Model 819 Windows SDK

Ver.1.0.1

Contents

Terminology

channel video input. Each 819 has 16 input channels. Multiple boards are enumerated contiguously, so that channels are addressed by a single channel number (1-based): board #1 containing channels 1-16, board #2 – channels 17-32, etc. When audio support is added, channels will be differentiated as video and audio.

stream output of model 819 in a specific format corresponding to a channel. One channel may produce multiple streams, e.g. H.264 primary and secondary streams with different settings, JPEG stream, etc.

capture node a set of streams produced by one or more channels combined to facilitate capture within a single application thread. Each node receives an individual signal from the driver once data is available for any of the attached streams. Depending on the desired application architecture streams may be combined into nodes to simplify data processing. For example, one node may combine archived streams and be handled in the archiving thread. Another node may combine streams that are being presented live and be handled by a streaming thread. Extreme cases of one thread per stream or one thread handling all streams are also possible, though may not be practical.

Limitations

JPEG capture is not supported.

Data Types

For complete reference to 819 data types please refer to s819.h.

Captured resolutions

```
typedef enum {
    RES_QCIF = 0,
    RES_CIF,
    RES_D1,
    RES_MAX,           //for array sizing only
} S819_RESOLUTION;
```

	NTSC	PAL
RES_QCIF	176x120	176x144
RES_CIF	352x240	352x288
RES_D1	704x480	704x576

TV standards

```
typedef enum {
    TVS_PAL,
```

```
    TVS_NTSC,
    TVS_MAX,        //for array sizing
} S819_TVSTANDARD;
```

Bitrate control mode

```
typedef enum {
    BITRATE_H264_NO_RC,     // Var with max limit
    BITRATE_H264_CBR,       //constant bitrate
    BITRATE_H264_VBR,       //variable bitrate
    BITRATE_MAX,           //for array sizing only
} S819_BR_CTRL;
```

TBC.

Capture mode

```
typedef struct {
    S819_TVSTANDARD     tvs;
    int                 bright;
    int                 contrast;
    int                 saturation;
    int                 hue;
    int                 sharpness;
    int                 reserved[8]
} CHAN_MODE;
```

bright brightness control. Valid values: -128 to 127. Default 0.

contrast contrast control. Valid values: 0 to 255. Default 0x64. Adjustment rate – 1% per step.

saturation saturation. The range of adjustment is 0 to 200%. Default value of 0x80 corresponds to 100%.

hue color hue (for NTSC input only). Range: -128 to 127. Default 0. Two lower bits have no effect. Positive values increase greenish tones, negative – purplish.

sharpness *Future use*

reserved *Future use*

Stream mode

```
typedef struct {
    S819_RESOLUTION     resolution;
    int                 fps;
    int                 gopSize;
    S819_BR_CTRL        brCtrl;
    int                 bitrate;
    int                 reserved[8];
} STREAM_MODE;
```

resolution resolution of the captured stream.

fps capture rate (frames per second). Range: 1-25 (PAL), 1-30 (NTSC) in 1 fps increments. This parameter does not affect JPEG capture rate, which is fixed at 2 fps.

interlace interlacing mode.

gopSize GOP size. 1 to 256. For H.264 streams only.

brCtrl Bitrate control mode. For H.264 streams only.

bitrate Requested bitrate (in bits per second). Please refer to s819.h for allowed ranges. Irrelevant for JPEG stream, JPEG quality is fixed.

reserved *Future use*

Stream type

```
typedef enum {
    STREAM_H264_PRI = 0,
    STREAM_H264_SEC,
    STREAM_JPEG,
    STREAM_YUV,
    STREAM_AUDIO,
    STREAM_MV,
    STREAM_MAX
} S819_STREAM_TYPE;
```

STREAM_H264_PRI primary H.264 stream.

STREAM_H264_SEC secondary H.264 stream. Resolution must be lower than that of the primary stream.

STREAM_JPEG JPEG stream.

STREAM_YUV preview stream

STREAM_AUDIO audio stream. For return type from waitbuffer only.

STREAM_MV future use

Capture channel

```
typedef struct {
    int                 ctrl;
} CONTROL;
```

Reserved for future use.

```
typedef struct {
    CHAN_MODE           cMode;  //channel capture mode
    STREAM_MODE         psMode; //primary H.264 stream
    STREAM_MODE         ssMode; //secondary H.264 stream
    STREAM_MODE         jsMode; //JPEG stream
    STREAM_MODE         usMode; //Uncompressed preview stream
    CONTROL             ctrl;   //reserved
```

```
    int                 reserved[32];
} S819_CHAN;
```

cMode channel capture mode. Applies to all streams captured from a channel.
psMode primary H.264 stream mode.
ssMode secondary H.264 stream mode.
JsMode JPEG stream mode.
UsMode Uncompressed preview stream
reserved *Reserved fields*

Capture buffer

```
typedef struct {
    int                 chan;
    S819_STREAM_TYPE    stream;
    char                *buf;
    int                 length;
    unsigned int        frm;
    unsigned int        pts;
    void                *rsv;
    unsigned int        ftype;
    int                 reserved[8];
} BUFFER;
```

chan channel number, 1 through maximum available channels based on the number of boards detected, but no more than 64.
stream type of stream captured into buffer.
buf pointer to buffer data.
length length of valid data currently in buffer, bytes.
frm Future use.
pts Future use.
rsv Reserved. Do not modified
ftype Frame type (For H.264 streams only. FTYPE_SPS, FTYPE_PPS, FTYPE_IFRAME, FTYPE_PFRAME, FTYPE_BFRAME)
reserved Additional reserved fields.

Motion detection data

```
Motion data is organized in 192 cells consisting of 16 columns and 12
rows.  The first 24 bytes of odddata and evendata is a bitmask where
a '1' indicates that motion was detected in the corresponding cell.
typedef struct {
    unsigned long   lTimeStamp;
    unsigned char   odddata[32*4];
```

```
    unsigned char   evendata[32*4];
} S819_MVFLAG_DATA;
```

lTimeStamp	timestamp associated with the captured motion data.
odddata	motion data from odd frames.
evendata	motion data from even frames.

Miscellaneous types

```
typedef int     ECODE;              //error code
typedef void    *HCNODE;            //capture node handle
```

SDK Functions

All functions returning an error code (type ECODE) return ECODE_OK, if success. Any other value indicates an error.

S819_Enumerate

```
ECODE S819_Enumerate(int *nChans)
```

Enumerates existing 819 boards and sets nChans to the number of available capture channels. This function needs to be called before any other SDK function.

S819_SetMode

```
ECODE S819_SetMode(int chan, S819_CHAN *s819chan)
```

If the argument *chan* is between 1 and the number of channels detected by S819_Enumerate, the function sets capture mode for a specified channel (*chan*) based on the values set in *s819chan*.
If *chan* = 0 the structure pointed to by *s819chan* is filled with default values.

S819_CreateCnode

```
HCNODE S819_CreateCnode(void)
```

Creates a new empty capture node. Streams are attached to a node by using *S819_AttachStreams* function.
Returns: a capture node handle or NULL in case of an error.

S819_DeleteCnode

```
void S819_DeleteCnode(HCNODE hCnode)
```

Deletes a capture node.

S819_AttachStreams

```
ECODE S819_AttachStreams(HCNODE hCnode, int chan, int streams)
```

Attaches stream(s) produced by a selected channel to a capture node.

hCnode	a valid capture node handle.
chan	selected channel number (1-based).

streams streams bitmask. To select a stream, set a bit of the parameter *streams* to 1 by using an SMASK macro. For example, streams = SMASK(STREAM_H264_PRI); or streams = SMASK(STREAM_H264_PRI) | SMASK(STREAM_H264_SEC);

S819_DetachStreams

```
ECODE S819_DetachStreams(HCNODE hCnode)
```

Detaches stream(s) from capture node.

hCnode a valid capture node handle.

S819_StartStreams

```
ECODE S819_StartStreams(HCNODE hCnode, int chan, int streams)
```

Starts capture of a combination of streams attached to a capture node. Other streams that may be attached to the same capture node are not affected.

hCnode a valid capture node handle.

chan selected channel number (1-based).

streams streams bitmask corresponding to desired stream combination. To select a stream, set a bit of the parameter *streams* to 1 by using an SMASK macro. For example, streams = SMASK(STREAM_H264_PRI); or streams = SMASK(STREAM_H264_PRI) | SMASK(STREAM_H264_SEC);

S819_StopStreams

```
ECODE S819_StopStreams(HCNODE hCnode, int chan, int streams)
```

Stops capture of a combination of streams attached to a capture node. Other streams that may be attached to the same capture node are not affected.

hCnode a valid capture node handle.

chan selected channel number (1-based).

streams streams bitmask corresponding to desired stream combination. To select a stream, set a bit of the parameter *streams* to 1 by using an SMASK macro. For example, streams = SMASK(STREAM_H264_PRI); or streams = SMASK(STREAM_H264_PRI) | SMASK(STREAM_H264_SEC);

S819_StartAll

```
ECODE S819_StartAll(HCNODE hCnode)
```

Starts capture of all streams attached to a selected capture node.

S819_StopAll

```
ECODE S819_StopAll(HCNODE hCnode)
```

Stops capture of all streams attached to a selected capture node.

S819_WaitBuffer

```
ECODE S819_WaitBuffer(HCNODE hCnode, BUFFER *buffer)
```

A blocking function that waits for data to become available for a selected capture node. The function times out if the data is not ready in 5 seconds and returns an error code. If the data is available the function fills in the structure pointed to by *buffer* with data parameters. A buffer needs to be returned back to the driver by calling S819_ReleaseBuffer.

S819_ReleaseBuffer

```
ECODE S819_ReleaseBuffer(BUFFER *buf)
```

Returns a buffer to the driver. It is essential to call this function after the data has been handled by the application. Once the buffer is released members of the `*buf` become invalid.

S819_Close

```
void S819_Close (void)
```

SDK cleanup. Must be called before an application terminates. Must be the last call to the SDK.

S819_SetXPSwitch

```
ECODE S819_SetXPSwitch (int brd, int inp, int out)
```

Controls output video crosspoint switch. Connects selected input to selected output.

brd board index (0-based).
inp selected input channel number (1-based).
out selected output (0-based).

S819_SetXPOut

```
ECODE S819_SetXPOut (int brd, int mask)
```

Enables/disables outputs of video crosspoint switch.

brd board index (0-based).

mask *bits 0-3 control outputs 0-3 (1 – enabled, 0 – disabled).*

S819_RegMVFlagCB

```
ECODE S819_RegMVFlagCB (S819_MVFlagCallBack funCallBack, void
*pContext)
```

Registers a callback function for motion detection. Each frame is divided into 192 cells consisting of 16 columns and 12 rows. If motion is detected in any one or more of these cells the callback function will fire.

funCallBack callback function to use. Callback function should be defined as follows: "void callback_name(unsigned int channel, S819_MVFLAG_DATA *pdata, void *pContext)"

pContext *context pointer*

S819_StartMVFlag

```
ECODE S819_StartMVFlag (HCNODE hCnode)
```

Starts capturing motion detection flags. Must have previously registered a callback function with S819_RegMVFlagCB.

hCnode a valid capture node handle.

S819_StopMVFlag

```
ECODE S819_StartMVFlag (HCNODE hCnode)
```

Stops capturing motion detection flags.

hCnode a valid capture node handle.

S819_GetMotionSensitivity

```
ECODE S819_GetMotionSensitivity (int chan, unsigned long *mvsens)
```

Gets the sensitivity setting for motion detection.

chan selected channel number (1-based).

mvsens a pointer to an unsigned long that is to receive the current motion sensitivity level.

S819_SetMotionSensitivity

```
ECODE S819_SetMotionSensitivity (int chan, unsigned long mvsens)
```

Sets the sensitivity level for motion detection.

chan selected channel number (1-based).

mvsens value of the motion sensitivity to set. Valid values are from 0-15 where 0 is most sensitive and 15 is least sensitive. Recommended values are from 3-8, default value is 3.

S819_GetSignalStatus

ECODE S819_GetSignalStatus (int chan, int *signal, int *vidsys)

Gets video connection status.

chan selected channel number (1-based).

signal pointer to the video signal status; 0 = video connected, 1 = no video connected.

Vidsys pointer to video system; 0 = NTSC, 1 = PAL.

MP4 Multiplexing Functions

The 819 SDK was designed to be as flexible as possible for end users. Some users may want to save the encoded data to the .MP4 (ISO/IEC 14496-14) file format. A separate optional DLL, sraymp4.dll, is provided for this. The functions for writing .mp4 files are described below. Sraymp4.dll is licensed for use only with Sensoray products.

mp4_writer_create

```
MP4API muxer_handle MP4CC mp4_writer_create();
```

Creates an instance of the mp4 muxer.

mp4_delete

```
MP4API void MP4CC mp4_delete(muxer_handle mux);
```

Deletes instance of mp4 muxer.

mp4_file_pointer

```
MP4API void MP4CC mp4_file_pointer(muxer_handle mux, FILE *file);
```

Gives file pointer to write mp4 file to. File handle should be opened as read-writable.

mp4_file_name_ascii

```
MP4API void MP4CC mp4_file_name_ascii(muxer_handle mux, char *fname);
```

Gives file name to write mp4 file to. File will be created by the DLL. Only one of mp4_file_pointer, mp4_file_name_ascii or mp4_file_name_unicode should be used.

mp4_file_name_unicode

```
MP4API void MP4CC mp4_file_name_ascii(muxer_handle mux, wchar_t
*fname);
```

Unicode version. Gives file name to write mp4 file to. File will be created by the DLL. Only one of mp4_file_pointer, mp4_file_name_ascii or mp4_file_name_unicode should be used.

mp4_video_track_create

```
MP4API unsigned int MP4CC mp4_video_track_create(muxer_handle mux,
unsigned int fourcc, unsigned int timebase, unsigned int
default_duration, unsigned int empty_edit_duration, unsigned int
width, unsigned int height, unsigned int profileIDC, unsigned int
levelIDC);
```

Create video track for mp4 file. Fourcc should be 'avc1' (use UUID provided in header or in C# wrapper). timebase is normally 90000. default_duration depends on the frame rate and the video system. It is in units of the timebase. For example PAL 25fps would have a duration of 3600. NTSC would have a duration of 3003. The empty_edit_duration may be set to 0. The width and height are set based on the chosen resolution. profileIDC and levelIDC should be set to 100 and 31 respectively. Returns the video track number to use as parameter to mp4_write_track.

mp4_write_track

```
MP4API unsigned int MP4CC mp4_write_track(muxer_handle mux, unsigned int trackno, unsigned char *buf, unsigned int len, unsigned int duration);
```

Writes data for the video track. Duration is usually the same as the default duration used in creating the track (3600 for 25fps PAL).

mp4_write_movie

```
MP4API unsigned int MP4CC mp4_write_movie(muxer_handle mux)
```

Writes the mp4 moov atom after creating tracks. Call when done writing the mp4 file and before calling mp4_delete.

Demo Applications and Examples

C# Visual Studio .NET Demo

A full feature demo application written in C# .NET is provided. This includes a class file to communicate with all the functions in the DLL. The different channels may be controlled by right-clicking on the channel of interest.

C source code

The following code demonstrates the ease of use for the SDK.

JPEG capture

Opens channel 2 in JPEG streaming mode and saves the first 30 JPEG files to disk (in the temp directory).

```
int _tmain(int argc, _TCHAR* argv[])
{
        int channels;
        S819_CHAN s819chan;
        HCNODE hcnode;
        BUFFER buf;
        int rc;
        FILE *fin;
        int i;
        char name[60];

        rc = S819_Enumerate(&channels);
        S819_SetMode (0, &s819chan);
        hcnode = S819_CreateCnode();
        S819_AttachStreams(hcnode, 2, SMASK(STREAM_JPEG));
        S819_StartStreams(hcnode, 2, SMASK(STREAM_JPEG));
        for (i = 0; i < 30; i++) {
                S819_WaitBuffer (hcnode, &buf);
                sprintf(name, "c:\\temp\\test%d.jpg", i);
                fin = fopen(name, "wb");
                fwrite(buf.buf, 1, buf.length, fin);
                fclose(fin);
        }
        S819_StopStreams(hcnode, 2, SMASK(STREAM_JPEG));
        S819_DetachStreams(hcnode);
        S819_DeleteCnode(hcnode);
        S819_Close();
        return 0;
}
```

H.264 Recording

Saves one file as raw H.264 output (test0.264) and another as ISO 14996-14 compliant .mp4.

```
int _tmain(int argc, _TCHAR* argv[])
{
	int channels;
	S819_CHAN s819chan;
	HCNODE hcnode;
	BUFFER buf;
	int rc;
	FILE *fin;
	int i = 0;
	char name[60];
	int vtrack;
	muxer_handle m;
	sprintf(name, "c:\\temp\\test%d.264", i);
	// for H264 file
	fin = fopen(name, "wb");
	m = mp4_writer_create();
	mp4_file_name_ascii(m, "c:\\temp\\test.mp4");
	vtrack = mp4_video_track_create(m, AVC1, 90000, 3600, 0, 352, 240, 100, 31);
	rc = S819_Enumerate(&channels);
	S819_SetMode(0, &s819chan);
	s819chan.psMode.resolution = RES_CIF;
	S819_SetMode(1, &s819chan);
	hcnode = S819_CreateCnode();
	S819_AttachStreams(hcnode, 1, SMASK(STREAM_H264_PRI));
	S819_StartStreams(hcnode, 1, SMASK(STREAM_H264_PRI));
	for (i = 0; i < 300; i++) {
		unsigned char *pdata;
		S819_WaitBuffer (hcnode, &buf);
		pdata = (unsigned char *) buf.buf;
		if (buf.stream == STREAM_AUDIO) {
			S819_ReleaseBuffer(&buf);
			continue;
		}
		if (buf.stream != STREAM_H264_PRI) {
			S819_ReleaseBuffer(&buf);
			continue;
		}
		fwrite(buf.buf, 1, buf.length, fin);
		mp4_write_track(m, vtrack, (unsigned char *) buf.buf, buf.length, 3003);
		S819_ReleaseBuffer(&buf);
	}
	mp4_write_movie(m); // write the header into overflow buffer
	mp4_delete(m);
	fclose(fin);
	S819_StopStreams(hcnode, 2, SMASK(STREAM_H264_PRI));
	S819_DetachStreams(hcnode);
	S819_DeleteCnode(hcnode);
	S819_Close();
	return 0;
}
```