSELECT 25
SV_OPTION_VDELAY 25
SV_OPTION_VIDEOMODE 25
sv_pulldown 30
SV_QUERY_CARRIER 26
SV_QUERY_DISPLAY_LINENR 26
SV_QUERY_GENLOCK 26
SV_QUERY_HDELAY 26
SV_QUERY_INPUTFILTER 26
SV_QUERY_INPUTPORT 26
SV_QUERY_INPUTRASTER 26
SV_QUERY_INPUTRASTER_GENLOCK 26
SV_QUERY_INPUTRASTER_GENLOCK_TYPE 26
SV_QUERY_INPUTRASTER_SDIA 26
SV_QUERY_INPUTRASTER_SDIB 27
SV_QUERY_INPUTRASTER_SDIC 27
SV_QUERY_IOCHANNELS 27
SV_QUERY_IOLINK_MAPPING 27
SV_QUERY_IOLINKS_INPUT 27
SV_QUERY_IOLINKS_OUTPUT 27
SV_QUERY_IOMODE 27
SV_QUERY_IOMODEINERROR 27
SV_QUERY_IOSPEED 28
SV_QUERY_IOSPEED_SDIA 28
SV_QUERY_IOSPEED_SDIB 28
SV_QUERY_IOSPEED_SDIC 28
SV_QUERY_IOSPEED_SDID 28
SV_QUERY_MODE_AVAILABLE 28
SV_QUERY_MODE_CURRENT 28
SV_QUERY_OUTPUTFILTER 28
SV_QUERY_OUTPUTPORT 28
SV_QUERY_RASTER_DROPFRAME 28
SV_QUERY_RASTER_FPS 29
SV_QUERY_RASTER_INTERLACE 29
SV_QUERY_RASTER_SEGMENTED 29
SV_QUERY_RASTER_XSIZE 29
SV_QUERY_RASTER_YSIZE 29
SV_QUERY_RASTERID 29
SV_QUERY_RECORD_LINENR 29
SV_QUERY_SMPTE352 29
SV_QUERY_STORAGE_XSIZE 29
SV_QUERY_STORAGE_YSIZE 29
SV_QUERY_SYNCMODE 30
SV_QUERY_SYNCOUT 30
SV_QUERY_SYNCOUTDELAY 30
SV_QUERY_SYNCOUTVDELAY 30
SV_QUERY_SYNCSTATE 30
SV_QUERY_TICK 30
SV_QUERY_VDELAY 30
SV_QUERY_VIDEOINERROR 30
sv_sync 31
sv_sync_output 32
sv_videomode 33
sv_vsyncwait 33
svvtrmaster
sv_asc2tc 91
SV_OPTION_VTRMASTER_EDITLAG 90
SV_OPTION_VTRMASTER_FLAGS 90
SV_OPTION_VTRMASTER_POSTROLL 91
SV_OPTION_VTRMASTER_PREROLL 91
SV_OPTION_VTRMASTER_TCTYPE 91
SV_OPTION_VTRMASTER_TOLERANCE 91
sv_tc2asc 92
sv_vtrcontrol 92
sv_vtrmaster 93
sv_vtrmaster_raw 95
target group of manual iv
tick 35
video rasters vi