DVS Digital Video Systems AG

DVS SDK - DVS Software Development Kit

Reference Guide

DVS Digital Video Systems AG
Version 3.2.0 for the DVS SDK 3.2.0.0 or higher (incl. 3.0.2.9 beta)

Table of Contents

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 OEM products manufactured by DVS. It is a software package that – once installed properly – provides a complete development and runtime environment, including not only an SDK but helpful tools and drivers as well.

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

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

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 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 OEM product.

What's New in the DVS SDK 3.0

The following details the most important features implemented in the DVS SDK 3.0 as well as the decisive differences compared to its predecessor version 2.7:

New Features:

- Multiple channels for in- and/or output.
- Different rasters between in- and output (independent I/O).
- Version resources in all `*.exe`, `*.dll` and `*.sys` files.
- Different drivers for the different DVS OEM products.
- The DVSConf program now only requires a single file to load/unload a driver (`dvs.inf`).
- Support of Windows Vista™.

Differences to Version 2.7:

- Different version numbering for fixes and patches.
- 64-bit DLLs renamed to `dvs<...>64.dll`.
- `*.mak` and `*.amd64` files removed from the DVS SDK.
- Mapped memory operation removed for the SDStationOEM and SDStationOEM II.
- No DMA into physical memory address.

Major Changes:

- Timecode is no longer swapped in some functions. It provides the same format in all functions.
- Support of `SV_MASTER_RAW` / `SV_MASTER_CODE` removed. Instead use sv_vtrmaster_raw().

Supported DVS OEM Products

The following DVS OEM products are supported by the DVS SDK 3.0:

DVS OEM Product	**Serial Numbers** (first two digits)
SDStationOEM	11
SDStationOEM II	19
Centaurus®	13
Centaurus II	PCI-X: 20 PCIe: 23

Centaurus II LT*	24
HydraCinema	21

* Centaurus II LT is in most respects identical to Centaurus II. Therefore it is subsumed in this reference guide under 'Centaurus II'.

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 3.2.0:

- Added descriptions for SV_OPTION_ALPHAGAIN and SV_OPTION_ALPHAOFFSET.
- Parameter *setup* documented in the description of function *sv_open()*.
- Added new mixer functionality with SV_OPTION_ALPHAMIXER.
- Added first 4K raster (see chapter Info – Video Rasters) together with SV_OPTION_DETECTION_NO4K.
- Added SV_FIFO_FLAG_DONTBLOCK to *sv_fifo_getbuffer()*.
- Added SV_OPTION_ASSIGN_LTC, SV_OPTION_ASSIGN_VTR and SV_OPTION_DETECTION_TOLERANCE.
- Added *sv_fifo_ancbuffer* parameter structure for *sv_fifo_anc()*.
- Added SV_OPTION_MULTICHANNEL and SV_OPTION_MULTICHANNEL_LOCK.
- Added *opentype* parameter SV_OPENTYPE_DEFAULT for the function *sv_openex()*.
- Added option calls SV_OPTION_HWWATCHDOG_<xxx>.
- Added descriptions for SV_OPTION_FLUSH_TIMECODE, SV_OPTION_LTCDELAY, SV_OPTION_TICK, SV_QUERY_FEATURE_AUDIOCHANNELS, SV_QUERY_PULLDOWNPHASE_DISPLAY, SV_QUERY_PULLDOWNPHASE_RECORD, SV_QUERY_RASTER_DROPFRAME, SV_QUERY_RASTER_FPS, SV_QUERY_RASTER_INTERLACE, SV_QUERY_RASTER_SEGMENTED, SV_QUERY_RASTER_XSIZE, SV_QUERY_RASTER_YSIZE, SV_QUERY_STORAGE_XSIZE, SV_QUERY_STORAGE_YSIZE, and SV_QUERY_VOLTAGE_2V3.

New in Version 3.0.2.1

- Changed notes for SV_OPTION_ANCCOMPLETE.

New in Version 3.0.2:

- Added function *sv_fifo_anclayout()*.
- Added SV_FIFO_FLAG_ANC and *anc* in *sv_fifo_buffer*.
- Added SV_OPTION_ANCCOMPLETE functionality.
- Added mixer functionality in FIFO API with SV_FIFO_FLAG_VIDEO_B. and *video_b* in *sv_fifo_buffer*.
- Added *sv_debugprint()*.
- Changed function *sv_fifo_lut()* to handle 3D LUTs.

New in Version 3.0.1:

First reference guide for the DVS SDK 3.0.

- Added SV_QUERY_AUDIO_MAXCHANNELS.
- Changed SV_QUERY_AUDIOCHANNELS.
- Updated *sv_storage_status()*.
- Added *sv_fifo_anc()*, *sv_fifo_lut()* and *sv_fifo_matrix()*.

- Updated *sv_fifo_ancdata()*.
- Added *sv_fifo_bypass()* and *sv_jack_memorysetup()*.
- Added `SV_FEATURE_INDEPENDENT_IO` and `SV_FEATURE_MULTIJACK`.
- Added *sv_jack_assign()* and *sv_jack_status()*.
- Modified description of *sv_fifo_init()* due to jack handling.
- Added `SV_IOMODE_OUTPUT_ENABLE`.
- Added *sv_fifo_sanitycheck()* and *sv_fifo_sanitylevel()*.
- Added example program *dmaspeed*.

DVS SDK Module Index

DVS SDK Modules

Here is a list of all modules:

DVS SDK Data Structure Index

DVS SDK Data Structures

Here are the data structures with brief descriptions:

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)
- 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:

```
int 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));
    return TRUE;
  }

  return FALSE;
}
```

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*)

Returns debug information captured in 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*)

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*)

Returns a string that describes the *errorcode*.

Parameters:

sv – Handle returned from the function sv_open().

errorcode – Code of the error that should be described.

Returns:

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:

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*:

- `23` – SDStationOEM and SDStationOEM II.
- `25` – Centaurus and Centaurus II.
- `26` – HydraCinema.

Returns:

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

Note:

The serial numbers of the supported DVS OEM products (first digits) are listed in the section Supported DVS OEM Products.

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

This function programs the device license.

Parameters:

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

knum – Number of the license key that will hold the license. Possible values range from zero to two (`0..2`).

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

Returns:

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

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 – Sets a timeout for a delayed opening (currently not available).

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 also internally set when the function *sv_open()* is called.
- `SV_OPENPROGRAM_SVPROGRAM` – SV program.
- `SV_OPENPROGRAM_TESTPROGRAM` – DVS test program.
- `SV_OPENPROGRAM_DEMOPROGRAM` – OEM 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` – OEM 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_RS422C` – Serial port C.
- `SV_OPENTYPE_RS422D` – Serial port D.
- `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*)

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*)

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*)

Checks if the application version matches the version numbers of library and driver. 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*)

Checks if the firmware version of the board matches the recommended version (revision). This function 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 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 and/or driver version detected.

API – Storage Functions

Detailed Description

This chapter provides a description of basic functions to control the DVS video device. These functions work on the storage and are not intended for real-time transfers to or from the system memory. The storage is the memory module on the board.

Defines

- #define SV_OPTION_FASTMODE
- #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_FASTMODE
- #define SV_QUERY_FASTMOTION
- #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_FASTMODE

This define is obsolete. It was used to set the fast motion mode, i.e. when the speed is greater one (`1`).

#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 only affects the RAM-recorder mode, not the FIFO API.

#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 only affects the RAM-recorder mode, not the FIFO API.

#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 only affects the RAM-recorder mode, not the FIFO API.

#define SV_OPTION_SPEED

This define sets the speed. The denominator is by default `0x10000` but can be set with `SV_OPTION_SPEEDBASE`.

Note:

This define only affects the RAM-recorder mode, not the FIFO API.

#define SV_OPTION_SPEEDBASE

This define sets the base denominator. The denominator is by default `0x10000`.

Note:

This define only affects the RAM-recorder mode, not the FIFO API.

#define SV_QUERY_AUDIOSIZE

This define returns the size of an audio buffer for the specified frame (for *sv_sv2host()* and *sv_host2sv()* transfers).

#define SV_QUERY_AUDIOSIZE_FROMHOST

Same as `SV_QUERY_AUDIOSIZE`.

#define SV_QUERY_AUDIOSIZE_TOHOST

Same as `SV_QUERY_AUDIOSIZE`.

#define SV_QUERY_FASTMODE

This query is obsolete. Same as `SV_QUERY_FASTMOTION`.

#define SV_QUERY_FASTMOTION

This query is obsolete. It was used to return the setting of `SV_OPTION_FASTMODE`.

#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.

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`.

See also:

The define `SV_QUERY_INTERLACEID_STORAGE`.

#define SV_QUERY_LOOPMODE

This query returns the setting of `SV_OPTION_LOOPMODE`.

#define SV_QUERY_PRESET

This query returns the preset setting. See the function sv_preset().

#define SV_QUERY_REPEATMODE

This define returns the setting of `SV_OPTION_REPEAT`.

#define SV_QUERY_SLOWMOTION

This define returns the setting of `SV_OPTION_SLOWMOTION`.

#define SV_QUERY_STREAMERSIZE

This define returns the size of one video frame.

Function Documentation

int sv_black (sv_handle * *sv*)

Sets the output to display a black frame. This function will stop any running input or output FIFO.

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>`.

int sv_colorbar (sv_handle * *sv*)

Sets the output to display a colorbar. This function will stop any running input or output FIFO.

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>`.

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 from 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), this parameter determines the position where a VTR operation will be started.

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.

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;
  }

  return SV_OK;
}
```

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_...)`.

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 function is not intended to be used together with the FIFO API.

See also:

The define SV_OPTION_REPEAT and 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.
- `SV_TYPE_HEADER` – Obsolete.

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 function is not intended to be used together with the FIFO API.

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));
      return res;
    }

    for(count = 0; count < nframes; 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;
      }
    }

    return SV_OK;
  }
```

int sv_inpoint (sv_handle * *sv*, int *frame*)

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>`.

int sv_live (sv_handle * *sv*)

This function enables the live mode (EE). It will stop any running input or output FIFO.

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>`.

See also:

The function sv_showinput().

int sv_outpoint (sv_handle * *sv*, int *frame*)

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>`.

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 (`SV_REPEAT_<xxx>`).

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 function is not intended to be used together with the FIFO API.

See also:

The define `SV_OPTION_REPEAT` and the function *sv_goto()*.

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 – Channels to record. See list below.

Parameters for *preset*:

Combine these values to enable the desired channels:

- `SV_PRESET_VIDEO` – Video 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_KEY` – Key channel.

- SV_PRESET_TIMECODE – Timecode.

Returns:

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

Note:

This function should be used when controlling a DDR only.

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), this parameter determines the position where the VTR operation will be started.

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_showinput (sv_handle * *sv*, int *showinput*, int *spare*)

This function enables the live mode (EE). It will stop any running input or output FIFO.

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_BYPASS – Obsolete. It was used for a direct bypass on previous disk recorder products by DVS.
- SV_SHOWINPUT_FRAMEBUFFERED – The live signal will be written to the memory first before sent to the output. This setting is the default one on all newer DVS video devices, such as the Centaurus or the SDStationOEM II video board.

Returns:

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

See also:

The function sv_live().

int sv_stop (sv_handle * *sv*)

Stops any running record or display operation. This function is a kind of an emergency break. It 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. This function will also stop any running input or output FIFO.

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>`.

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 function is not intended to be used together with the FIFO API.

See also:

The function sv_host2sv().

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_mixer_status (sv_handle *sv, sv_mixer_info *pinfo)
- 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_CENTAURUS` – Centaurus and Centaurus II.
- `SV_DEVTYPE_HDBOARD` – Obsolete. Device type for a previous disk recorder product by DVS.
- `SV_DEVTYPE_JPEG` – HydraCinema.
- `SV_DEVTYPE_SDBOARD` – SDStationOEM and SDStationOEM 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 OEM product (first digits) are listed in section Supported DVS OEM 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_AUTOPULLDOWN` – Obsolete.

- `SV_FEATURE_CAPTURE` – The function *sv_capture()* is supported.
- `SV_FEATURE_DUALLINK` – Dual-link operation is supported.
- `SV_FEATURE_HEADERTRANSFER` – Obsolete.
- `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_LTC_RECORDOUT` – Obsolete.
- `SV_FEATURE_LUTSUPPORT` – Loadable LUTs are supported.
- `SV_FEATURE_MIXERSUPPORT` – Mixer mode commands are supported (see e.g. the function *sv_mixer_mode()*).
- `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_NOHSWTRANSFER` – Obsolete.
- `SV_FEATURE_PLAYLISTMAP` – Obsolete.
- `SV_FEATURE_RASTERLIST` – The generation of a raster list is supported.
- `SV_FEATURE_SCSISWITCH` – Obsolete.
- `SV_FEATURE_VTRMASTER_LOCAL` – Obsolete.
- `SV_FEATURE_ZOOMANDPAN` – Support of a sub-pixel panning and zooming by any scaling factor (DVS internal only).
- `SV_FEATURE_ZOOMSUPPORT` – Integer factor zooming is supported (`1`, `2`, `4`, `8`, `16`, `32`) as well as pixel panning (see e.g. the function *sv_zoom()*).

#define SV_QUERY_FEATURE_AUDIOCHANNELS

This define returns the audio channels licensed on your DVS OEM 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. It has replaced the define `SV_QUERY_DLVERSION_PATCH`.

#define SV_QUERY_VERSION_DVSOEM

This query returns the DVSOEM library version. It has replaced the define `SV_QUERY_DLVERSION_EPROM`.

Function Documentation

int sv_mixer_status (sv_handle * *sv*, sv_mixer_info * *pinfo*)

This function returns information about the current mixer settings in the structure *sv_mixer_info*.

Parameters:

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

pinfo – Pointer to the structure *sv_mixer_info*.

Returns:

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

Note:

This mixer functionality is available on the SDStationOEM and SDStationOEM II only.

See also:

The functions *sv_mixer_input()* and *sv_mixer_mode()*.

int sv_raster_status (sv_handle * *sv*, int *rasterid*, sv_rasterheader * *raster*, int *rastersize*, int * *nrasters*, int *spare*)

This function returns information about the current video raster settings of the DVS video device as well as about all available rasters and their index numbers in the raster table, and fills these into the structure *sv_rasterheader*.

Parameters:

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

rasterid – If this parameter is set to the value -1, the function returns the number of available rasters. In case this parameter contains a raster identifier, the function fills the information of this raster into the structure *sv_rasterheader*.

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>.

Example:

```
int example_dumprasters(sv_handle * sv)
{
  int res;

  res = sv_raster_status(sv, -1, &current, sizeof(current), &nrasters, 0);

  for(i = 0; (res == SV_OK) && (i < nrasters); i++) {
    res = sv_raster_status(sv, i, &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 the inpoint and the outpoint) are only updated on a frame basis on the video device. Read-backs of these parameters are possible only on a frame basis and will be retrieved from the last frame that was processed completely.

Parameters:

sv – Handle returned from the function sv_open().

info – Handle *sv_info*.

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, devicecount;
    int module, modulecount;
    char tmp[64];

    devicecount = 1;
    modulecount = 1;

    for(device = 0; (res == SV_OK) && (device < devicecount); device++) {
      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) {
            if(device == 0) {
              devicecount = version.devicecount;
            }
            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 – Control Functions

Detailed Description

This chapter describes various control functions to control, for example, zooming and panning or pulldown.

Defines

- #define SV_OPTION_GAMMA
- #define SV_OPTION_GAMMA_BLUE
- #define SV_OPTION_GAMMA_GREEN
- #define SV_OPTION_GAMMA_RED
- #define SV_OPTION_OUTDURINGREC
- #define SV_OPTION_RS422A_PINOUT
- #define SV_OPTION_RS422B_PINOUT
- #define SV_OPTION_RS422C_PINOUT
- #define SV_OPTION_RS422D_PINOUT
- #define SV_QUERY_GAMMA
- #define SV_QUERY_GAMMA_BLUE
- #define SV_QUERY_GAMMA_GREEN
- #define SV_QUERY_GAMMA_RED
- #define SV_QUERY_OUTDURINGREC
- #define SV_QUERY_PULLDOWN
- #define SV_QUERY_PULLDOWNPHASE
- #define SV_QUERY_PULLDOWNPHASE_DISPLAY
- #define SV_QUERY_PULLDOWNPHASE_RECORD
- #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 spare)
- 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_mixer_input (sv_handle *sv, int porta, int portb, int portc, int spare)
- int sv_mixer_mode (sv_handle *sv, int mode, int param, int start, int end, int nframes, int spare)
- int sv_zoom (sv_handle *sv, int xzoom, int yzoom, int xpanning, int ypanning, int flags)

Define Documentation

#define SV_OPTION_GAMMA

This define sets a gamma value for the luma channel of the analog output by setting the hardware-implemented look-up table (LUT). This is a fixed point float value, `value = (int)(doublevalue * 0x10000)`. To set different gamma values for each RGB component use the further parameters `RED`, `GREEN`, `BLUE` for `SV_OPTION_GAMMA` (see, for example, `SV_OPTION_GAMMA_RED`).

#define SV_OPTION_GAMMA_BLUE

This define sets gamma for the blue channel. Affects only the analog output. Applicable in RGB video modes only. This is a fixed point float value, `value = (int)(doublevalue * 0x10000)`.

#define SV_OPTION_GAMMA_GREEN

This define sets gamma for the green channel. Affects only the analog output. Applicable in RGB video modes only. This is a fixed point float value, `value = (int)(doublevalue * 0x10000)`.

#define SV_OPTION_GAMMA_RED

This define sets a gamma value for the red channel. Affects only the analog output. Applicable in RGB video modes only. This is a fixed point float value, `value = (int)(doublevalue * 0x10000)`.

#define SV_OPTION_OUTDURINGREC

This define is obsolete. To determine what should be given out during record you have to use an output FIFO.

#define SV_OPTION_RS422A_PINOUT

This option call independently configures the physical pin-out and the logical task 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.
- `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.
- `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.

Note:

Most video boards by DVS support a changing of the physical pin-out of this port. Boards where this feature is not supported will return `SV_ERROR_WRONG_HARDWARE`. In the above explanations a reverse pin-out means that pins 2/8 and 3/7 are swapped.

When the port has been already opened by the function sv_rs422_open(), this option call will return SV_ERROR_ALREADY_OPENED.

When using multiple calls of SV_OPTION_RS422[A, B, C, D]_PINOUT, only one master task 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.

#define SV_OPTION_RS422B_PINOUT

This option call independently configures the physical pin-out and the logical task of the RS-422 port B.

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.
- SV_RS422_TASK_DEFAULT – Sets this port to its default task setting which is SV_RS422_TASK_SLAVE.
- SV_RS422_TASK_NONE – No task is assigned to this port.
- 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.

Note:

Most video boards by DVS support a changing of the physical pin-out of this port. Boards where this feature is not supported will return SV_ERROR_WRONG_HARDWARE. In the above explanations a reverse pin-out means that pins 2/8 and 3/7 are swapped.

When the port has been already opened by the function sv_rs422_open(), this option call will return SV_ERROR_ALREADY_OPENED.

When using multiple calls of SV_OPTION_RS422[A, B, C, D]_PINOUT, only one master task 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.

#define SV_OPTION_RS422C_PINOUT

This option call independently configures the physical pin-out and the logical task of the RS-422 port C.

For a complete parameter list please refer to SV_OPTION_RS422A_PINOUT. Only the following two parameters differ from their meaning given under SV_OPTION_RS422A_PINOUT:

- 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_NONE.

Note:

The RS-422 ports C and D are not available for all DVS OEM products. Please check the features of your DVS video device whether they are present.

For further notes please refer to SV_OPTION_RS422A_PINOUT.

#define SV_OPTION_RS422D_PINOUT

This option call independently configures the physical pin-out and the logical task of the RS-422 port D.

For a complete parameter list please refer to SV_OPTION_RS422A_PINOUT. Only the following two parameters differ from their meaning given under SV_OPTION_RS422A_PINOUT:

- 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_NONE.

Note:

The RS-422 ports C and D are not available for all DVS OEM products. Please check the features of your DVS video device whether they are present.

For further notes please refer to SV_OPTION_RS422A_PINOUT.

#define SV_QUERY_GAMMA

This query returns the current setting of luma gamma. See the define SV_OPTION_GAMMA.

#define SV_QUERY_GAMMA_BLUE

This define returns the current setting of blue gamma. See the define SV_OPTION_GAMMA_BLUE.

#define SV_QUERY_GAMMA_GREEN

This query returns the current setting of green gamma. See the define SV_OPTION_GAMMA_GREEN.

#define SV_QUERY_GAMMA_RED

This define returns the current setting of red gamma. See the define SV_OPTION_GAMMA_RED.

#define SV_QUERY_OUTDURINGREC

This define is obsolete. For further information see the define SV_OPTION_OUTDURINGREC.

#define SV_QUERY_PULLDOWN

This define returns the pulldown phase of the material currently recorded or played out.

#define SV_QUERY_PULLDOWNPHASE

This define returns the pulldown phase currently detected at the input of the device.

#define SV_QUERY_PULLDOWNPHASE_DISPLAY

This define returns the current pulldown phase during a display (output).

#define SV_QUERY_PULLDOWNPHASE_RECORD

This define returns the current pulldown phase during a record (input).

#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 *spare*)

This function programs the look-up table for a color correction at the output of the DVS video device.

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.

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_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 – Matrix mode. 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.
- `SV_MATRIX_CCIR601CGR` – RGB full to SD YUV head.
- `SV_MATRIX_CCIR601INV` – RGB head to SD YUV full.
- `SV_MATRIX_SMPTE274` – RGB to HD YUV.
- `SV_MATRIX_SMPTE274CGR` – RGB full to HD YUV head.
- `SV_MATRIX_SMPTE274INV` – RGB head to HD YUV full.
- `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_NOFILTER` – Obsolete. It was used for a previous disk recorder product by DVS.
- `SV_INPUTFILTER_5TAPS` – Interpolation or decimation filter width.
- `SV_INPUTFILTER_9TAPS` – Interpolation or decimation filter width.
- `SV_INPUTFILTER_13TAPS` – Interpolation or decimation filter width.
- `SV_INPUTFILTER_17TAPS` – 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_NOFILTER` – Obsolete. It was used for a previous disk recorder product by DVS.
- `SV_OUTPUTFILTER_5TAPS` – Interpolation or decimation filter width.
- `SV_OUTPUTFILTER_9TAPS` – Interpolation or decimation filter width.
- `SV_OUTPUTFILTER_13TAPS` – Interpolation or decimation filter width.
- `SV_OUTPUTFILTER_17TAPS` – 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 function *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 – Matrix mode. See list below.

pmatrix – Pointer to the structure *sv_matrixexinfo*.

pquery – Returns the current matrix settings (can be `NULL`).

Parameters for *matrixtype*:

- `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.
- `SV_MATRIX_CCIR601CGR` – RGB full to SD YUV head.
- `SV_MATRIX_CCIR601INV` – RGB head to SD YUV full.
- `SV_MATRIX_SMPTE274` – RGB to HD YUV.
- `SV_MATRIX_SMPTE274CGR` – RGB full to HD YUV head.
- `SV_MATRIX_SMPTE274INV` – RGB head to HD YUV full.
- `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 *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 SV_OPTION_INPUTFILTER or SV_OPTION_OUTPUTFILTER respectively.

See also:

The function *sv_matrix()*.

int sv_mixer_input (sv_handle * *sv*, int *porta*, int *portb*, int *portc*, int *spare*)

This function controls the input ports for a mixer operation.

Parameters:

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

porta – Mixer input port A. First video channel.

portb – Mixer input port B. Second video channel.

portc – Mixer control input port.

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:

Possible values for *porta*, *portb* and *portc* are SV_MIXER_INPUT_<xxx>.

This mixer functionality is available on the SDStationOEM and SDStationOEM II only.

See also:

The functions *sv_mixer_mode()* and *sv_mixer_status()*.

int sv_mixer_mode (sv_handle * *sv*, int *mode*, int *param*, int *start*, int *end*, int *nframes*, int *spare*)

This function selects the mixer operation mode. With it you can mix the incoming video signal with the material present in the video buffer. A mixer operation mode is active until either the mixer operation mode or the video mode is changed. This function influences both the analog as well as the digital outputs.

Parameters:

sv – Handle returned from the function sv_open().

mode – Sets the mixer operation mode (`SV_MIXER_MODE_<xxx>`, see the file `dvs_clib.h` for all possible defines).

param – Adds a further specification to the parameter *mode*.

start – Frame number of the frame where the mixer operation should start.

end – Frame number of the frame where the mixer operation should end.

nframes – Number of frames that the mixer operation should last. If *nframes* is specified, the mixer operation moves from start to end over *nframes* frames.

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:

This mixer functionality is available on the SDStationOEM and SDStationOEM II only.

See also:

The functions sv_mixer_input() and sv_mixer_status().

int sv_zoom (sv_handle * *sv*, int *xzoom*, int *yzoom*, int *xpanning*, int *ypanning*, int *flags*)

This function performs a zooming and panning.

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`.
- `SV_ZOOMFLAGS_PIXELREPETITION` – Forces a pixel repetition even if the hardware supports a better filtering.
- `SV_ZOOMFLAGS_BILINEAR` – Forces a bilinear filter even if the hardware supports a better filtering.

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 not supported on the SDStationOEM and SDStationOEM II.

On Centaurus and Centaurus II it can be used without limitations.

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).

Defines

- #define SV_OPTION_TICK

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)

Define Documentation

#define SV_OPTION_TICK

DVS internal define. Needed by the function *sv_option_setat()*.

Function Documentation

int sv_option (sv_handle * *sv*, int *code*, int *value*)

Sets an `SV_OPTION_<xxx>` value (same as *sv_option_set()*).

Parameters:

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

code – Code for `SV_OPTION_<xxx>`.

value – Value to set.

Returns:

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

Note:

For possible `SV_OPTION_<xxx>` control codes please see the appropriate chapters in this reference guide.

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 – Code for `SV_OPTION_<xxx>`.

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>`.

Note:

For possible `SV_OPTION_<xxx>` control codes please see the appropriate chapters in this reference guide.

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 (same as sv_option()).

Parameters:

sv – Handle returned from the function sv_open().

option – Code for SV_OPTION_<xxx>.

value – Value to set.

Returns:

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

Note:

For possible SV_OPTION_<xxx> control codes please see the appropriate chapters in this reference guide.

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 – Code for `SV_OPTION_<xxx>`.

value – Value to set.

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 – Control code for `SV_QUERY_<xxx>`.

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>`.

Note:

For possible `SV_QUERY_<xxx>` control codes please see the appropriate chapters in this reference guide.

API – VTR 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).

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).

#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>`.

Examples:

```
0:0:0:0  -> 00:00:00:00
0:0:0.0  -> 00:00:00.00
1:23:4   -> 00:01:23:04
-1       -> 00:00:00:01
9        -> 00:00:00:09
99       -> 00:00:00:99
0        -> 00:00:00:00
101      -> 00:00:01:01
12345678 -> 12:34:56:78
-22      -> 00:00:00:22
```

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>.

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("Current Time Code: %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 stops 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 section 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 previously set `SV_MASTER_INPOINT`, `SV_MASTER_OUTPOINT` or `SV_MASTER_NFRAMES`.

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. This function takes symbolic command defines. See *sv_vtrmaster_raw()* which can be used to work with raw RS-422 commands.

Parameters:

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

opcode – Any `SV_MASTER_<xxx>` code. 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.
- 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 SV_OPTION_VTRMASTER_TCTYPE.
- SV_MASTER_TOLERANCE – Sets the edit tolerance of the VTR master. It affects eventual timeouts. For possible types see SV_OPTION_VTRMASTER_TOLERANCE.
- SV_MASTER_EDITFIELD_START – Sets the start field for the editing. Default is the first field (value zero (0)).
- SV_MASTER_EDITFIELD_END – Sets the end field for the editing. Default is the last field (value one (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 – not supported on all VTRs). The value is the number of frames to step.
- 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>.

Note:

For 64-bit compatibility use *sv_vtrmaster_raw()* instead of SV_MASTER_CODE / SV_MASTER_RAW.

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. This function 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 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));
      return res;
    }

    // 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;
    }

    return SV_OK;
  }
```

API – Audio Functions

Detailed Description

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

Defines

- #define SV_OPTION_AUDIOANALOGOUT
- #define SV_OPTION_AUDIOBITS
- #define SV_OPTION_AUDIOCHANNELS
- #define SV_OPTION_AUDIOFREQ
- #define SV_OPTION_AUDIOINPUT
- #define SV_OPTION_AUDIOMAXAIV
- #define SV_OPTION_AUDIOMUTE
- #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_AUDIOSIZE_TICK
- #define SV_QUERY_WORDCLOCK

Define Documentation

#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:

- `16` – 16 bit audio.
- `32` – 32 bit audio.

Note:

Centaurus and Centaurus II support 32-bit audio only. A bit depth of 16 bit is not supported on these boards.

#define SV_OPTION_AUDIOCHANNELS

This define sets the number of audio channels.

#define SV_OPTION_AUDIOFREQ

This define sets the audio frequency:

- `48000` – Default is 48000 Hz.

#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.

#define SV_OPTION_AUDIOMAXAIV

This define sets the maximum number of embedded audio channels. This is 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 inputs can be selected:

- `0` – No embedded audio.
- `4` – Four mono channels, one group.
- `8` – Eight mono channels, two 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_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 audio channels that can be configured with `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 audio channels. 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_AUDIOSIZE_TICK

This define returns the number of audio samples referring to the specified tick value.

#define SV_QUERY_WORDCLOCK

This define returns the current setting of wordclock. 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_ANALOG
- #define SV_OPTION_DETECTION_NO4K
- #define SV_OPTION_DETECTION_TOLERANCE
- #define SV_OPTION_DISABLESWITCHINGLINE
- #define SV_OPTION_FIELD_DOMINANCE
- #define SV_OPTION_HDELAY
- #define SV_OPTION_HWWATCHDOG_ACTION
- #define SV_OPTION_HWWATCHDOG_REFRESH
- #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_OUTPUTDVI
- #define SV_OPTION_OUTPUTFILTER
- #define SV_OPTION_OUTPUTPORT
- #define SV_OPTION_OVERLAY_MODE
- #define SV_OPTION_SYNCMODE
- #define SV_OPTION_SYNCOUT
- #define SV_OPTION_SYNCOUTDELAY
- #define SV_OPTION_SYNCOUTVDELAY
- #define SV_OPTION_VDELAY
- #define SV_OPTION_VIDEOMODE
- #define SV_OPTION_ZEROLSB
- #define SV_QUERY_ANALOG
- #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_IOMODE
- #define SV_QUERY_MODE_AVAILABLE
- #define SV_QUERY_MODE_CURRENT
- #define SV_QUERY_OUTPUTFILTER
- #define SV_QUERY_OUTPUTPORT
- #define SV_QUERY_OVERLAY
- #define SV_QUERY_OVERLAYMODE
- #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_SDILINKDELAY
- #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_overlaytext (sv_handle *sv, int xpos, int ypos, char *pbuffer, int count)
- 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 additionally with the define `SV_OPTION_ALPHAOFFSET`. This 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()*.

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 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.

The alpha mixer is only available on Centaurus II.

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 the 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()*.

See also:

The defines SV_OPTION_ALPHAMIXER and SV_OPTION_ALPHAGAIN.

#define SV_OPTION_ANALOG

This define configures the analog video output(s) of the board:

- SV_ANALOG_BLACKLEVEL_MASK – Mask for the blacklevel. It is valid for NTSC CVBS/SVHS only. The following settings are possible:
  - SV_ANALOG_BLACKLEVEL_BLACK75 – NTSC USA, i.e. black level is 7.5 IRE.
  - SV_ANALOG_BLACKLEVEL_BLACK0 – NTSC Japan, i.e. black level is 0 IRE.
- SV_ANALOG_FORCE_MASK – Mask to force an NTSC color carrier in PAL and vice versa. The following settings are possible:
  - SV_ANALOG_FORCE_NONE – Switches the color carrier according to the selected video mode (default setting).
  - SV_ANALOG_FORCE_PAL – Forces the color carrier to PAL independent of the video mode.
  - SV_ANALOG_FORCE_NTSC – Forces the color carrier to NTSC independent of the video mode.
- SV_ANALOG_OUTPUT_MASK – Mask for the analog output color mode. The following settings are possible:
  - SV_ANALOG_OUTPUT_YC – Y/C or SVHS.
  - SV_ANALOG_OUTPUT_YUV – YUV with sync on Y.
  - SV_ANALOG_OUTPUT_RGB – RGB with sync on green.
  - SV_ANALOG_OUTPUT_YUVS – YUV with separate sync.
  - SV_ANALOG_OUTPUT_RGBS – RGB with separate sync.
  - SV_ANALOG_OUTPUT_CVBS – CVBS.

Note:

This define has replaced the define SV_OPTION_SVHS.

#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 or 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.

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 (all firmwares) and Centaurus II (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 at least <x>.2.68.8_11_4.

#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_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.

Note:

This define has replaced the define `SV_OPTION_SOFDELAY`.

#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`.

The hardware watchdog is only available on Centaurus II.

See also:

The defines SV_OPTION_HWWATCHDOG_REFRESH, 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). This call should be used in combination with `SV_HWWATCHDOG_MANUAL` only.

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 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`)).

As long a the power is 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. Then the driver will 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 set 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`.

#define SV_OPTION_INPUTFILTER

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

#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_ANALOG – Selects the analog input.
- SV_INPUTPORT_SDI2 – Selects the second SDI input, i.e. link B of the SDI will be used.
- SV_INPUTPORT_DVI – Selects the DVI input.
- 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 and Centaurus II the call for SV_INPUTPORT_SDI2 uses the explicitly designated (HD) link B input of the board (with Centaurus it can be found on the mounted module). 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. If no output I/O mode is set, both input and output will use the same video I/O mode.

- SV_IOMODE_YUV422 – YUV 4:2:2.
- SV_IOMODE_RGB – RGB 4:4:4.
- SV_IOMODE_YUV444 – YUV 4:4:4.
- SV_IOMODE_YUV422A – YUVA 4:2:2:4.
- SV_IOMODE_RGBA – RGBA 4:4:4:4.
- SV_IOMODE_YUV444A – YUVA 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_IO_MASK – Mask for the I/O mode (normal).
- SV_IOMODE_OUTPUT_MASK – Mask for the I/O mode (output).
- SV_IOMODE_OUTPUT2MODE(x) – Converts the 'output' I/O mode to 'normal'.
- SV_IOMODE_OUTPUT_YUV422 – YUV 4:2:2.
- SV_IOMODE_OUTPUT_RGB – RGB 4:4:4.
- SV_IOMODE_OUTPUT_YUV444 – YUV 4:4:4.
- SV_IOMODE_OUTPUT_YUV422A – YUVA 4:2:2:4.
- SV_IOMODE_OUTPUT_RGBA – RGBA 4:4:4:4.
- SV_IOMODE_OUTPUT_YUV444A – YUVA 4:4:4:4.
- SV_IOMODE_OUTPUT_YUV422_12 – YUV 4:2:2 (12 bit).
- SV_IOMODE_OUTPUT_YUV444_12 – YUV 4:4:4 (12 bit).
- SV_IOMODE_OUTPUT_RGB_12 – RGB 4:4:4 (12 bit).
- SV_IOMODE_OUTPUT_ENABLE – If you use SV_IOMODE_OUTPUT_<xxx>, you have to set this flag, too.

#define SV_OPTION_OUTPUTDVI

Currently this define provides no functionality.

#define SV_OPTION_OUTPUTFILTER

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

#define SV_OPTION_OUTPUTPORT

This define configures the video output. Some DVS video devices provide additional 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_SWAPPED` – Output ports A and B are swapped (SDStationOEM and SDStationOEM II only): For example, with YUVA the A port gives out the key data, while the B port gives out the video data.
- `SV_OUTPUTPORT_MIRROR` – Output ports A and B show both the output signal of port A (Centaurus and Centaurus II only).

#define SV_OPTION_OVERLAY_MODE

This define enables or disables the video board's overlay given out via the analog outputs. Additionally, you can give out the overlay through the digital outputs.

- `SV_OVERLAY_OFF` – Hide overlay.
- `SV_OVERLAY_ON` – Show overlay.
- `SV_OVERLAY_DIGITAL` – Show overlay (on all outputs).

Note:

This feature is available on the SDStationOEM and SDStationOEM II only.

This define has replaced the define `SV_OPTION_OVERLAY_ENAB`.

See also:

The function *sv_overlaytext()*.

#define SV_OPTION_SYNCMODE

This define sets the sync mode. See the function *sv_sync()* for parameters.

#define SV_OPTION_SYNCOUT

This define sets the video sync output.

Values:

- `SV_SYNCOUT_OFF` – No sync output signal.
- `SV_SYNCOUT_BILEVEL` – Bilevel sync output.
- `SV_SYNCOUT_TRILEVEL` – Trilevel sync output.
- `SV_SYNCOUT_HVTTL_HFVF` – HVTTL with falling H and falling V.
- `SV_SYNCOUT_HVTTL_HFVR` – HVTTL with falling H and rising V.
- `SV_SYNCOUT_HVTTL_HRVF` – HVTTL with rising H and falling V.
- `SV_SYNCOUT_HVTTL_HRVR` – HVTTL with rising H and rising V.
- `SV_SYNCOUT_DEFAULT` – Raster default sync output mode.

Masks:

- `SV_SYNCOUT_OUTPUT_GREEN` – Sets the sync on green.
- `SV_SYNCOUT_LEVEL_SET(<x>)` – Sets the voltage level. Can be set to either 0.7 V or 4.0 V.

#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.

Note:

Not available on the SDStationOEM and SDStationOEM II.

#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.

Note:

Not available on the SDStationOEM and SDStationOEM II.

#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. It performs the same operation as the function *sv_videomode()*.

#define SV_OPTION_ZEROLSB

This setting makes sure that the video bits in the SDI stream are rounded to 8-bit data. When operating with 8-bit data in the storage, this is not necessarily the case as color space conversions affect all bits.

Note:

This setting only shows an effect after a subsequent call to the function *sv_videomode()*.

#define SV_QUERY_ANALOG

This define returns the setting of `SV_OPTION_ANALOG`. It has replaced the define `SV_QUERY_SVHS`.

#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 current raster available on the SDI input.

#define SV_QUERY_INPUTRASTER_GENLOCK

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

Note:

Not available for the SDStationOEM and SDStationOEM II.

#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.

Note:

Not available for the SDStationOEM and SDStationOEM II.

#define SV_QUERY_INPUTRASTER_SDIA

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

#define SV_QUERY_INPUTRASTER_SDIB

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

#define SV_QUERY_INPUTRASTER_SDIC

This query returns the raster of the signal connected to the SDI input channel C (DVS internal only).

#define SV_QUERY_IOMODE

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

#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 mode (`SV_MODE_<xxx>`, see the file *`dvs_clib.h`* for all possible defines).

#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_OVERLAY

This define will return `TRUE` when the overlay feature is supported.

#define SV_QUERY_OVERLAYMODE

This define returns the current overlay mode. See the define `SV_OPTION_OVERLAY_MODE`.

#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_SDILINKDELAY

This define returns the delay between SDI link A and B (DVS internal 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 function *sv_sync()*.

#define SV_QUERY_SYNCOUT

This query returns the setting of the sync output. See 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_overlaytext (sv_handle * *sv*, int *xpos*, int *ypos*, char * *pbuffer*, int *count*)

This function writes text to the DVS video board's overlay. It can be enabled/disabled with the SV_OPTION_OVERLAY_MODE parameter of the function *sv_option()*. The overlay has a maximum size of three lines with 24 characters each.

Parameters:

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

xpos – Vertical start position of the text that is kept in the parameter buffer. Possible values range from zero to two (0..2).

ypos – Horizontal start position of the text that is kept in the parameter buffer. Possible values range from zero to 23 (0..23).

pbuffer – This parameter holds the text that should be displayed at the position *xpos* and *ypos*.

count – This parameter defines the overlay size. The complete overlay can be written in one call if *count* equals 24*3 and *xpos* as well as *ypos* equal zero (0). Otherwise only one line at a time is written.

Returns:

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

Note:

Available on the SDStationOEM and SDStationOEM II only.

You can clear the overlay by setting *xpos*, *ypos* and *count* to zero (0), and *pbuffer* to NULL.

Example:

```
  int example_overlaytext(sv_handle * sv)
  {
    int res;

    // Clear overlay
    res = sv_overlaytext (sv, 0, 0, NULL, 0);
    if(res != SV_OK) {
      printf("overlaytextexample: sv_overlaytext() clear failed = %d '%s'", res,
sv_geterrortext(res));
      return FALSE;
    }

    // Write text to overlay
    res = sv_overlaytext (sv, 0, 0, "Hello World", 24*3);
    if(res != SV_OK) {
      printf("error: sv_overlaytext() failed = %d '%s'", res,
sv_geterrortext(res));
      return FALSE;
    }
    return TRUE;
  }
```

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 selects the sync mode of the DVS video device. This defines how the output raster generator shall be clocked and synchronized.

Parameters:

sv – Handle returned from the function sv_open().

sync – `SV_SYNC_<xxx>`. 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. Use bilevel or 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 (not for the SDStationOEM and SDStationOEM II) and uses the reference input signal. Usually used for HD rasters.

- SV_SYNC_HVTTL – Sets a separate H and V sync mode (not for the SDStationOEM and SDStationOEM II). 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_NTSC – Obsolete. Instead use SV_SYNC_FLAG_SDTV.
- 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).

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 TRUE;
   }

   return FALSE;
 }
```

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

This function configures the sync output.

Parameters:

sv – Handle returned from the function sv_open().

syncout – SV_SYNCOUT_<xxx>. See list below.

Parameters for *syncout*:

- SV_SYNCOUT_OFF – Sync output disabled (0 V).
- SV_SYNCOUT_BILEVEL – Bilevel sync (default is 0.7 V).
- SV_SYNCOUT_TRILEVEL – Trilevel sync (default is 0.7 V).
- SV_SYNCOUT_HVTTL – H and V sync (default is 4.0 V). Combine with one of the following TTL signal forms:
  - SV_SYNCOUT_HVTTL_HFVF – TTL signal, horizontal falling, vertical falling.
  - SV_SYNCOUT_HVTTL_HRVF – TTL signal, horizontal rising, vertical falling.
  - SV_SYNCOUT_HVTTL_HFVR – TTL signal, horizontal falling, vertical rising.
  - SV_SYNCOUT_HVTTL_HRVR – TTL signal, horizontal rising, vertical rising.
- SV_SYNCOUT_USERDEF – User-defined sync. Only available with customized video rasters.
- SV_SYNCOUT_DEFAULT – Uses the raster module's default sync.

Parameters for *syncout* (Flags):

- SV_SYNCOUT_OUTPUT_GREEN – Enables a sync on green for bilevel and trilevel sync outputs.

Returns:

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

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.

Parameters:

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

mode – SV_MODE_<xxx>. 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 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.

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 the OEM 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 over an input video signal, 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. The SDStationOEM and SDStationOEM II require an alignment of 8 bytes, while other DVS hardware may even have a higher alignment. You can query the minimum needed alignment for DMA transfers for your DVS OEM 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 OEM 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.

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 3.0 (see section Supported DVS OEM Products) the global clock is a hardware based microsecond counter.

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, use either a DDR device or 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

- #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_NODMA
- #define SV_FIFO_FLAG_NODMAADDR
- #define SV_FIFO_FLAG_PHYSADDR
- #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_FIFO_FLAG_VSYNCWAIT_RT
- #define SV_OPTION_DROPMODE
- #define SV_OPTION_WATCHDOG_ACTION
- #define SV_OPTION_WATCHDOG_TIMEOUT

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 bdma, 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

sv_fifo

Void handle to the internal structure describing the FIFO.

#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

Defines that all audio channels are stored multiplexed in one buffer, and not in stereo pairs. This is the native format for newer boards by DVS such as the Centaurus video board. It is not supported by the SDStationOEM and SDStationOEM II.

When setting this flag, all audio data will be accessible by *sv_fifo_buffer.audio[0].addr[0]* directly. The value of *sv_fifo_buffer.audio[0].size* has to be adjusted to a multiple of the initial value by the caller.

If this flag is not set on newer DVS boards, there will be a copy done by the CPU to reformat the audio data.

#define SV_FIFO_FLAG_AUDIOONLY

Selects a transfer of audio only (DMA FIFO).

#define SV_FIFO_FLAG_CLOCKEDOPERATION

Performs the same operation as `SV_FIFO_FLAG_TIMEDOPERATION`, except that the clock is used instead of the vertical sync timestamp.

#define SV_FIFO_FLAG_DMARECTANGLE

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 at *sv_fifo_getbuffer()*. But because a capturing starts before the first *sv_fifo_getbuffer()* / *sv_fifo_putbuffer()* 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

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_NODMA

Disables and skips the DMA transfer for the current frame (DMA FIFO).

#define SV_FIFO_FLAG_NODMAADDR

The entries in *pbuffer->dma.addr* / *pbuffer->dma.size* are not used. Instead the addresses in the video/audio substructure will be used.

#define SV_FIFO_FLAG_PHYSADDR

Currently not used.

#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()* is performed.

Note:

This flag overrules the define `SV_FIFO_FLAG_FIELD`.

#define SV_FIFO_FLAG_REPEAT_2TIMES

For further information see the define `SV_FIFO_FLAG_REPEAT_ONCE`.

#define SV_FIFO_FLAG_REPEAT_3TIMES

For further information see the define `SV_FIFO_FLAG_REPEAT_ONCE`.

#define SV_FIFO_FLAG_REPEAT_4TIMES

For further information 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>`.

#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 look at the *`dpxio`* 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

Changes the audio buffer size for an output FIFO. With this flag it is possible to modify the audio buffer sizes slightly which will be useful, for example, when playing out audio material with a different audio data distribution. This 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 `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.

#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 either in 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 `SV_OPTION_ALPHAMIXER` for further information.

This functionality is available for Centaurus II only.

#define SV_FIFO_FLAG_VIDEOONLY

Selects a transfer of video only (DMA FIFO).

#define SV_FIFO_FLAG_VSYNCWAIT

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. As an alternative use *sv_fifo_vsyncwait()* in combination with the functions *sv_fifo_getbuffer()* and *sv_fifo_putbuffer()*.

#define SV_FIFO_FLAG_VSYNCWAIT_RT

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. As an alternative, use *sv_fifo_vsyncwait()* in combination with the functions *sv_fifo_getbuffer()* and *sv_fifo_putbuffer()*.

#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 colorbar image.

See also:

The defines `SV_OPTION_DROPMODE` and `SV_OPTION_WATCHDOG_TIMEOUT`.

#define SV_OPTION_WATCHDOG_TIMEOUT

Defines a timeout after which the watchdog should react. The unit is in ticks.

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 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().

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 was too small.

Note:

This function is not available for the SDStationOEM and SDStationOEM II.

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 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 get 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 only applicable on an output FIFO. The corresponding input FIFO needs to be in running state. The input and output jacks need to be configured to the same raster and storage mode settings.

On the SDStationOEM and SDStationOEM II it is only possible to bypass all audio channels at once.

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

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*)

Specifies a DMA scatter/gather operation that enables an image cut-out of 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:

The flag `SV_FIFO_FLAG_DMARECTANGLE` must be used with 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. 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 those on the video board) are added at the end of the scan line. So the video is not centered but left aligned if gaps exist.

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

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 the pointer values indicate the current offsets from the start of the DMA buffer.

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 *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 *sv_fifo_putbuffer()*.
- `SV_ERROR_NODATA` – Will be returned when no buffer is recorded and only when the flag `SV_FIFO_FLAG_DONTBLOCK` is used.

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

This function initializes the FIFO for in- or output video operations. Returns a *pfifo* handle to be used in subsequent 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. If *bshared* is set, the same memory will be used for an input and output.

Parameters:

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

ppfifo – Pointer to the returned FIFO handle.

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. In the DVS SDK 3.0 one can address more than the two default jacks (see chapter API – Jack API).

bshared – For separate FIFOs this has to be set to FALSE, while for a shared storage it must be TRUE. This parameter is ignored for any user-defined jack (except the two default output and input FIFOs).

bdma – Zero (0) for a mapped FIFO (obsolete), one (1) for a DMA FIFO, two (2) for a DMA FIFO without automatic DMA.

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

nframes – Maximum number of frames in the FIFO. Zero (0) means the maximum that is possible (depending on the buffer size).

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 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. Each component providing a length of 16 bit results in a 29478-bytes LUT buffer. For performance reasons, the function always expects a 32768-bytes LUT buffer.

In any case, the native value range is always zero to 65535 (0..65535).

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

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).
- `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.

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

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.

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 version parameter 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 *sv_fifo_sanitycheck()* before calling these functions.

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 respectively.

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 when already started.

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. May be used to avoid blocking.

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>` with, for example:

- `SV_ERROR_FIFO_PUTBUFFER` – Will be returned when called while an *sv_fifo_getbuffer()* operation is still pending.

Note:

This function may not be called when a pending *sv_fifo_getbuffer()* operation is still open.

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_FIFOF_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>`. For further details about possible error codes see the function *sv_fifo_stop()*.

Note:

This function may not be called when a pending *sv_fifo_getbuffer()* operation is still open.

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 on the input or output and returns the control to the program flow after the sync has occurred.

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 would have 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. Under Windows the overlapped parameter is the normal Windows overlapped parameter that allows overlapped I/O transfers. If this is set to `NULL`, a normal synchronous DMA transfer is done, otherwise the transfers 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 *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 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. Under Windows the overlapped parameter is the normal Windows overlapped parameter that allows overlapped I/O transfers. If this is set to `NULL`, a normal synchronous DMA transfer is done, otherwise the transfers 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. Under Windows the overlapped parameter is the normal Windows overlapped parameter that allows overlapped I/O transfers. If this is set to `NULL`, a normal synchronous DMA transfer is done, otherwise the transfers 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 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. Under Windows the overlapped parameter is the normal Windows overlapped parameter that allows overlapped I/O transfers. If this is set to `NULL`, a normal synchronous DMA transfer is done, otherwise the transfers 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 has to be selected by inserting `SV_PRESET_<xxx>` codes into the *channel* parameter.

Parameters:

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

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

channel – Sets the active channels. For possible codes see list below.

field1addr – Offset (memory address) of the beginning of field 1.

field1size – Size of the active video area of field 1.

field2addr – Offset (memory address) of the beginning of field 2.

field2size – Size of the active video area of field 2.

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 – 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 OEM product. As a rule older OEM 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 OEM 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 OEM Product	Input Pipeline	Output Pipeline	Independent I/O
SDStationOEM	1 single link or 1 dual link	1 single link or 1 dual link	No
SDStationOEM II	1 single link or 1 dual link	1 single link or 1 dual link	No
Centaurus	1 single link or 1 dual link	1 single link or 1 dual link	Yes
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	No
HydraCinema	–	2 single link or 2 dual link	Yes

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

Please note that the *sv_jack_assign()* and *sv_jack_find()* functions mentioned in this chapter are obsolete and should not be used anymore. The DVS SDK uses static assignments of jack IDs/indexes to jack names instead (see below).

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 dual-link output (jack ID 0)
  res = sv_jack_option_set(sv, 0, SV_OPTION_VIDEOMODE, SV_MODE_PAL |
SV_MODE_COLOR_RGBA);
  res = sv_jack_option_set(sv, 0, SV_OPTION_IOMODE, SV_IOMODE_RGBA);
  // 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:

I/O Port (Jack)	Jack ID
SDI Out A	0
SDI In A	1
SDI Out B	2
SDI In B	3

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 OEM 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_INPUT);

  // 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
  res = sv_fifo_init(sv, &pfifo_out, 0,...);
  ...

  // THREAD input A
  res = sv_fifo_init(sv, &pfifo_in_a, 1,...);
  ...
  // THREAD input B
  res = sv_fifo_init(sv, &pfifo_in_a, 3,...);
  ...
```

Multi-channel and Global Functions

In multi-channel operation mode you normally cannot use any of the global functions of the DVS SDK (e.g. the function *sv_videomode()*) due to the fact that they do not provide any jack parameters. If calling such 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")`. 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)	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).

Defines

- #define SV_OPTION_MULTICHANNEL
- #define SV_OPTION_MULTICHANNEL_LOCK

Functions

- int sv_jack_assign (sv_handle *sv, int jack, int channel)
- int sv_jack_find (sv_handle *sv, int jack, char *pstring, int stringsize, int *pjack)
- 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. Possible values are:

- `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 HydraCinema the default is 'on'.
- `SV_MULTICHANNEL_INPUT` – Turns the multi-channel operation mode on for an input only.
- `SV_MULTICHANNEL_OUTPUT` – Turns the multi-channel operation mode on for an output only.

With the multi-channel operation mode activated you can configure further jacks beyond the two default jacks. Once configured, separate FIFOs can be run on these jacks. To assign a timecode to another jack than the default jack use the defines SV_OPTION_ASSIGN_LTC and SV_OPTION_ASSIGN_VTR (Centaurus II only).

Note:

The multi-channel operation mode is only available on Centaurus II and HydraCinema.

On Centaurus II the multi-channel operation mode is 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 normally used for dual link are reconfigured as two independent and separate YUV422 links.

By using the multi-channel operation mode in conjunction with the define SV_OPTION_ALPHAMIXER you can mix (merge) to images available in the video board storage.

See also:

The defines SV_OPTION_MULTICHANNEL_LOCK, SV_OPTION_ASSIGN_LTC and SV_OPTION_ASSIGN_VTR.

#define SV_OPTION_MULTICHANNEL_LOCK

This define can be used to lock the tick counter of all output jacks. Afterwards all jacks will provide the same tick as the first jack. It works only if the jacks are configured to the same video raster. Possible values are:

- `0` – Multi-channel tick lock off.
- `1` – Multi-channel tick lock on.

See also:

The define SV_OPTION_MULTICHANNEL.

Function Documentation

int sv_jack_assign (sv_handle * *sv*, int *jack*, int *channel*)

This function is obsolete. It was used to assign a hardware channel to a jack. Currently the DVS SDK uses fixed IDs for the jacks (see the introduction to chapter API – Jack API).

Parameters:

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

jack – Jack ID/index.

channel – Channel to which this jack should be connected.

Returns:

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

int sv_jack_find (sv_handle * *sv*, int *jack*, char * *pstring*, int *stringsize*, int * *pjack*)

This function is obsolete. It was used to retrieve a jack index from a given jack name, or vice versa (retrieve a jack name from a given jack index). Currently the DVS SDK uses fixed IDs for the jacks (see the introduction to chapter API – Jack API).

Parameters:

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

jack – Jack ID/index (input parameter, set to -1 if unknown).

pstring – Jack name (input and output parameter).

stringsize – Size of the parameter *pstring*.

pjack – Jack ID/index (output parameter).

Returns:

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

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 needs to be told how many jacks the application is about to use and how much memory should be assigned to each jack.

There is no automatic memory assignment as only your application can know, how many jacks it will need 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 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>` code.

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>` code.

value – Value to set.

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>` code.

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>`.

OEM – Capture

Detailed Description

The Capture API enables you to capture a down-converted version 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). Because down-conversion is mainly used to give out a signal at the SD-only output of the board, the capturing supports the YUV422 8 bit format only.

All `SV_OPTION_PROXY_<xxx>` settings detailed in the following affect the SD output of the DVS video device as well as the buffers returned by the *sv_capture()* function.

This API cannot be used in conjunction with an SDStationOEM or SDStationOEM II.

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_TIMECODE
- #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)

Define Documentation

#define SV_OPTION_PROXY_ASPECTRATIO

This define sets the aspect ratio of the down-converted material. The value is expected as a fixed point float.

Note:

Not available for the SDStationOEM and SDStationOEM II.

#define SV_OPTION_PROXY_OPTIONS

This define sets various options related to the function *sv_capture()* and the down-converted output:

- `SV_PROXY_OPTION_NTSCJAPAN` – Sets the NTSC clamping level to Japan standards (DVS internal only).

- `SV_PROXY_OPTION_SDTVFULL` – Sets the proxy output to give out full-sized SDTV material, otherwise it will be down-scaled as well. This flag has an influence only when the output is set to an SDTV mode.

Note:

Not available for the SDStationOEM and SDStationOEM II.

#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.

Not available for the SDStationOEM and SDStationOEM II.

#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.

Note:

Not available for the SDStationOEM and SDStationOEM II.

#define SV_OPTION_PROXY_TIMECODE

This define enables a display of timecode information on the SD output:

- `SV_PROXY_TIMECODE_DISABLED` – No timecode information is displayed.
- `SV_PROXY_TIMECODE_LTC` – The LTC timecode is displayed.
- `SV_PROXY_TIMECODE_VITC` – The VITC timecode is displayed.
- `SV_PROXY_TIMECODE_KEYCODE` – The keycode is displayed.

Note:

This define is available for DVS internal use only.

#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.

Note:

Not available for the SDStationOEM and SDStationOEM II.

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 captured from the video output in a down-scaled format. The format can be adjusted by various calls of `SV_OPTION_PROXY_<xxx>`. The driver internally buffers up to four fields.

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 the 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. 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_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 down-scaled 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>`.

OEM – 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_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_LTC
- #define SV_OPTION_ASSIGN_VTR
- #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_LTCFLAGS
- #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_CLOSEDCAPTION
- #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_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_VITCINPUTLINE
- #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.

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 `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 `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.

Note:

The `SV_ANCCOMPLETE_ON` setting works with Centaurus (firmware version 2.<x>.50c or higher) and Centaurus II only.

The `SV_ANCCOMPLETE_STREAMER` setting is supported on Centaurus II only.

The `SV_ANCCOMPLETE_STREAMER` setting works in SDTV rasters only. In the past it could be used in HDTV rasters as well, but this approach has been abandoned. To stream the ANC data in other rasters than SDTV 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.

Note:

Only supported on Centaurus, Centaurus II and the SDStationOEM II.

#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_LTC

This define assigns the LTC timecode 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.

Note:

This feature is only available on Centaurus II.

See also:

The define `SV_OPTION_MULTICHANNEL`.

#define SV_OPTION_ASSIGN_VTR

This define assigns the RS-422 timecode 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.

Note:

This feature is only available on Centaurus II.

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 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_LTCFLAGS

Currently not used.

#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_PLAYLIST – Obsolete. It was used to determine that the timecode from a play list should be used.
- 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.
- SV_LTCSOURCE_VCOUNT – Flag to send out a tick count as user bytes.

#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 are:

- `SV_VITCLINE_DEFAULT` – Outputs the VITC on the first possible line number.
- `SV_VITCLINE_DISABLED` – Disables the VITC output.
- `SV_VITCLINE_ARP201` – Enables the full three-line VITC (only possible on Centaurus, Centaurus II and the SDStationOEM II).

Note:

Analog VITC only exists in SDTV rasters.

#define SV_OPTION_VITCREADERLINE

This define sets the line number where the VITC reader should capture the analog VITC timecode. Possible values are:

- `SV_VITCLINE_DEFAULT` – Reads the VITC from the first possible line number.
- `SV_VITCLINE_DISABLED` – Disables the VITC reader.

Note:

Analog VITC only exists in SDTV rasters.

#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_CLOSEDCAPTION

This define returns the two last found closed-caption bytes.

#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_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_VITCINPUTLINE

This define returns the line number where the VITC was found.

Note:

This is available on the SDStationOEM and the SDStationOEM II only.

#define SV_QUERY_VITCLINE

This define returns the number of the line where the VITC is sent out.

#define SV_QUERY_VITCREADERLINE

This define returns the number of the line where the VITC will be read.

#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>`.

OEM – 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_EPLDTYPE
- #define SV_QUERY_HW_EPLDVERSION
- #define SV_QUERY_HW_FLASHSIZE
- #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_1V5
- #define SV_QUERY_VOLTAGE_2V3
- #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_EPLDTYPE

For DVS internal use only.

#define SV_QUERY_HW_EPLDVERSION

This define returns the hardware EPLD version.

#define SV_QUERY_HW_FLASHSIZE

For DVS internal use only.

#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 OEM products (first digits) are listed in the section Supported DVS OEM 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_1V5

This define returns the measured 1.5 V (value is fixed float).

#define SV_QUERY_VOLTAGE_2V3

This define returns the measured 2.3 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).

OEM – 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 `SV_OPTION_GPI` can be used to set the GPI output data. You can always read the GPI input with the function *sv_query()* by using `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.

Pin No.	Signal
1	–
2	GND
3	GPI_OUT0 (GPI output 0)
4	GPI_IN0 (GPI input 0)
5	–
6	–
7	GPI_OUT1 (GPI output 1)
8	GPI_IN1 (GPI input 1)
9	GND

Note:

The GPI inputs are voltage sensing inputs with TTL trigger levels (> 2 V = high, < 0,8 V = low). Without any input they are set to 'high'. Thus, with a connected switch the user will be able to connect the voltage level to ground (GND) and no extra power supply has to be set for the GPI inputs.

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 `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 `SV_OPTION_GPI`.
- `SV_GPIOUT_INOUTPOINT` – bit0-Inpoint / bit1-Outpoint.
- `SV_GPIOUT_PULLDOWN` – bit0-PhaseA bit1-Valid.
- `SV_GPIOUT_PULLDOWNPHASE` – 00-A 01-B 10-C 11-D.
- `SV_GPIOUT_REPEATED` – bit0-Valid bit1-Repeated(=!valid).

#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 `SV_OPTION_GPIIN`.

#define SV_QUERY_GPIOUT

This define returns the setting of the GPI output. See the define `SV_OPTION_GPIOUT`.

OEM – The sv_rs422_<xxx> Functions

Detailed Description

The *sv_rs422_<xxx>* functions allow you to access the serial remote ports on a lower level where you can configure them to your personal needs. Mainly these functions are used to access the two auxiliary serial ports for remote control.

The auxiliary RS-422 ports are available, for example, on the SDStationOEM II where you have a total of four RS-422 ports available (two main ports and two auxiliary ports). On most other boards there are only the two main serial ports available.

On Centaurus and Centaurus II you can use these functions to access the main ports 0 and 1, for example, in case they should provide a different configuration than the one automatically set with the default RS-422 functions.

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 because normally they do not support a higher speed.

Pins on a 9-pin Slave Connector:

Pin No.	Signal
1	Frame GND
2	Transmit (-)
3	Receive (+)
4	GND
5	Not connected
6	GND
7	Transmit (+)
8	Receive (-)
9	Frame GND

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 line.

flags – Additional flags to control the behavior of this function. See list below.

Values for *device*:

- `0` – Main RS-422 port A (Centaurus and Centaurus II only).
- `1` – Main RS-422 port B (Centaurus and Centaurus II only).
- `2` – Auxiliary RS-422 port C (SDStationOEM and SDStationOEM II only).
- `3` – Auxiliary RS-422 port D (SDStationOEM and SDStationOEM II only).

Parameters for *flags*:

- `SV_RS422_OPENFLAG_SWAPPINOUT` – Uses a reverse pin-out (available for Centaurus, Centaurus II and the SDStationOEM II).

Returns:

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

Note:

Note that on the SDStationOEM and SDStationOEM II both ports C and D are wired as slave ports. You need a crossover cable when connecting to a VTR to perform master control.

On Centaurus and Centaurus II the remote control port A is wired as a master port and port B is wired as a slave port. In case you set the `SV_RS422_OPENFLAG_SWAPPINOUT` flag, the master/slave setting will get swapped.

An RS-422 null-modem cable has the pins 2 and 8 swapped as well as the pins 3 and 7.

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 is free of any interpretation of the RS-422 commands and 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 for 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>`.

OEM – The sv_slaveinfo_<xxx> Functions

Detailed Description

The *sv_slaveinfo_<xxx>* functions described in this chapter allow you to implement a VTR emulation with the low-level CRC and communications handling performed in the driver.

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 from the driver about the connected master. 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 from the driver and sent to the connected master.

Parameters:

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

slaveinfo – Pointer to the structure *sv_slaveinfo*.

Returns:

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

Note:

If the flag `SV_SLAVEINFO_USESTATUSANDTC` is not set, the status and timecode provided by the FIFO API will be used.

OEM – Tracing

Detailed Description

This section describes the DVSOEM debug tracing.

With the debug version of the DVSOEM (`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. Copy the file `dvsoemdbg.dll` into the directory where the file `dvsoem.dll` of your application is stored.

To use this under UNIX use the file `libdvsoemdbg.a` in your application 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.

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 do not have any useful functionality anymore.

Defines

- #define SV_OPTION_ANALOG_BLACKLEVEL_B_U
- #define SV_OPTION_ANALOG_BLACKLEVEL_BASE
- #define SV_OPTION_ANALOG_BLACKLEVEL_G_Y
- #define SV_OPTION_ANALOG_BLACKLEVEL_R_V
- #define SV_OPTION_ANALOG_GAIN_B_U
- #define SV_OPTION_ANALOG_GAIN_BASE
- #define SV_OPTION_ANALOG_GAIN_G_Y
- #define SV_OPTION_ANALOG_GAIN_R_V
- #define SV_OPTION_ANALOG_OFFSET_B_U
- #define SV_OPTION_ANALOG_OFFSET_BASE
- #define SV_OPTION_ANALOG_OFFSET_G_Y
- #define SV_OPTION_ANALOG_OFFSET_R_V
- #define SV_OPTION_ANALOG_RW_DEFAULT
- #define SV_OPTION_ANALOG_RW_USERDEF
- #define SV_OPTION_BUMPMODE
- #define SV_OPTION_CLUT
- #define SV_OPTION_CYCLES
- #define SV_OPTION_GAMMA_RW_DEFAULT
- #define SV_OPTION_RESETBEFORERECORD
- #define SV_OPTION_STRIPE_FORMAT
- #define SV_OPTION_VIDEOMODE_NOTRASTER
- #define SV_QUERY_AUDIOSEGMENT
- #define SV_QUERY_INTERLACE_ID
- #define SV_QUERY_PRONTOVISION
- #define SV_QUERY_VSYNCWAIT
- #define SV_QUERY_VTRMASTER_LOCAL

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)
- int sv_mixer_edge (sv_handle *sv, int edge, int edgeparam, int width, int spare)
- char * sv_vtrerror (sv_handle *sv, int code)

Define Documentation

#define SV_OPTION_ANALOG_BLACKLEVEL_B_U

This define has no functionality anymore. It was used for an analog output configuration on a previous disk recorder product by DVS.

#define SV_OPTION_ANALOG_BLACKLEVEL_BASE

This define has no functionality anymore. It was used for an analog output configuration on a previous disk recorder product by DVS.

#define SV_OPTION_ANALOG_BLACKLEVEL_G_Y

This define has no functionality anymore. It was used for an analog output configuration on a previous disk recorder product by DVS.

#define SV_OPTION_ANALOG_BLACKLEVEL_R_V

This define has no functionality anymore. It was used for an analog output configuration on a previous disk recorder product by DVS.

#define SV_OPTION_ANALOG_GAIN_B_U

This define has no functionality anymore. It was used for an analog output configuration on a previous disk recorder product by DVS.

#define SV_OPTION_ANALOG_GAIN_BASE

This define has no functionality anymore. It was used for an analog output configuration on a previous disk recorder product by DVS.

#define SV_OPTION_ANALOG_GAIN_G_Y

This define has no functionality anymore. It was used for an analog output configuration on a previous disk recorder product by DVS.

#define SV_OPTION_ANALOG_GAIN_R_V

This define has no functionality anymore. It was used for an analog output configuration on a previous disk recorder product by DVS.

#define SV_OPTION_ANALOG_OFFSET_B_U

This define has no functionality anymore. It was used for an analog output configuration on a previous disk recorder product by DVS.

#define SV_OPTION_ANALOG_OFFSET_BASE

This define has no functionality anymore. It was used for an analog output configuration on a previous disk recorder product by DVS.

#define SV_OPTION_ANALOG_OFFSET_G_Y

This define has no functionality anymore. It was used for an analog output configuration on a previous disk recorder product by DVS.

#define SV_OPTION_ANALOG_OFFSET_R_V

This define has no functionality anymore. It was used for an analog output configuration on a previous disk recorder product by DVS.

#define SV_OPTION_ANALOG_RW_DEFAULT

This define has no functionality anymore. It was used for an analog output configuration on a previous disk recorder product by DVS.

#define SV_OPTION_ANALOG_RW_USERDEF

This define has no functionality anymore. It was intended for an analog output configuration.

Note:

Not implemented.

#define SV_OPTION_BUMPMODE

This define has no functionality anymore. It was used to enable or disable a correction of the vertical position during slowmotion/fastmotion.

#define SV_OPTION_CLUT

This define has no functionality anymore.

#define SV_OPTION_CYCLES

This define has no functionality anymore.

#define SV_OPTION_GAMMA_RW_DEFAULT

This define has no functionality anymore. It was intended for an analog output configuration.

Note:

Not implemented.

#define SV_OPTION_RESETBEFORERECORD

This define has no functionality anymore.

#define SV_OPTION_STRIPE_FORMAT

This define has no functionality anymore.

#define SV_OPTION_VIDEOMODE_NOTRASTER

Same as `SV_OPTION_VIDEOMODE`.

#define SV_QUERY_AUDIOSEGMENT

This define has no functionality anymore. It was used to get the currently active audio segment.

#define SV_QUERY_INTERLACE_ID

This define has no functionality anymore. It was used to return the field order in temporal direction for the current video mode.

#define SV_QUERY_PRONTOVISION

This define has no functionality anymore. It was used to get the current device setup of a DVS Pronto Vision.

#define SV_QUERY_VSYNCWAIT

This define has no functionality anymore. Instead use the function *sv_vsyncwait()*. It was used to return the field number count (interlaced raster) or the frame number count (progressive raster) after waiting for the next vertical sync.

#define SV_QUERY_VTRMASTER_LOCAL

This define has no functionality anymore.

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 timecode *tc* is not zero (`0`), the VTR connected to the RS-422 master port will use this timecode.

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>`.

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 – Currently not used.

tc – Timecode for the VTR operation. A value of zero (`0`) means no VTR operation.

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>`.

Note:

This function is not intended to be used together with the FIFO API.

int sv_mixer_edge (sv_handle * *sv*, int *edge*, int *edgeparam*, int *width*, int *spare*)

This function is obsolete. There is no replacement function available. It was used to set the edge type and width for *sv_mixer_<xxx>* operations.

Parameters:

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

edge – Defines the edge type (`SV_MIXER_EDGE_<xxx>`).

edgeparam – Optional parameter for the edge type.

width – Width of *edge*.

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>`.

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 – Video Rasters

The following lists the supported video rasters. All frequencies indicate the frame rate.

SDTV:

- SV_MODE_PAL – 720 x 576, 25.00 Hz, interlaced
- SV_MODE_NTSC – 720 x 486, 29.97 Hz, interlaced
- SV_MODE_PALHR – 960 x 576, 25.00 Hz, interlaced
- SV_MODE_NTSCHR – 960 x 486, 29.97 Hz, interlaced

SMPTE240:

- SV_MODE_SMPTE240_29I – SMPTE240 1920 x 1035, 29.97 Hz, interlaced
- SV_MODE_SMPTE240_30I – SMPTE240 1920 x 1035, 30.00 Hz, interlaced

SMPTE274:

- SV_MODE_SMPTE274_23I – SMPTE274 1920 x 1080, 23.98 Hz, interlaced
- SV_MODE_SMPTE274_24I – SMPTE274 1920 x 1080, 24.00 Hz, interlaced
- SV_MODE_SMPTE274_25I – SMPTE274 1920 x 1080, 25.00 Hz, interlaced
- SV_MODE_SMPTE274_29I – SMPTE274 1920 x 1080, 29.97 Hz, interlaced
- SV_MODE_SMPTE274_30I – SMPTE274 1920 x 1038, 30.00 Hz, interlaced
- SV_MODE_SMPTE274_23sF – SMPTE274 1920 x 1080, 23.98 Hz, segmented frame
- SV_MODE_SMPTE274_24sF – SMPTE274 1920 x 1080, 24.00 Hz, segmented frame
- SV_MODE_SMPTE274_25sF – SMPTE274 1920 x 1080, 25.00 Hz, segmented frame
- SV_MODE_SMPTE274_29sF – SMPTE274 1920 x 1080, 29.97 Hz, segmented frame
- SV_MODE_SMPTE274_30sF – SMPTE274 1920 x 1080, 30.00 Hz, segmented frame
- SV_MODE_SMPTE274_23P – SMPTE274 1920 x 1080, 23.98 Hz, progressive
- SV_MODE_SMPTE274_24P – SMPTE274 1920 x 1080, 24.00 Hz, progressive
- SV_MODE_SMPTE274_25P – SMPTE274 1920 x 1080, 25.00 Hz, progressive
- SV_MODE_SMPTE274_29P – SMPTE274 1920 x 1080, 29.97 Hz, progressive
- SV_MODE_SMPTE274_30P – SMPTE274 1920 x 1080, 30.00 Hz, progressive

SMPTE274 (Analog Output only):

- SV_MODE_SMPTE274_47P – 1920 x 1080, 47.95 Hz, progressive
- SV_MODE_SMPTE274_48P – 1920 x 1080, 48.00 Hz, progressive
- SV_MODE_SMPTE274_59P – 1920 x 1080, 59.94 Hz, progressive
- SV_MODE_SMPTE274_60P – 1920 x 1080, 60.00 Hz, progressive
- SV_MODE_SMPTE274_71P – 1920 x 1080, 71.93 Hz, progressive
- SV_MODE_SMPTE274_72P – 1920 x 1080, 72.00 Hz, progressive

SMPTE295:

- SV_MODE_SMPTE295_25I – SMPTE295 1920 x 1080 (total lines 1250 per frame), 25.00 Hz, interlaced

SMPTE296:

- SV_MODE_SMPTE296_23P – SMPTE296 1280 x 720, 23.98 Hz, progressive
- SV_MODE_SMPTE296_24P – SMPTE296 1280 x 720, 24.00 Hz, progressive
- SV_MODE_SMPTE296_25P – SMPTE296 1280 x 720, 25.00 Hz, progressive
- SV_MODE_SMPTE296_29P – SMPTE296 1280 x 720, 29.97 Hz, progressive
- SV_MODE_SMPTE296_30P – SMPTE296 1280 x 720, 30.00 Hz, progressive

- `SV_MODE_SMPTE296_50P` – SMPTE296 1280 x 720, 50.00 Hz, progressive
- `SV_MODE_SMPTE296_59P` – SMPTE296 1280 x 720, 59.94 Hz, progressive
- `SV_MODE_SMPTE296_60P` – SMPTE296 1280 x 720, 60.00 Hz, progressive
- `SV_MODE_SMPTE296_71P` – SMPTE296 1280 x 720, 71.93 Hz, progressive
- `SV_MODE_SMPTE296_72P` – SMPTE296 1280 x 720, 72.00 Hz, progressive

FILM2K:

- `SV_MODE_FILM2K_1080_23sF` – FILM2K 2048 x 1080, 23.98 Hz, segmented frame
- `SV_MODE_FILM2K_1080_24sF` – FILM2K 2048 x 1080, 24.00 Hz, segmented frame
- `SV_MODE_FILM2K_1080_23P` – FILM2K 2048 x 1080, 23.98 Hz, progressive
- `SV_MODE_FILM2K_1080_24P` – FILM2K 2048 x 1080, 24.00 Hz, progressive
- `SV_MODE_FILM2K_1536_24P` – FILM2K 2048 x 1536, 24.00 Hz, progressive
- `SV_MODE_FILM2K_1536_24sF` – FILM2K 2048 x 1536, 24.00 Hz, segmented frame
- `SV_MODE_FILM2K_1536_48P` – FILM2K 2048 x 1536, 48.00 Hz, progressive
- `SV_MODE_FILM2K_1556_14sF` – FILM2K/HSDL 2048 x 1556, 14.99 Hz, segmented frame
- `SV_MODE_FILM2K_1556_15sF` – FILM2K/HSDL 2048 x 1556, 15.00 Hz, segmented frame
- `SV_MODE_FILM2K_1556_24P` – FILM2K 2048 x 1556, 24.00 Hz, progressive
- `SV_MODE_FILM2K_1556_24sF` – FILM2K 2048 x 1556, 24.00 Hz, segmented frame
- `SV_MODE_FILM2K_1556_29sF` – FILM2K 2048 x 1556, 29.97 Hz, segmented frame
- `SV_MODE_FILM2K_1556_30sF` – FILM2K 2048 x 1556, 30.00 Hz, segmented frame
- `SV_MODE_FILM2K_1556_48P` – FILM2K 2048 x 1556, 48.00 Hz, progressive

FILM4K:

- `SV_MODE_FILM4K_3112_5sF` – FILM4K 4096 x 3112, 5.00 Hz, segmented frame

VESA Rasters:

- `SV_MODE_VESA_640x480_59P` – 640 x 480, 59.94 Hz, progressive
- `SV_MODE_VESA_640x480_60P` – 640 x 480, 60.00 Hz, progressive
- `SV_MODE_VESA_640x480_71P` – 640 x 480, 71.93 Hz, progressive
- `SV_MODE_VESA_640x480_72P` – 640 x 480, 72.00 Hz, progressive
- `SV_MODE_VESA_800x600_59P` – 800 x 600, 59.94 Hz, progressive
- `SV_MODE_VESA_800x600_60P` – 800 x 600, 60.00 Hz, progressive
- `SV_MODE_VESA_800x600_71P` – 800 x 600, 71.93 Hz, progressive
- `SV_MODE_VESA_800x600_72P` – 800 x 600, 72.00 Hz, progressive
- `SV_MODE_VESA_1024x768_59P` – 1024 x 768, 59.94 Hz, progressive
- `SV_MODE_VESA_1024x768_60P` – 1024 x 768, 60.00 Hz, progressive
- `SV_MODE_VESA_1024x768_71P` – 1024 x 768, 71.93 Hz, progressive
- `SV_MODE_VESA_1024x768_72P` – 1024 x 768, 72.00 Hz, progressive
- `SV_MODE_VESA_1280x1024_59P` – 1280 x 1024, 59.94 Hz, progressive
- `SV_MODE_VESA_1280x1024_60P` – 1280 x 1024, 60.00 Hz, progressive
- `SV_MODE_VESA_1280x1024_71P` – 1280 x 1024, 71.93 Hz, progressive
- `SV_MODE_VESA_1280x1024_72P` – 1280 x 1024, 72.00 Hz, progressive
- `SV_MODE_VESA_1600x1200_59P` – 1600 x 1200, 59.94 Hz, progressive
- `SV_MODE_VESA_1600x1200_60P` – 1600 x 1200, 60.00 Hz, progressive
- `SV_MODE_VESA_1600x1200_71P` – 1600 x 1200, 71.93 Hz, progressive
- `SV_MODE_VESA_1600x1200_72P` – 1600 x 1200, 72.00 Hz, progressive

VESA Rasters, NTSC Syncable:

- `SV_MODE_VESA_1024x768_29I` – 1024 x 768, 29.97 Hz, interlaced
- `SV_MODE_VESA_1280x1024_29I` – 1280 x 1024, 29.97 Hz, interlaced
- `SV_MODE_VESA_1600x1200_29I` – 1600 x 1200, 29.97 Hz, interlaced

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 00AABBCC ...
98765432 98765432 98765432 --101010 ...
```

12 bit

`SV_MODE_NBITS_12B` – Not supported on the SDStationOEM

```
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` – Not supported on the SDStationOEM

```
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 section 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 UYUY.

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 section 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_1CHANNEL` – Configures one stereo audio channel (only supported on the SDStationOEM and SDStationOEM II).
- `SV_MODE_AUDIO_2CHANNEL` – Configures two stereo audio channels (only supported on the SDStationOEM and SDStationOEM II).
- `SV_MODE_AUDIO_4CHANNEL` – Configures four stereo audio channels (only supported on the SDStationOEM and SDStationOEM II).
- `SV_MODE_AUDIO_6CHANNEL` – Configures six stereo audio channels (only supported on the SDStationOEM and SDStationOEM II).
- `SV_MODE_AUDIO_8CHANNEL` – Configures eight stereo audio channels.

Audio Sample Size:

- `SV_MODE_AUDIOBITS_16` – Configures a sample size of 16 bit (only supported on the SDStationOEM and SDStationOEM II).
- `SV_MODE_AUDIOBITS_32` – Configures a sample size of 32 bit.

Info – Storage Formats

This section describes the video storage formats that can be used in the video buffers of the DVS video boards.

Frame Storage:

- `SV_MODE_STORAGE_FRAME` – Both fields of interlaced rasters are stored in one buffer. Without this flag the fields are stored in two separate buffers.

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 section 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_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_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_MEMORY (168) – Not all memory modules found.
- SV_ERROR_DRIVER_MEMORYMATCH (169) – Mounted memory modules do not match.
- SV_ERROR_DRIVER_RESOURCES (79) – The driver did not get resources from the kernel.
- SV_ERROR_DRIVER_MISMATCH (147) – Driver and library version mismatch detected.
- 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 on 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_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_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_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_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_12BITS (175) – You tried to use a 12-bit I/O mode without license.

- SV_ERROR_LICENCE_KEYCHANNEL (132) – You tried to use a key channel without license.
- SV_ERROR_LICENCE_MIXER (133) – You tried to use the mixer 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_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_STREAMER (130) – You tried to use the streamer mode 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_MMAPFAILED (87) – Memory mapping function failed.
- 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_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_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 SDStation.
- 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 VTR front panel switch.
- 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.

Example Projects Overview

The DVS SDK comes with some example projects that can be analyzed to understand the SDK programming. This section describes the function and purposes of these example projects. The projects are stored in the directory `sdk3.<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 `sdk3.<x>.<y>.<z>/{win32, linux, ...}/bin` directory.

Most of the examples mentioned in the following demonstrate the usage of the 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.

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.

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 (*pvspeed* in the DVS SDK 2.7)

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.

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. 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 Capture API and displays the current output signal in a down-scaled 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.

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.

Example – dpxio

Detailed Description

This chapter shows the use of the 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*)

Dumps timing measurement results.

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_status()*.

In all error logging functions we use the function *sv_geterrortext()* to get a readable form of the SV error codes.

```
    res = sv_status(dpxio->sv, &svinfo);
    if(res != SV_OK) {
      printf("ERROR: sv_info() failed = %d '%s'\n", res, sv_geterrortext(res));
      running = FALSE;
    }
```

For the input we use the size returned by the function *sv_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 a good idea to page-align any 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 need at least to be at aligned to the size returned from the driver for the board's minimum DMA alignment, for example, for Centaurus this is 16 bytes and for the SDStationOEM it is 8 bytes.

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 editfield 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 editfield 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 the *init* parameter set 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 in 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 for 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 for which 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 `cmodtst` program.

Some fields in the *sv_fifo_buffer* structure need to be set and the flag `SV_FIFO_FLAG_STORAGEMODE` needs to be set in the sv_fifo_getbuffer() call. In the sv_fifo_putbuffer() call the needed 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 if any frames where 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*)

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*)

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_fifo_ancbuffer Struct Reference

Detailed Description

The following details the ANC data structure used by the function *sv_fifo_anc()*.

Field Documentation

unsigned char sv_fifo_ancbuffer::data[256]

Buffer for the data payload (element *datasize*).

int sv_fifo_ancbuffer::datasize

Data payload, i.e. the number of bytes in the data element.

int sv_fifo_ancbuffer::did

Data ID of this ANC packet.

int sv_fifo_ancbuffer::field

Field index (`0` or `1`) of this ANC packet.

int sv_fifo_ancbuffer::linenr

Line number of this ANC packet.

int sv_fifo_ancbuffer::pad[6]

Reserved for future use. Set to zero (`0`).

int sv_fifo_ancbuffer::sdid

Secondary data ID of this ANC packet.

int sv_fifo_ancbuffer::vanc

Position of this packet. Either VANC (`1`) or HANC (`0`).

sv_fifo_buffer Struct Reference

Detailed Description

The following details the FIFO buffer structure used by the functions *sv_fifo_getbuffer()* and *sv_fifo_putbuffer()*.

Field Documentation

int sv_fifo_buffer::aclock_high

Clock of the MSBs (most significant bytes) when audio was captured. Valid for an input only.

int sv_fifo_buffer::aclock_low

Clock of the LSBs (least significant bytes) when audio was captured. Valid for an input only.

char* sv_fifo_buffer::addr

Pointer/offset to the ANC data.

char* sv_fifo_buffer::addr

Pointer/offset to the video data channel B (second video image, see the define `SV_FIFO_FLAG_VIDEO_B`).

char* sv_fifo_buffer::addr[4]

Pointer/offset to the audio data of channels 5 to 8.

char* sv_fifo_buffer::addr[4]

Pointer/offset to the audio data of channels 1 to 4.

char* sv_fifo_buffer::addr

Pointer/offset to the video data.

char* sv_fifo_buffer::addr

Pointer for the DMA.

int sv_fifo_buffer::afilm_tc[2]

Analog film timecode.

int sv_fifo_buffer::afilm_ub[2]

Analog film user bytes.

struct { ... } sv_fifo_buffer::anc[2]

Array containing two ANC fields.

struct { ... } sv_fifo_buffer::anctimecode

ANC timecode substructure.

int sv_fifo_buffer::aprod_tc[2]

Analog production timecode.

int sv_fifo_buffer::aprod_ub[2]

Analog production user bytes.

struct { ... } sv_fifo_buffer::audio[2]

Array containing two audio fields.

struct { ... } sv_fifo_buffer::audio2[2]

Array containing two audio fields.

int sv_fifo_buffer::clock_high

Clock of the MSBs (most significant bytes) when a frame was captured. Valid for an input only.

int sv_fifo_buffer::clock_low

Clock of the LSBs (least significant bytes) when a frame was captured. Valid for an input only.

int sv_fifo_buffer::closedcaption[2]

Analog closed caption. Valid for an input only.

int sv_fifo_buffer::cmd

The received VTR command. Valid for an input only.

struct { ... } sv_fifo_buffer::control

Various buffer related values.

unsigned char sv_fifo_buffer::data[16]

Data bytes. Valid for an input only.

int sv_fifo_buffer::dataoffset

Offset to the first pixel in the buffer.

int sv_fifo_buffer::dltc_tc

Digital/ANC LTC timecode.

int sv_fifo_buffer::dltc_ub

Digital/ANC LTC user bytes.

struct { ... } sv_fifo_buffer::dma

DMA substructure (not used when setting `SV_FIFO_FLAG_NODMADDR`).

int sv_fifo_buffer::dvitc_tc[2]

Digital/ANC VITC timecode.

int sv_fifo_buffer::dvitc_ub[2]

Digital/ANC VITC user bytes.

int sv_fifo_buffer::fifoid

FIFO/frame ID of the buffer. Do not change.

int sv_fifo_buffer::film_tc[2]

Digital/ANC film timecode (RP201).

int sv_fifo_buffer::film_ub[2]

Digital/ANC film user bytes (RP201).

int sv_fifo_buffer::flags

Used internally only, i.e. do not change.

int sv_fifo_buffer::gpi

GPI information of the FIFO buffer.

int sv_fifo_buffer::length

Number of data bytes. Valid for an input only.

int sv_fifo_buffer::lineoffset

Offset from line to line or zero (`0`) for default.

int sv_fifo_buffer::ltc_tc

Analog LTC timecode without bit masking.

int sv_fifo_buffer::ltc_ub

Analog LTC user bytes.

int sv_fifo_buffer::pad

Reserved for future use.

int sv_fifo_buffer::prod_tc[2]

Digital/ANC production timecode (RP201).

int sv_fifo_buffer::prod_ub[2]

Digital/ANC production user bytes (RP201).

int sv_fifo_buffer::size

Size of ANC data.

struct { ... } sv_fifo_buffer::storage

Dynamic storage mode. Valid for an output only.

int sv_fifo_buffer::storagemode

Image data storage mode.

int sv_fifo_buffer::tick

Tick when the frame was captured. Valid for an input only.

struct { ... } sv_fifo_buffer::timecode

Timecode substructure.

int sv_fifo_buffer::version

Used internally only, i.e. do not change.

struct { ... } sv_fifo_buffer::video[2]

Array containing two video fields.

struct { ... } sv_fifo_buffer::video_b[2]

Array containing two video fields (second video image, see the define `SV_FIFO_FLAG_VIDEO_B`).

int sv_fifo_buffer::vitc_tc

Analog VITC timecode.

int sv_fifo_buffer::vitc_tc2

Analog VITC timecode of field 2.

int sv_fifo_buffer::vitc_ub

Analog VITC user bytes.

int sv_fifo_buffer::vitc_ub2

Analog VITC user bytes of field 2.

int sv_fifo_buffer::vtr_info

VTR info bytes 0 to 3.

int sv_fifo_buffer::vtr_info2

VTR info bytes 4 to 7.

int sv_fifo_buffer::vtr_info3

VTR info bytes 8 to 11.

int sv_fifo_buffer::vtr_tc

VTR timecode.

int sv_fifo_buffer::vtr_tick

Capture tick for VTR timecode and user bytes.

int sv_fifo_buffer::vtr_ub

VTR user bytes.

struct { ... } sv_fifo_buffer::vtrcmd

Substructure containing the incoming VTR commands.

int sv_fifo_buffer::xoffset

Image data x-offset from center.

int sv_fifo_buffer::xsize

Image data x-size.

int sv_fifo_buffer::yoffset

Image data y-offset from center.

int sv_fifo_buffer::ysize

Image data y-size.

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()*.

Field Documentation

int sv_fifo_bufferinfo::clock_high

Buffer clock (upper 32 bits).

int sv_fifo_bufferinfo::clock_low

Buffer clock (lower 32 bits).

int sv_fifo_bufferinfo::clock_tolerance

Buffer clock operation tolerance.

int sv_fifo_bufferinfo::padding[32-6]

Reserved for future use.

int sv_fifo_bufferinfo::size

Size of the structure.

int sv_fifo_bufferinfo::version

Version of this structure.

int sv_fifo_bufferinfo::when

Buffer tick.

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.

Field Documentation

int sv_fifo_configinfo::abuffersize

Size of the audio data in the board memory that corresponds to one frame of video.

int sv_fifo_configinfo::ancbuffersize

Size of the ANC data in the board memory that corresponds to one frame of video.

int sv_fifo_configinfo::dmaalignment

Needed alignment of a DMA buffer.

int sv_fifo_configinfo::dmarect_lineoffset

Line to line offset for the current DMA rectangle.

int sv_fifo_configinfo::dmarect_xoffset

X-offset for the current DMA rectangle.

int sv_fifo_configinfo::dmarect_xsize

X-size for the current DMA rectangle.

int sv_fifo_configinfo::dmarect_yoffset

Y-offset for the current DMA rectangle.

int sv_fifo_configinfo::dmarect_ysize

Y-size for the current DMA rectangle.

int sv_fifo_configinfo::entries

Number of valid entries in this structure.

int sv_fifo_configinfo::field1offset

Offset to the start of field 1.

int sv_fifo_configinfo::field2offset

Offset to the start of field 2.

int sv_fifo_configinfo::nbuffers

Maximum number of buffers in this video mode.

int sv_fifo_configinfo::pad[64-15]

Reserved for future use.

void* sv_fifo_configinfo::unused1

No longer used.

void* sv_fifo_configinfo::unused2

No longer used.

int sv_fifo_configinfo::vbuffersize

Size of one video frame in the board memory.

sv_fifo_info Struct Reference

Detailed Description

The following describes the structure *sv_fifo_info* which is used by the function *sv_fifo_status()*.

Field Documentation

int sv_fifo_info::audioinerror

Audio input error code.

int sv_fifo_info::availbuffers

Number of free buffers.

int sv_fifo_info::clock_high

Clock time of the last vertical sync (upper 32 bits).

int sv_fifo_info::clock_low

Clock time of the last vertical sync (lower 32 bits).

int sv_fifo_info::clocknow_high

Current clock time (upper 32 bits).

int sv_fifo_info::clocknow_low

Current clock time (lower 32 bits).

int sv_fifo_info::displaytick

Current display tick.

int sv_fifo_info::dropped

Number of dropped frames.

int sv_fifo_info::nbuffers

Total number of buffers.

int sv_fifo_info::openprogram

Program which tried to open the device.

int sv_fifo_info::opentick

Tick of the time when the program tried to open the device.

int sv_fifo_info::pad26[8]

Reserved for future use.

int sv_fifo_info::recordtick

Current record tick.

int sv_fifo_info::tick

Current tick.

int sv_fifo_info::videoinerror

Video input error code.

int sv_fifo_info::waitdropped

Number of dropped waits for the vertical sync of the function *sv_fifo_getbuffer()*.

int sv_fifo_info::waitnotintime

Number of times with waits for the vertical sync that occurred not in real time.

Index