

Sensoray DVR Control Protocol

Copyright © 2023 Sensoray Company, Inc.

December 5, 2023

API Version 1.3

SENSORAY | embedded electronics



Designed and manufactured in the U.S.A.

SENSORAY | p. 503.684.8005 | email: info@SENSORAY.com | www.SENSORAY.com

7313 SW Tech Center Drive | Portland, OR 97203

Introduction

All Sensoray DVRs are capable of operating stand-alone with an attached keypad or keyboard. In addition, some models include a communication interface that allows the DVR to be remotely operated via Sensoray's DVR Control Protocol (DCP). This document specifies DCP, provides example implementations, and introduces an API that fully implements DCP.

Functional overview

A DCP-enabled DVR is a server that responds to commands from clients. It provides a single endpoint through which it communicates with all clients.

A client must provide at least one unique endpoint for each DVR it controls. One endpoint (required) is used to send commands to and receive replies from the DVR. A second, optional endpoint may be used to receive event notifications from the DVR.

All information exchanged between DVR and client is conveyed via messages. Three message types are defined: command, reply and event. Command and reply messages are used in synchronous transactions, whereas event messages are asynchronous, unsolicited "interrupt" messages issued by the server. The latter allows a client to be notified when the DVR state is changed automatically or by another client, thereby making it unnecessary to poll the DVR.

Each message is encapsulated in a COBS-encoded packet with NUL (0x00) packet delimiter. When character-oriented interfaces are used (e.g., serial communication interface), this allows a client to reliably detect message boundaries and to differentiate between reply and event messages, which are necessarily multiplexed into a single character stream.

Basic operation

A DVR function is invoked by sending a command packet to the DVR. Upon receiving a command packet, the DVR will process the command and send a reply packet to the endpoint from which the command was received. The DVR will not execute the command or send a reply if a packet error is detected; clients should detect missing replies via timeout.

Event reporting

When the DVR detects an internal, asynchronous event (e.g., storage becomes full while recording), it will send an event message to any client which has previously enabled reporting for that event type. Clients do not acknowledge or reply to received event messages.

Upon DVR power-up, reporting is disabled for all event types. A client may enable event reporting by sending to the server a `CMD_ENABLE_EVENT_REPORTING` command that specifies the event types that are to be reported.

Only one remote client can receive event messages. If a remote client attempts to enable or disable event reporting when it is already enabled for another remote client, the request will be honored and event reporting will be silently terminated for the other client.

Data conventions

Types

The following prefixes are used to indicate data type:

Prefix	Type
u8_	unsigned 8-bit integer
u16_	unsigned 16-bit integer
u32_	unsigned 32-bit integer
str_	unsigned 8-bit array containing a null-terminated string
s16_	signed 16-bit integer
dbl_	double-precision floating point

Byte order and packing

Multi-byte values are always little endian, byte-aligned.

Packet structure

An encoded packet consists of a header byte, payload and packet delimiter:

HDR	PAYLOAD	DELIM
-----	---------	-------

Field	Length (bytes)	Function	Details
HDR	1	COBS header	
PAYLOAD	2 to 255	COBS payload	COBS-encoded TYPE, BODY and SUM
DELIM	1	Packet delimiter	0x00

The unencoded/decoded payload consists of a message type code, variable-length message body, and checksum:

HDR	PAYLOAD		DELIM
Payload:	TYPE	BODY	SUM

Field	Length (bytes)	Function	Details
TYPE	1	Message type	0 = command; 64 = reply; 128 = event notification
BODY	0 to 253	Message body	Function-specific content, structure and length
SUM	1	Checksum	SUM = 0 - (modulo 256 sum of TYPE and BODY bytes). To validate, sum all packet bytes (including SUM) and verify least-significant byte = 0.

Command messages

Command messages are conveyed in the BODY fields of command packets (TYPE = 0). A command message consists of an Opcode byte followed by zero or more data bytes. The data may consist of any combination of single and multi-byte values. Opcode values are defined in header file `dvr_protocol.h`.

Payload:	0	BODY	SUM
Command msg:	Opcode	Data	

Reply messages

Reply messages are conveyed in the BODY fields of reply packets (TYPE = 64). A reply message consists of an Errcode byte followed by four or more data bytes. The data may consist of any combination of single and multi-byte values. DVR status (u32_stat) is always included at the end of the data (see Status flags).

Payload:	64	BODY	SUM
Reply msg:	Errcode	Data	

Errcode indicates whether the associated command executed normally and, if not, why execution failed. Errcode values are defined in header file `dvr_protocol.h`.

Errcode	Description	Data
DVR_ERR_OK	Command executed normally	Command-specific
DVR_ERR_INVALID_OPCODE	Command not supported	–
DVR_ERR_ILLEGAL_ARGVAL	Command argument outside legal range	–
DVR_ERR_MISSING_ARG	Missing command argument	–
DVR_ERR_FUNC_PROHIBITED	Command could not execute in current DVR state	–
DVR_ERR_FILE_NOT_FOUND	Specified file does not exist	–
DVR_ERR_MALLOC	Can't allocate memory	–
DVR_ERR_OPEN	Can't open DVR communication interface	
DVR_ERR_CLOSE	Can't close DVR communication interface	
DVR_ERR_SOCKETCREATE	Can't create socket	
DVR_ERR_SOCKETBIND	Socket bind failed	
DVR_ERR_RXTHREAD	Failed to start receiver thread	
DVR_ERR_TXTHREAD	Failed to start transmitter thread	
DVR_ERR_MUTEXCREATE	Can't create mutex	
DVR_ERR_LOCKCLOSED	Lock closed or timeout	
DVR_ERR_SOCKETCLOSED	Socket closed or shutdown	
DVR_ERR_OVERSIZE	Command arguments too large	
DVR_ERR_PACKETSEND	Failed to send command to DVR	
DVR_ERR_PACKETREAD	Socket closed or read error	
DVR_ERR_RUNTPACKET	Abnormally short DVR reply packet	
DVR_ERR_MISSINGDELIM	Abnormally long DVR reply packet	
DVR_ERR_CHECKSUM	Receive checksum error	
DVR_ERR_MSGTYPE	Unexpected/unknown TYPE code in DVR reply or event	
DVR_ERR_DEPLETED	Missing expected data in DVR reply	
DVR_ERR_STRINGBUF	String buffer too small to receive DVR reply string	
DVR_ERR_TIMEOUT	Read timeout	
DVR_ERR_EVENTTYPE	Unrecognized event type	
DVR_ERR_STORAGE	Out of storage	
DVR_ERR_VIDLOCK	Video not locked	

The data in a reply message is guaranteed to be valid only when Errcode is DVR_ERR_OK; for other Errcode values the data, if present, should be ignored.

Event messages

Event messages are conveyed in the BODY fields of event notification packets (TYPE = 128). An event message consists of an Event byte followed by zero or more data bytes. The data may consist of any combination of single and multi-byte values. Event code values are defined in header file dvr_protocol.h.

Payload:	128	BODY	SUM
Reply msg:	Event	Data	

Event indicates the type of event being reported and the data attributes.

Event	Data
DVR_EVENT_STATUS_CHANGE	u32_stat, u32_changemask

Commands, API Version 1.3

General

Command		Reply data	Min. API version
Opcode	Data		
CMD_GET_API_VERSION	–	u32_apiver, u32_stat	1.0
CMD_GET_DVR_STATUS	–	u32_stat	1.0
CMD_ENABLE_EVENT_REPORTING	u8_clientid, u32_eventflags, u16_port	u32_stat	1.0
CMD_GET_HARDWARE_INFO	–	u32_hwver, str_model, u32_stat	1.0
CMD_SET_DISPLAY_MODE	u8_dispmode	u32_stat	1.0
CMD_GET_GENERAL_SETTINGS	–	u8_dispmode, u32_stat	1.0
CMD_SET_LOW_POWER	u8_enable_low_power	u32_stat	1.3
CMD_SET_CAPTURE_RES	u32_width, u32_height	u32_stat	1.3

u32_apiver: 0xAABBCCCC, where AA is major version, BB is minor version, and CCCC is firmware version. For example, 0x010204d2 would indicate API version 1.2 with firmware version 1234 in decimal.

u32_hwver: currently returns 0x01000000 (may change in future version)

str_model: the Sensoray Model number for the product, for example “S4013” or “S4023”

u32_stat: bitwise-or of DVR_STAT_* (see dvr_protocol.h)

u32_eventflags: bitwise-or of DVR_STAT_* (see dvr_protocol.h)

u8_dispmode: one of enum DVR_DISPLAY_MODE (see dvr_protocol.h)

u8_enable_low_power: 0=normal mode 1=low power(video input/output disabled) 2=very low power (CPU sleep; DRAM in self-refresh mode; must pulse GPIO to wake up - low at least 500ns)

u32_width, u32_height: resolution of the captured video for video source. (Not applicable to AHD/Composite input.)

Note: Remote clients must set u8_clientid=0

Date/time

Command		Reply data	Min. API version
Opcode	Data		
CMD_SET_DATETIME	u32_clock	u32_stat	1.0
CMD_GET_DATETIME	–	u32_clock, u32_stat	1.0

u32_clock: an integer counting the number of seconds since 12:00:00AM Jan 1, 1970. See also: DateTime conversion functions in this document.

Record

Command		Reply data	Min. API version
Opcode	Data		
CMD_RECORD_START	u8_dev_id, u8_format, u16_bitrate_kbps	str_outfilename, u32_stat	1.0
CMD_RECORD_STOP	–	u32_stat	1.0
CMD_RECORD_PAUSE	–	u32_stat	1.0
CMD_RECORD_RESUME	–	u32_stat	1.0
CMD_SET_RECORD_FILENAME	str_filename	u32_stat	1.3
CMD_SET_RECORD_METADATA	u8_data()	u32_stat	1.3
CMD_ENABLE_RECORD_METADATA	u8_enable	u32_stat	1.3

u8_dev_id: one of enum DVR_STORAGE_DEVID (see dvr_protocol.h) [MMC=0, INTERNAL=1, EXTERNAL=2, BOTH=3]

u8 format: one of enum DVR_VIDREC_FORMAT (see dvr_protocol.h) [RECFMT_MP4=0, RECFMT_TS=1]

u16_bitrate_kbps: an integer for the recording video rate in kilobits/sec. For example, 2 Mbps would be 2000. If zero, uses default or last set bitrate.

str_outfilename: returned filename for the recorded video.

str_filename: requested filename for the recorded video. The default filename is “^d_^t.mp4” for MP4 format or “^d_^t.ts” for TS format. The filename may contain special codes:

- ^d current date in YYYYMMDD format
- ^t current time in HHMMSS format
- ^c counter starting at 1, incremented until would not overwrite an existing file

u8_data: binary data to be inserted into the MP4 file at the end of the file after the MOOV atom. The data will be encapsulated in a SKIP box, and the size of the box will be that of the metadata plus the box header (8 bytes). Note: Data must be uploaded in chunks - when the data length is non-zero, it is appended to existing data, otherwise the metadata is cleared. Any time metadata is sent, the metadata enable is forced to disabled (to prevent a partial metadata being written to a file.) After the complete data is sent, it must be enabled via CMD_ENABLE_RECORD_METADATA. The metadata should be sent and enabled prior to starting the recording, since its SKIP box will be inserted at the start of the file after the FTYP atom.

u8_enable: Set to 1 to enable insertion of metadata in recording, 0 otherwise.

Play

Command		Reply data	Min. API version
Opcode	Data		
CMD_PLAY_SKIP	s16_seconds	u32_stat	1.0
CMD_PLAY_START	u8_dev, str_filename	u32_stat	1.0
CMD_PLAY_STOP	–	u32_stat	1.0
CMD_PLAY_PAUSE	–	u32_stat	1.0
CMD_PLAY_RESUME	–	u32_stat	1.0
CMD_SET_PLAY_SPEED	u8_speed	u32_stat	1.0
CMD_GET_PLAY_SETTINGS	–	u8_speed, u32_stat	1.0

s16_seconds: an integer number of seconds to seek forward (positive) or backward (negative) in the currently playing video.

u8_dev: one of enum DVR_STORAGE_DEVID (see dvr_protocol.h) [MMC=0, INTERNAL=1, EXTERNAL=2]

str_filename: the file name of the video to be played.

u8_speed: one of enum DVR_PLAY_SPEED (see dvr_protocol.h) [PLAY_NORMAL=0, PLAY_FAST_FORWARD=1, PLAY_FAST_REVERSE=2]

Snapshot

Command		Reply data	Min. API version
Opcode	Data		
CMD_VIEW_SNAPSHOT	u8_dev, str_filename	u32_stat	1.0
CMD_GRAB_SNAPSHOT	u8_dev, u8_quality	str_outfilename, u32_stat	1.0
CMD_SNAPSHOT_STOP	-	u32_stat	1.1
CMD_SET_SNAPSHOT_FILENAME	str_filename	u32_stat	1.3
CMD_SET_SNAPSHOT_METADATA	u8_data()	u32_stat	1.3
CMD_ENABLE_SNAPSHOT_METADATA	u8_enable	u32_stat	1.3

u8_dev: one of enum DVR_STORAGE_DEVID (see dvr_protocol.h) [MMC=0, INTERNAL=1, EXTERNAL=2]

str_filename: the file name of the snapshot to be grabbed or displayed. The default filename is “^d_^t.jpg”. The filename may contain special codes:

- ^d current date in YYYYMMDD format
- ^t current time in HHMMSS format
- ^c counter starting at 1, incremented until would not overwrite an existing file

str_outfilename: returned file name of the captured snapshot.

u8_quality: the JPEG quality, ranged 2 (lowest) to 97 (highest.) If zero, uses default or last set quality.

u8_data: binary data to be inserted into the JPEG file in front of the EOI marker. The data should start with a valid metadata marker to avoid corrupting the JPEG bytestream. Suggested usage: comment 0xFFFE marker followed by 2-byte length and text data, or 0xFFEn APPn Application-specific data. Note: Data must be uploaded in chunks - when the data length is non-zero, it is appended to existing data, otherwise the metadata is cleared. Any time metadata is sent, the metadata enable is forced to disabled (to prevent a partial metadata being written to a file.) After the complete data is sent, it must be enabled via CMD_ENABLE_SNAPSHOT_METADATA.

u8_enable: Set to 1 to enable insertion of metadata in JPEG snapshots, 0 otherwise.

File system

Command		Reply data	Min. API version
Opcode	Data		
CMD_GET_FILECOUNT	u8_dev	u32_filecount, u32_stat	1.0
CMD_GET_FILENAME	u8_dev, u32_fileindex	str_filename, u32_stat	1.0
CMD_GET_FILENAME_SIZE	u8_dev, u32_fileindex	str_filename, u32_size, u32_stat	1.3
CMD_DELETE_FILE	u8_dev, str_filename	u32_stat	1.0
CMD_RENAME_FILE	u8_dev, str_oldname, str_newname	u32_stat	1.0
CMD_GET_FILESYSTEM_INFO	u8_dev	u32_free_kb, u32_total_kb, u32_stat	1.3
CMD_OPEN_FILE	u8_dev, u8_mode, str_pathname	u32_stat	1.3
CMD_CLOSE_FILE		u32_stat	1.3
CMD_READ_FILE_CHUNK	u8_count	u8_data[], u32_stat	1.3
CMD_WRITE_FILE_CHUNK	u8_data[]	u32_stat	1.3

u8_dev: one of enum DVR_STORAGE_DEVID (see dvr_protocol.h) [MMC=0, INTERNAL=1, EXTERNAL=2]

u32_filecount: the number of files present in the video storage directory (DCIM) on the specified device.

u32_fileindex: the zero-based index of the file whose name is to be retrieved. To enumerate all the files on the device, iterate through index 0 to file count - 1.

str_filename: the file name returned at the specified index, or the file name to be deleted. The directory used relative to the file name is the video storage directory (DCIM) on the specified device.

u32_size: the size of the file returned at the specified index.

str_oldname: the existing file to be renamed

str_newname: the new name of the file to be renamed. If the file name is already in use, the error returned will be DVR_ERR_ILLEGAL_ARGVAL.

u32_free_kb: available number of kilobytes (1000 bytes) on the device indicated.

u32_total_kb: total size of the device in kilobytes (1000 bytes).

str_pathname – the file path name to be opened, relative the root directory, and must use the forward-slash character “/” as directory separators. For example, “DCIM/image.jpg”.

u8_mode: For reading, set to 0. For writing, set to 1.

u8_count: The number of bytes to read. This is limited to the maximum size of a reply packet (size TBD.)

u8_data: Data in reply to a READ command, or data to be written in a WRITE command.

Notes on CMD_OPEN_FILE: Only one file may be opened at a time. If a new file is opened, any previously opened file will be closed. If a file is not closed before power down, data may be lost. The READ/WRITE command must match the mode used to open the file. The READ command may return zero-length data at the end of the file, or DVR_ERR_STORAGE if the storage device is removed. Similarly, the WRITE command may DVR_ERR_STORAGE if the storage device returns an error, perhaps if the storage becomes full or the file size would exceed FAT32 limitation.

Video overlay

Command		Reply data	Min. API version
Opcode	Data		
CMD_SET_OSD_TEXT	u8_region, str_txt	u32_stat	1.0
CMD_SET_OSD_POSITION	u8_region, u16_top, u16_left	u32_stat	1.0
CMD_SET_OSD_ENABLE	u8_region, u8_enable	u32_stat	1.0
CMD_SET_OSD_BACKGROUND	u8_region, u8_bg	u32_stat	1.0
CMD_SET_OSD_CLOCK_FORMAT	u8_datefmt, u8_timefmt	u32_stat	1.0
CMD_GET_OSD_CLOCK_FORMAT	–	u8_datefmt, u8_timefmt, u32_stat	1.0
CMD_GET_OSD_SETTINGS	u8_region	u8_enable, u8_bg, u16_top, u16_left, u32_stat	1.0
CMD_OSD_IMAGE_CREATE	u8_region, u16_size	u32_stat	1.0
CMD_OSD_IMAGE_UPLOAD_CHUNK	u8_region, u16_addr, u8_data()	u32_stat	1.0
CMD_OSD_IMAGE_INSTALL	u8_region, u8_imagetype	u32_stat	1.0

u8_region: integer range 0 to 15, each region can be positioned independently. Note: regions 0 to 13 are shared with DVR Secondary Overlay Text, each line uses one region. API version 1.2 allows value 16 to address the Primary Overlay Text region.

str_txt: text to be displayed. Use '^n' to insert newline, '^d' for date, '^t' for time, '^e' for encoder 0, '^f' for encoder 1. API version 1.2 adds additional virtual encoders '^q' for encoder 2, '^w' for encoder 3. Note: unscaled encoder counts are displayed as integers, otherwise with 2 decimal places (.00) for encoders 0,1 and 1 decimal place (.0) for encoders 2,3.

u16_top, u16_left: the vertical and horizontal position for the upper left hand corner of the text region.

u8_enable: set the osd enabled and size [0=off 1=8x14 2=16x16 3=19x32(HD only)]

u8_bg: text region color and background [0=white/black 1=white/transparent 2,3=not available 4=black/transparent 5=black/white 6=white/blue] Note: solid backgrounds (not transparent) offer better performance. Too many or too large overlays may impact frame rate and DVR responsiveness.

u8_datefmt: one of enum DVR_DATE_FORMAT (see dvr_protocol.h for details)

u8_timefmt: one of enum DVR_TIME_FORMAT (see dvr_protocol.h for details)

u16_size: size of image to upload, in bytes.

u16_addr: byte address of the upload chunk.

u8_data: upload chunk data.

u8_imagetype: one of enum DVR_OSD_IMAGETYPE (see dvr_protocol.h for details) [0=PNG 1=BMP]

Video adjust

Command		Reply data	Min. API version
Opcode	Data		
CMD_SET_VIDEO_ADJUST	u8_param_id, u8_setting	u32_stat	1.0
CMD_GET_VIDEO_ADJUST	u8_param_id	u8_setting, u32_stat	1.0

u8_param_id: one of enum DVR_VIDEO_ADJ_ID (see dvr_protocol.h for details)

u8_setting: value to be set or retrieved

Enum index	Param Name	Param value range and defaults	Supported Model
0	ADJ_BRIGHTNESS	0 to 255, default 128	401X, 4023
1	ADJ_AUTOBRIGHT	0=off 1=on, default 1	401X
2	ADJ_CONTRAST	0 to 255, default 128	401X, 4023
3	ADJ_HUE	(s8) -128 to 127, default 0	401X, 4023
4	ADJ_SATURATION	0 to 255, default 128	401X, 4023
5	ADJ_CHROMA_FILTER	3=Low 4=High, default 4	401X
6	ADJ_LUMA_FILTER	3=Low 0=High, default 3	401X
7	ADJ_COMB_FILTER	0=Off 1=On 3=Adaptive, default 3	401X

Video configuration

Command		Reply data	Min. API version
Opcode	Data		
CMD_SET_VIDEO_CFG	u8_format, u8_deint, u8_decimate	u32_stat	1.0
CMD_GET_VIDEO_CFG	–	u8_format, u8_deint, u8_decimate, u32_stat	1.0

u8_format: one of enum DVR_VIDEO_FORMAT (see dvr_protocol.h for details) [0=VIDFMT_NTSC, 1=VIDFMT_PAL] Note: applicable to model 4011/4013 only. HD models use auto-detect input format.

u8_deint: enable/disable deinterlace. 0=interlaced capture, 1=deinterlace (interpolate)

u8_decimate: set frame skipping, 0=no skip, 1=skip every other frame, 2=encode every 3rd frame, 4=every 5th frame

Audio

Command		Reply data	Min. API version
Opcode	Data		
CMD_SEL_AUDIO_INPUT	u8_src		1.0
CMD_SET_VOLUME	u8_chansel, u8_vol	u32_stat	1.0
CMD_SET_MUTE	u8_chansel, u8_mute	u32_stat	1.0
CMD_SET_AUTOVOL	u8_chansel, u8_agc	u32_stat	1.0
CMD_GET_AUDIO_SETTINGS	–	u8: src, lvol, rvol, lmute, rmute, lagc, ragc, u32_stat	1.0

u8_src: one of enum DVR_AUDIO_INPUT (see dvr_protocol.h for details) [0=AUDIOIN_MIC, 1=AUDIOIN_LINE, 2=AUDIOIN_DIGITAL (SDI only)]

u8_chansel: one of enum DVR_AUDIO_CHAN (see dvr_protocol.h for details) [1=AUDIOCHAN_LEFT, 2=AUDIOCHAN_RIGHT, 3=AUDIOCHAN_BOTH]

u8_vol: integer range 0 to 119. (It is actually gain, not a volume control, so start at 0 and increase as needed. Too large may result in clipping and crackly audio.)

u8_mute: enable or disable audio channel, 0=unmute, 1=muted

u8_agc: automatic gain control, 0=disable, 1=enable

Incremental encoder

Command		Reply data	Min. API version
Opcode	Data		
CMD_SET_ENCODER_COUNTS	u8_chan, u32_counts	u32_stat	1.0
CMD_GET_ENCODER_COUNTS	u8_chan	u32_counts, u32_stat	1.0
CMD_SET_ENCODER_SCALAR	u8_chan, dbl_scalar	u32_stat	1.0
CMD_USE_OFFBOARD_ENCODER	u8_chan, u8_offboard	u32_stat	1.0
CMD_GET_ENCODER_SETTINGS	u8_chan	dbl_scalar, u8_offboard, u32_stat	1.0

u8_chan: encoder channel 0 or 1. API version 1.2 adds channels 2 and 3 as virtual encoders (always offboard.)

u32_counts: raw encoder counts range 0 to $2^{32}-1$

dbl_scalar: scale multiplier applied to raw encoder count before being displayed in overlays

u8_offboard: when set to 1, calling this function will disable internal encoders. All encoder count changes must then be done using CMD_SET_ENCODER_COUNTS. Once set, the internal encoders cannot be re-enabled (until power-cycle.)

Menu system

Command		Reply data	Min. API version
Opcode	Data		
CMD_MENU_DISPLAY	u8_enable	u32_stat	1.1
CMD_MENU_MOVE_CURSOR	u8_up	u32_stat	1.1
CMD_MENU_ENTER	–	u8_menuid, u8_executable, str_menutext, u32_stat	1.1
CMD_MENU_EXIT	–	u32_stat	1.1

u8_enable: 0=show menu interface, 1=hide menu interface (exit out all menus)

u8_up: 0=move cursor down, 1=move cursor up

u8_menuid, u8_executable, str_menutext: not implemented

Status flags

A group of DVR status flags (`u32_stat`) is included at the end of all reply messages and in event messages of type `DVR_EVENT_STATUS_CHANGE`:

Status flag	Description	Min. API version
<code>DVR_STAT_EVENTS_ENABLED</code>	Asynchronous event reporting enabled	1.0
<code>DVR_STAT_KBD_ATTACHED</code>	Keyboard detected	1.0
<code>DVR_STAT_MEM_ATTACHED</code>	Memory stick mounted	1.0
<code>DVR_STAT_MENU_OPEN</code>	Menu displayed	1.0
<code>DVR_STAT_PLAY_PAUSED</code>	Playing paused	1.0
<code>DVR_STAT_PLAYING</code>	Playing (may be paused or running)	1.0
<code>DVR_STAT_VIEW_SNAPSHOT</code>	Snapshot is being viewed	1.0
<code>DVR_STAT_RECORD_PAUSED</code>	Recording paused	1.0
<code>DVR_STAT_RECORDING</code>	Recording (may be paused or running)	1.0
<code>DVR_STAT_SERVER_SHUTDOWN</code>	DVR is preparing to shut down	1.0
<code>DVR_STAT_VIDEO_LOCK</code>	Input video detected	1.0
<code>DVR_STAT_WIFI_ENABLED</code>	WiFi enabled	1.0
<code>DVR_STAT_WRITE_ERROR</code>	Disk full or write error	1.0

Example packets

The following examples show encoded command (c) and reply (r) packets for various DVR functions. Each packet is expressed as a sequence of hexadecimal bytes and includes a packet delimiter (00) at the end. Note that in actual operation, the DVR status flags (`u32_stat`) at the end of every reply packet will indicate the current DVR status and therefore may differ from the values shown below.

Set overlay region 0 text to "Overlay text"

```
c: 01 02 3C 0D 4F 76 65 72 6C 61 79 20 74 65 78 74 02 FD 00  
r: 02 40 01 01 01 01 02 C0 00
```

Position overlay region 0 top-left corner at (x, y) = (50, 100)

```
c: 01 02 3D 02 64 02 32 02 2D 00  
r: 02 40 01 01 01 01 02 C0 00
```

Set channel 0 incremental encoder counts to 0x12345678

```
c: 01 02 78 06 78 56 34 12 74 00  
r: 02 40 01 01 01 01 02 C0 00
```

Read channel 0 incremental encoder counts, which currently are 0x12345678

```
c: 01 02 79 02 87 00  
r: 02 40 05 78 56 34 12 01 01 01 02 AC 00
```

Client implementation

When developing code for a DVR client, software designers may use Sensoray's API or implement the DVR protocol directly.

As a design aid for the latter case, Sensoray offers a basic serial client example (`simple_sdvr_codec.c`). This is a complete working example except for low-level serial communication functions `send_byte()` and `recv_byte()`, which must be supplied by the developer.

Architecture

The basic serial client example only has a command endpoint (no event endpoint), which is handled with a single thread. Since there is no event endpoint, the client must poll the DVR to detect asynchronous DVR state changes:

More sophisticated clients can use multiple threads to efficiently monitor asynchronous DVR state changes in real time, eliminating both the overhead and latency of polling. For example, this architecture is employed in Sensoray's API for DVRs:

API

A standard API is available which is compatible with all Sensoray DVRs. The API includes a complete set of DVR management functions that implement DCP.

A DVR command is invoked by calling the associated API function; this causes an encoded command packet to be sent to the DVR. Upon receiving the reply packet, the API function will decode and extract relevant information and return it to the caller. A blocking function is provided to receive asynchronous event messages.

DVR command functions are thread-safe, thus ensuring conflict-free operation when multiple threads simultaneously communicate with a DVR. This allows a remote client to, for example, upload overlay graphics in the background (via low-priority thread) while maintaining GUI responsiveness (via higher-priority DVR control thread).

The API employs UDP sockets as endpoints. Each UDP datagram encapsulates one message. Consequently, the API is inherently compatible with DVRs that communicate over Ethernet.

Interface bridge

Some DVR models communicate with remote clients via interfaces such as USB or serial COM, which don't employ sockets. Sensoray provides client-side "bridge" software for these, which transparently redirect the API sockets to the communication interface.

Example

This simple example shows how a remote client can use Sensoray's standard API and serial COM bridge to monitor and control a DVR. Note that error checking is omitted for clarity; production software should always perform error checking and handling.

```
#include "dvrapi.h"
#include "dvr_ser_bridge.h"

#define DVRID    0 // DVR identifier in range [0:31]
#define COMPORT 1 // Serial COM port identifier

int example(void)
{
    u32 stat;           // DVR status
    char clip[100];    // video clip filename

    bridge_api_open();           // init bridge
    dvr_api_open(0);             // init standard API
    bridge_dev_open(DVRID, COMPORT); // open bridge to DVR
    dvr_open(DVRID, NULL, 200);  // enable client access to DVR

    dvr_record_start(dvrid, DVR_STORAGE_INTERNAL, // record a 5-second clip
        RECFMT_MP4, 2500, clip, sizeof(clip), &stat);
    printf("recording to %s\n", clip);
    sleep(5000);
    dvr_record_stop(DVRID, &stat);

    dvr_play(DVRID, DVR_STORAGE_INTERNAL, clip, &stat); // start playback
    dvr_api_close(); // shutdown and close API
    bridge_api_close(); // shutdown and close bridge
}
```

Simple API

A simplified API is provided for use on microcontrollers where a full socket bridge may not be feasible. A small C library is provided, with stub functions for opening/closing and reading/writing to the serial port. The packet encoding/decoding functions work on memory in-place, and transmit to the serial port directly.

Modify `dvr_simple.c` and replace the "open_com_port", "close_com_port", "send_byte" and "recv_byte" functions to use the microcontroller serial port. The current functions target Linux for testing and use `/dev/ttyS0`.

The simplified function interface in "dvr_simple.h" removes the "dvid" and "status" parameters from most functions. This assumes that the embedded system will be connected to only a single DVR board. The last status can still be retrieved using the function `dvr_get_last_status()`. The simplified API is not thread-safe.

The event reporting is not implemented at this time.

Example

This simple example shows how a remote client can use Sensoray's standard API and serial COM bridge to monitor and control a DVR. Note that error checking is omitted for clarity; production software should always perform error checking and handling. Additionally, the client code should first check the API version and repeat until a valid response is received, to verify that the DVR is powered up and ready for communication before sending further commands.

```
#include "dvr_simple.h"

int main(int unused)
{
    int errcode;
    u32 status;      // dvr status flags
    char filename[MAX_FILENAME_SIZE];

    open_com_port();

    dvr_record_start(BOTH, RECFMT_MP4, 0 /*default*/, filename);
    printf("recording started, filename=%s\n", filename);
    sleep(5); // record for 5 seconds
    dvr_record_stop();

    dvr_play_start(INTERNAL, filename);
    sleep(5); // play back full duration
    dvr_play_stop();

    close_com_port();
}
```

DateTime Conversion Functions

These functions allow conversion between the `u32_clock` values used in the DATETIME commands, and the year/month/day/hour/minute/seconds in a Julian date.

```
static const unsigned int daysSinceJan1st[2][13] = {
    {0, 31, 59, 90, 120, 151, 181, 212, 243, 273, 304, 334, 365}, // 365 days, non-leap
    {0, 31, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335, 366} // 366 days, leap
};

unsigned int julian_to_datetime(int year, int month, int day, int hour, int minute, int seconds)
{
    int leap = !(year % 4) && (year % 100 || !(year % 400));
    int yday = daysSinceJan1st[leap][month-1] + day-1;
```



```

year -= 1900; // adjust to tm_year

return seconds + minute*60 + hour*3600 + yday*86400 +
    (year-70)*31536000 + ((year-69)/4)*86400 -
    ((year-1)/100)*86400 + ((year+299)/400)*86400;
}

// https://stackoverflow.com/a/11197532
void datetime_to_julian(unsigned int datetime, int *year, int *month, int *day, int *hour, int
*minute, int *seconds)
{
    unsigned long long sec;
    unsigned int wday, quadricentennials, centennials, quadrennials, annuals;
    unsigned int leap, yday, mday, mon;

    // Re-bias from 1970 to 1601:
    // 1970 - 1601 = 369 = 3*100 + 17*4 + 1 years (incl. 89 leap days) =
    // (3*100*(365+24/100) + 17*4*(365+1/4) + 1*365)*24*3600 seconds
    sec = datetime + 11644473600ULL;

    wday = (sec / 86400 + 1) % 7; // day of week

    // Remove multiples of 400 years (incl. 97 leap days)
    quadricentennials = sec / 12622780800ULL; // 400*365.2425*24*3600
    sec %= 12622780800ULL;

    // Remove multiples of 100 years (incl. 24 leap days), can't be more than 3
    // (because multiples of 4*100=400 years (incl. leap days) have been removed)
    centennials = (sec / 3155673600ULL); // 100*(365+24/100)*24*3600
    if (centennials > 3) {
        centennials = 3;
    }
    sec -= centennials * 3155673600ULL;

    // Remove multiples of 4 years (incl. 1 leap day), can't be more than 24
    // (because multiples of 25*4=100 years (incl. leap days) have been removed)
    quadrennials = (sec / 126230400); // 4*(365+1/4)*24*3600
    if (quadrennials > 24) {
        quadrennials = 24;
    }
    sec -= quadrennials * 126230400ULL;

    // Remove multiples of years (incl. 0 leap days), can't be more than 3
    // (because multiples of 4 years (incl. leap days) have been removed)
    annuals = (sec / 31536000); // 365*24*3600
    if (annuals > 3) {
        annuals = 3;
    }
    sec -= annuals * 31536000ULL;

    // Calculate the year and find out if it's leap
    *year = 1601 + quadricentennials * 400 + centennials * 100 + quadrennials * 4 + annuals;
    leap = !(*year % 4) && (*year % 100 || !(*year % 400));

    // Calculate the day of the year and the time
    yday = sec / 86400;
    sec %= 86400;
    *hour = sec / 3600;
    sec %= 3600;
    *minute = sec / 60;
    sec %= 60;
    *seconds = sec;

    // Calculate the month
    for (*day = mon = 1; mon < 13; mon++) {
        if (yday < daysSinceJan1st[leap][mon])

```

```
        {
            *day += yday - daysSinceJan1st[leap][mon - 1];
            break;
        }
    }
    *month = mon;
}
```