SR API

Manual

SmartRay GmbH
Bgm.-Finsterwalder-Ring 12
D-82515 Wolfratshausen
www.smartray.de

SR_API Version	3.6.9
Docu. Version	1.7.0
Docu. Date	2013-07-26

Table of Contents

1 Introduction 1

2 Files You Need 1

3 Objects / Structures 2

- 3.1 Camera Object 2
- 3.2 3D – World Coordinates 4

4 **Functions** 5

- 4.1 SR_API_GetErrorString 5
- 4.2 SR_API_GetAPIVersion 5
- 4.3 SR_API_Init 5
- 4.4 SR_API_Exit 6
- 4.5 SR_API_StartCameraConnectionManagement 6
- 4.6 SR_API_StopCameraConnectionManagement 6
- 4.7 SR_API_StartCam 7
- 4.8 SR_API_StopCam 7
- 4.9 SR_API_SetLaser 8
- 4.10 SR_API_SetOutput 8
- 4.11 SR_API_GetInput 9
- 4.12 SR_API_ReadCamParsFromFile 9
- 4.13 SR_API_SendParsToCam 9
- 4.14 SR_API_RegisterUserCB 10
- 4.14.1 Callback function SRCB_CAMDATAMODE 10
- 4.14.2 Callback function SRCB_CAMDATAEXTMODE 11
- 4.15 SR_API_LoadLutFromFile 12
- 4.16 SR_API_SaveLutToFile 12
- 4.17 SR_API_ ReadLutInfoFile 13
- 4.18 SR_API_CreateWorldData 14
- 4.19 SR_API_RemapWorldToMatrix 15
- 4.20 SR_API_WorldtoImage 16

5 Administration 17

- 5.1 SR_API_Update_Cam 17

5.2 SR_API_ChangeIp 17
5.3 SR_API_GetMACAdr 18
5.4 SR_API_GetCamVersion 18
5.5 SR_API_GetSensorType 19
5.6 SR_API_GetSerial 19
5.7 SR_API_GetSensorName 20
5.8 SR_API_GetSensorSize 20
5.9 SR_API_GetSensorCenterPosition 21
6 Camera Parameter 22
6.1 SR_API_SetImager 22
6.2 SR_API_GetImager 23
6.3 SR_API_SetExposure 24
6.4 SR_API_GetExposure 24
6.5 SR_API_SetTrigger 26
6.6 SR_API_GetTrigger 27
6.7 SR_API_SetReadInput 28
6.8 SR_API_GetReadInput 29
6.9 SR_API_SetModule 29
6.10 SR_API_GetModule 30
6.11 SR_API_SetParamProfile 30
6.12 SR_API_GetParamProfile 31
7 Return value and Error codes 32
8 Revision History 32

1 Introduction

The SmartRay (SR) API is made for software developers for ease of integration of SmartRay 3D Vision Sensors into custom projects. All sensor functionalities can be accessed with the help of this API software interface. The sensor data can be accessed with the user defined call back functions in the application program. This document is valid only if the application program uses SR_API 3.6.x version.

2 Files You Need

- SR_Api.dll
- SR_Api.lib
- SR_Api_public.h
- sr_api_errorcodes.h
- libgio-2.0-0.dll
- libglib-2.0-0.dll
- libgmodule-2.0-0.dll
- libgobject-2.0-0.dll
- libgthread-2.0-0.dll
- libguide40.dll
- SR_Tools.dll
- xerces-c_3_1.dll
- zlib1.dll
- cv100.dll
- cxcore100.dll
- intl.dll

3 Objects / Structures

3.1 Camera Object

Each camera needs a camera object which has to be defined and configured with TCP/IP relevant data by the user before operation can start.

```
// Kamera Beschreibung
typedef struct{
        int cam_index;
        char name[260];
        char IPAdr[17];
        int portnum;
        int command;
        unsigned char header[9];
        void *pcamdata;
        int camdatasize;
        int dataavailable;
        int connectionstate;
        int tcp_handle;
        int active;
        int alive_time;
        int archiv_active;
        int archiv_handle;
        void *lut;
        void *usercbf;
7       int digio_out[4];
        int digio_in[4];
        int laser_status;
        int laserlight;
        int fps;
        int tps;
        float Temp;
        int running;
} CAMDESC;
```

cam_index	Number of the camera. (Start counting with '0' and increase by one.)
Name	Name of camera (optional).
IPAdr	IP address of camera.
Portnum	Portnumber of camera.
command	Identifier for received data. With this value the API can identify which callback has to be called.
header	A 10Bytes header which can contain additional data information.
pcamdata	Pointer to received data.
camdatasize	Number of bytes of received data.
dataavailable	Internal flag which notifys the receipt of a data package.
connectionstate	TCP/IP connection state of the camera. The states are as follows: ● STATE_CAMERA_CLOSED ● STATE_CONNECT_TO_CAMERA ● STATE_CONNECT_TO_CAMERA_DELAYED ● STATE_CAMERA_CONNECTED
tcp_handle	Internal handle for the camera (TCP/IP) connection
active	Internal flag which notifys the general state of the camera connection management. 0: Connection management not running 1: Connection management running
alive_time	Internal Signal. Do not change!
archiv_active	Stores data from camera in camname.dat file between a start and stop command. This data file can be used for simulation purposes. 1: active 0: off
archiv_handle	Internal file handle for the datalog functionality.
lut	Pointer to the world data Look Up Table. Do not change!
usercbf	If no callback was registered or an unknown command was received then the callback function defined here will be called.
digio_out digio_in	Contains sensor's digital io status. Updated every 200ms with alive signal.
laser_status	Contains laser setting (enable/disable)
laserlight	Contains laser light status (on/off)
fps	Contains Frames per second info
tps	Trigger Per Second bit[15..0]; Triggeroverflow bit[31] = 1/0
Temp	Contains temperature info (actual not supported)
running	internal use; do not change

3.2 3D – World Coordinates

3D Weltkoordinatenpunkte werden in dieser Struktur beschrieben.

```
// 3D (real world point)
typedef struct{
        float x;
        float y;
        float z;
} SR_3DPOINT;
```

x	Position value normal to the laser plane.
y	Position value within the laser line.
z	Position value in measuring direction.

4 Functions

4.1 SR_API_GetErrorString

Returns an error string, generated out of an error code

Prototype:	
int SR_API_GetErrorString (int errorCode, char** errorString);	
Arguments:	
errorCode	Negative return code from a SR_API_...() function
errorString	String with error description
Return:	0: success –2: NULL pointer given for errorString

4.2 SR_API_GetAPIVersion

Returns API version as an argument to check version of SR_API.dll.

Prototype:	
int SR_API_GetApiVersion (char** version);	
Arguments:	
version	Pointer to SR_API version string

4.3 SR_API_Init

Use this function first to initialize the SR_API.
If the sensor sends a message it will come through this message callback.
See also 'SR API error and message codes.xls'.

Prototype:	
Int SR_API_Init (int (*statusMessage) (STATUS_MSG_ARGLST));	
Arguments:	
statusMessage	Pointer to the status message function (callback). For 'STATUS_MSG_ARGLST' see SR_API_public.h.

4.4 SR_API_Exit

When you are finished working with the SR API asserts this function to clean up the system.
Active sensor connections will not be terminated! Use SR_API_StopCameraConnectionManagement instead.

Prototype:	
int SR_API_Exit (void);	
Arguments:	
None	

4.5 SR_API_StartCameraConnectionManagement

The sensor connection management gets started.
Messages about the connection state will come over the Status Message callback.

Prototype:	
int SR_API_StartCameraConnectionManagement (CAMDESC* cam);	
Arguments:	
cam	Camera object.

4.6 SR_API_StopCameraConnectionManagement

The sensor connection management gets stopped.
Messages about the connection state will come over the Status Message callback.

Prototype:	
int SR_API_StopCameraConnectionManagement (CAMDESC* cam);	
Arguments:	
cam	Camera object.

4.7 SR_API_StartCam

The sensor starts operating. The operating mode is defined by parameters sent to the sensor or parameters stored in the sensor.

Prototype:	
	int SR_API_StartCam (CAMDESC* cam);
Arguments:	
cam	Camera object.

4.8 SR_API_StopCam

The sensor stops operating / idle states. Alive signal is still sent.

Prototype:	
	int SR_API_StopCam (CAMDESC* cam);
Arguments:	
cam	Camera object.

4.9 SR_API_SetLaser

Change operation mode of the laser and activate it.
The laser is not automatically activated after start up.

Prototype:	
`int SR_API_SetLaser (CAMDESC *cd, int pulseMode, int externalMode, int enable);`	
Arguments:	
cd	Camera object.
pulseMode	LASER_PULSED 0: Laser is synchronized with exposure time. LASER_CONTINOUS 1: Laser is always on if enabled.
externalMode	LASER_SW_ONLY 0: Laser is only software controlled. LASER_EXT_ON 1: Laser is synchronized with In0 of process interface
enable	LASER_ENABLE 1: Activates laser. LASER_DISABLE 0: Deactivates laser.

4.10 SR_API_SetOutput

Set the digital 24V outputs of the process interface.

Prototype:	
`int SR_API_SetOutput (CAMDESC *cd, int channel, int val);`	
Arguments:	
cd	Camera object.
channel	Choose which IO to set (0-3)
val	IO_HIGH 1: Output high. IO_LOW 0: Output low.

4.11 SR_API_GetInput

Read the digital 24V inputs of the sensor.

Prototype:	
int SR_API_GetInput (CAMDESC *cd, int channel, int val);	
Arguments:	
cd	Camera object.
channel	Choose which IO to read (0-3)
val	IO_HIGH 1: Output high. IO_LOW 0: Output low.

4.12 SR_API_ReadCamParsFromFile

The function reads a parameter set from file. The filename extension is '.par'. Use SR_Studio software to define your parameter set for the sensor.

Prototype:	
int SR_API_ReadCamParsFromFile (char *fileName);	
Arguments:	
fileName	Filename

4.13 SR_API_SendParsToCam

The function sends a parameter set to the sensor.

Prototype:	
Int SR_API_SendParsToCam (CAMDESC* cam);	
Arguments:	
cam	Camera object

4.14 SR_API_RegisterUserCB

Use this function to define and activate callback functions.
Received sensor data will then call the right callback. Use 'dataType' to determine the data format and 'command' to identify the received data package.

Prototype:	
`int SR_API_RegisterUserCB (int CBtype, int command, void *userCB);`	
Arguments:	
CBtype	Available callback types (depends on customer version): ● CB_RAWDATAMODE ● CB_CAMDATAMODE ● CB_CAMDATAEXTMODE ● CB_PIXELMODE
command	Identifier for received data. With this value the API can identify which callback has to be called.
userCB	User callback function pointer of selected callback type (see SR_API.h).

4.14.1 Callback function SRCB_CAMDATAMODE

Use this CB to receive liveimage or raw profile image data.

Prototype:	
`int (*ucbf_camdatamode) (CAMDESC* cam, int dataType, int startX, int height, int width, void* pData);`	
Arguments:	
cam	Camera object
dataType	Datatype: 0: 8Bit data 1: 16Bit data
startX	Same as CMOS-ROI startX; Has to be used with SR_API_createWorldData(..)
height	Number of lines of received image / profile.
width	Number of columns of received image /profile.
pData	Pointer to received image.

4.14.2 Callback function SRCB_CAMDATAEXTMODE

Use this CB to receive raw profile data + extended data.

Prototype:	
`int (*ucbf_camdataextmode) (CAMDESC* cam, int dataType, int startX, int height, int width, void* pData, int lenExtData, void* extData);`	
Arguments:	
cam	Camera object
dataType	Datatype: 0: 8Bit data 1: 16Bit data
startX	Same as CMOS-ROI startX; Has to be used with SR_API_createWorldData(..)
height	Number of lines of received image / profile.
width	Number of columns of received image / profile.
pData	Pointer to received image / profile.
lenExtData	Number of 32bit int. data in the extData pointer
extData	Extended data pointerExtended data pointer (int)extData[0] : timestamp counter (int)extData[1] : profile stamp counter (int)extData[2...] : do not use – dependent on sensor configuration

4.15 SR_API_LoadLutFromFile

The function reads a Look Up Table from a file and activates it for the given camera object.

Prototype:	
int SR_API_LoadLutFromFile (CAMDESC *cd, char *fileName, char *info);	
Arguments:	
cd	Camera object
fileName	path & filename of type '*.lut'.
info	Contains header information of lut. Max size: 1024Bytes

4.16 SR_API_SaveLutToFile

The function saves the Look Up Table in a file.

Prototype:	
int SR_API_SaveLutToFile (CAMDESC *cd, char *fileName, char *info);	
Arguments:	
cd	Camera object
fileName	Path & file name of type '*.lut'.
info	Contains header information of lut. Max size: 1024Bytes

4.17 SR_API_ ReadLutInfoFile

This functrion reads the Look up Table Information from the file.

Prototype:

```
int SR_API_ReadLutInfoFile (CAMDESC *cd,  char *fileName, char *info );
```

Arguments:

cd	Camera object
fileName	Path & file name of type '*.lut'.
info	Returns the header information of lut. Max size: 1024Bytes

4.18 SR_API_CreateWorldData

The function converts imagedata to worldcoordinates.
Make sure that a LUT is loaded. The LUT contains information about how to convert the data.

Prototype:	
int SR_API_CreateWorldData (CAMDESC *cd, int startX, int width, unsigned short *profil, SR_3DPOINT *world);	
Arguments:	
cd	Camera object.
startX	StartX value from ROI setting. Value is vailable in the arguments from the callback function Attention! Some sensors may use StartY instead of StartX (e.g. SR1200)
width	Width or number of points in a profile.
profil	Pointer to the image based profile data.
world	Pointer to the array where worldcoordinates will be stored. World points containing Values lower or same as "NONWORLDMARK" should be removed ! Markers for invalid points: #define NONWORLDMARK -999990.0 #define NONWORLDVALUE -999997.0 #define NONLUTVALUE -999998.0 #define BADWORLDVALUE -999999.0

4.19 SR_API_RemapWorldToMatrix

Remap world coordinate matrix into a 2D x/y frame with constant X spacing.
Y coordinate is the profile count. You need the remapping function to convert 3D World profiles into a 2D frame which you can compute with standard vision tools.
The 2D frame x coordinate is the profile Y coordinate with constant world spacing (mm) between the pixels. The „grey value“ of a pixel (16bit) will now be the profile distance (world Z) information. One „grey value“ will, depending on the scaleZ & offsetZ value, represent e.g.1µm. The 2D frame y coordinate will represent the distance of 2 profiles while the objecte to scan is moving.
Constant scan rate = 200Hz = 200scan/sec, object speed = 10mm/sec

-->> Scan Distance = 10mm/sec / 200scan/sec = 0.05mm/scan
So each pixel step in 2D frame y direction represents a distance of 0.5mm
For measurement purpose we have to set the 2D frame x step representation as well to 0.05mm!
Calculation abs (rangeEnd – rangeStart) / len = 0.05 [mm]
Example: len = 1280 (CMOS width) → 0.05 * 1280 = (_end – _start) = 64mm
-->> rangeStart = -32mm ; rangeEnd = 32mm

Prototype:

```
int SR_API_RemapWorldToMatrix (SR_3DPOINT *srW,
unsigned short *intens, int len,
float rangeStart, float rangeEnd,
unsigned short *iS, unsigned short *zS, float offsetZ, float scaleZ);
```

Arguments:

srW	Input World data, one profile
intens	Input Intensity data
len	Length / width of profile
rangeStart	Remaper Y world start coordinate
rangeEnd	Remaper Y world end coordinate
iS	Result array for intensity profile (16bit)
zS	Result array for World Z coordinate (16bit)
offsetZ	Value added to the world Z value before it is scaled (divided by scale Z; use 40 for offsetZ)
scaleZ	Divider to convert the world z (float) value into a 16bit unsigned short value (Use 0.001 for 1µm resolution)

4.20 SR_API_WorldtoImage

The function converts world data to imager coordinates

Prototype:	
int SR_API_CreateWorldData (CAMDESC *cd, SR_3DPOINT *world, int* imgX, int* imgY);	
Arguments:	
cd	Camera object.
world	World coordinate system in 3D according to the structure SR_3DPOINT
imgX	Pointer to the coordinates of X axis in image data
imgY	Pointer to the coordinates of Y axis in image data

5 Administration

5.1 SR_API_Update_Cam

This function updates the camera's firmware. The new firmware gets active after switching the camera off and on again. Use '*.fup' files.

Prototype:	
`int SR_API_Update_Cam (CAMDESC* cam, char* filename);`	
Arguments:	
cam	Camera object.
fileName	Path & file name of type '*.fup'.

5.2 SR_API_ChangeIp

IP address and port number of the camera can be changed.
We recommend to use SR_Studio for this.
The camera will use the new IP address after a power off/on.

Prototype:

```
int SR_API_ChangeIp (CAMDESC* cam,
          unsigned char* newIPadr,
          unsigned short newPort);
```

Arguments:	
cam	Camera object.
newIPadr	IP4 Address within 4 bytes
newPort	port number

5.3 SR_API_GetMACAdr

The MAC address of the camera can be read.

Prototype:	
int SR_API_GetMACAdr(CAMDESC* cam, unsigned char* mac);	
Arguments:	
cam	Camera object.
mac	MAC Address within 6bytes

5.4 SR_API_GetCamVersion

Use this function to read the hardware & software version of the camera.

Prototype:	
int SR_API_GetCamVersions(CAMDESC* cam, char* versionSW, char* versionHW);	
Arguments:	
cam	Camera object.
versionSW	Software Version String of Camera
versionHW	Hardware Version String of Camera

5.5 SR_API_GetSensorType

The part number of the sensor can be read.

Prototype:	
int SR_API_GetSensorType(CAMDESC* cam, char* partNumber);	
Arguments:	
cam	Camera object.
partNumber	Partnumber of Sensor eg. SR700: 3.000.026 SR1400: 3.000.015 SR1200: 3.000.014

5.6 SR_API_GetSerial

The serial number of the sensor can be read.

Prototype:	
int SR_API_GetSerial(CAMDESC* cam, unsigned char* serial);	
Arguments:	
cam	Camera object.
serial	Serial number of Sensor eg. 00 00 08 03 99

5.7 SR_API_GetSensorName

This function can be used to read the sensor name.

Prototype: int SR_API_GetSensorName(CAMDESC* cam, char* productName);	
Arguments:	
cam	Camera object.
productName	Reads the name of the sensor with the objective information eg. SR710

5.8 SR_API_GetSensorSize

This function can be used to read imager width and height.

Prototype: int SR_API_GetSensorSize(CAMDESC* cam, int* width, int*height);	
Arguments:	
cam	Camera object.
width	Returns the imager width
height	Returns the imager height

5.9 SR_API_GetSensorCenterPosition

This function can be used to read imager center position.

Prototype:	
`int SR_API_GetSensorCenterPosition(CAMDESC* cam, int* centerX, int* centerY);`	
Arguments:	
cam	Camera object.
centerX	Returns X coordinate of the center position of the imager
centerY	Returns Y coordinate of the center position of the imager

6 Camera Parameter

Camera parameters are used to program the camera. The parameters are stored in a parameter file and are sent to the camera. With this functions the parameters in the camera are changed without the need to setup a new parameter file.
To use this functions you have to:

1. Read a parameter file from disk
2. Send the parameters to the camera.

6.1 SR_API_SetImager

Set CMOS Module (imager) ROI values without sending a new parameterset. The sensor has to be stopped in order to set the ROI.

Prototype:

```
int SR_API_SetImager (CAMDESC *cd,  int inst,
int startX, int startY, int width, int height,
int gainEnable, int gain);
```

Arguments:

cd	Camera object.
inst	Module Instance use 0 (zero)
startX	ROI start point X position
startY width	ROI start point Y position ROI width
height	ROI height
gainEnable gain	Imager gain stage enable Gain table entries (0..15 but use 0..4 for SR14xx)

6.2 SR_API_GetImager

Get CMOS Module (imager) ROI values from loaded parameter set. The scanner has to be stopped and started again after this function is used for correct results.

Prototype:	
int SR_API_GetImager (CAMDESC *cd, int inst, int *startX, int *startX, int *width, int *height, int *gainEnable, int *gain);	
Arguments:	
cd	Camera object.
inst	Module Instance use 0 (zero)
startX	ROI start point X position
startY	ROI start point Y position
width	ROI width
height	ROI height
gainEnable	Imager gain stage enable
gain	Gain table entries

6.3 SR_API_SetExposure

Set Exposure Module values without sending a new parameter set.

Prototype: int SR_API_SetExposure (CAMDESC *cd, int inst, int enableDoubleExpo, int expoFirst, int expoSecond);	
Arguments:	
cd	Camera object.
inst	Module Instance use 0 (zero)
enableDoubleExpo	1: Enable double exposure mode
expoFirst	Value for 1st exposure time [µsec](low value)
expoSecond	Value for 2nd exposure time [µsec] (higher or same than 1st value)

6.4 SR_API_GetExposure

Get Exposure Module values from a loaded parameter set.

Prototype: int SR_API_GetExposure (CAMDESC *cd, int inst, int *enableDoubleExpo, *enableCtrlSmall, int *enableCtrlLarge, int *expo_small, int *expo_large);	
Arguments:	
cd	Camera object.
inst	Module Instance use 0 (zero)
enableDoubleExpo	1: Enable double exposure mode
expoFirst	Value for 1st exposure time [µsec](low value)
expoSecond	Value for 2nd exposure time [µsec] (higher or same than 1st value)

6.5 SR_API_SetTrigger

Set Trigger Module values without sending a new parameter set.

Prototype:	
int SR_API_SetTrigger (CAMDESC *cd, int inst, int mode, int source, int edge, int outputSelect, int trigFrq, int trigCnt, int trigOffset);	
Arguments:	
cd	Camera object.
inst	Module Instance use 0 (zero)
mode	0: free running 1: intern trigger frequency generator 2: extern trigger 3: extern trigger but enabled with IN0 input 4: free running but enabled with IN0 input
source	If an external trigger mode is seleted 0: trigger from input IN1 1: trigger from input IN2 2: trigger from input IN1 and IN1 signals from IN1 is logical ORED with IN2 3: internal Trigger (TTL) you cannot use !! 4: Quadrature Encoder (use IN1 for quad A / IN2 for quad B inputs)
edge	0: trigger with rising edge 1: trigger with falling edge 2: trigger with both edges
outputSelect	The internal trigger generator may output its freq. On 24Volt outputs 0: No outputs used 1: OUT0 used 2: OUT1 used 3: both outputs OUT0 / OUT1 used
trigFrq	Frequency of the internal freq. Generator use Values form 3 to 65000 Hz
triCnt	Extern Trigger divider 0: NO division 1: division by 2

	2: division by 3 ..... 100: division by 101 ...
trigOffset	Trigger offset. After a Camera Start Signal this number of triggers are not passed to the Camera, they are suppressed.

6.6 SR_API_GetTrigger

Get Trigger Module values from actual parameter set.

Prototype:

int SR_API_GetTrigger (CAMDESC *cd, int inst,
int *mode, int *source, int *edge, int *outputSelect,
int *trigFrq, int *trigCnt, int *trigOffset);

Arguments:

cd	Camera object.
inst	Module Instance use 0 (zero)
mode	0: free running 1: intern trigger frequency generator 2: extern trigger 3: extern trigger but enabled with IN0 input 4: free running but enabled with IN0 input
source	If an external trigger mode is seleted 0: trigger from input IN1 1: trigger from input IN2 2: trigger from input IN1 and IN1 signals from IN1 is logical ORED with IN2 3: internal Trigger (TTL) you can not use !! 4: Quadrature Encoder (use IN1 for quad A / IN2 for quad B inputs)
edge	0: trigger with rising edge 1: trigger with falling edge 2: trigger with both edges
outputSelect	The internal trigger generator may output its freq. On 24Volt outputs 0: No outputs used 1: OUT0 used 2: OUT1 used

	3: both outputs OUT0 / OUT1 used
trigFrq	Frequency of the internal freq. Generator use Values form 3 to 65000 Hz
trigCnt	Extern Trigger divider 0: NO division 1: division by 2 2: division by 3 ..... 100: division by 101 ...
trigOffset	Trigger offset. After a Camera Start Signal this number of triggers are not passed to the Camera, they are suppressed.

6.7 SR_API_SetReadInput

Set Read_Input Module values without sending a new parameterset.

Prototype: int SR_API_SetReadInput (CAMDESC* cd, int inst, int *active, int *condition);	
Arguments:	
cd	Camera object.
inst	Module Instance use 0 to 4
activ	Array of width 4 (active[4]) Each Index corresponds to a sensor digital input0/1/2/3 0: disables an Input[0..3] action 1: enables an Input[0..3] action
condition	Array of width 4 (active[4]) Each Index corresponds to a sensor digital input0/1/2/3 0: Input[0..3] tested for *negative* edge 1: Input[0..3] tested for *positive* edge

6.8 SR_API_GetReadInput

Get Read_Input Module values from loaded parameter set.

Prototype:	
int SR_API_GetReadInput (CAMDESC* cd, int inst, int *active, int *condition);	
Arguments:	
cd	Camera object.
inst	Module Instance use 0 to 4
activ	Array of width 4 (active[4]) Each Index corresponds to a sensor digital input0/1/2/3 0: disabled Input[0..3] action 1: enabled Input[0..3] action
condition	Array of width 4 (active[4]) Each Index corresponds to a sensor digital input0/1/2/3 0: Input[0..3] tested for *negative* edge 1: Input[0..3] tested for *positive* edge

6.9 SR_API_SetModule

Set the Module values without sending the parameterset.

Prototype:	
int SR_API_SetModule (CAMDESC* cd, int rowNum, int *nextName);	
Arguments:	
cd	Camera object.
rowNum	Row Number in the parameterset
nextName	String with the name of the next module

6.10 SR_API_GetModule

Get the Module values from the parameterset.

Prototype:

```
int SR_API_GetModule (CAMDESC* cd, int rowNum, char *name, char *methodName,
                int *methodCode, chat *nextName, int *inst, int *numModule);
```

Arguments:

cd	Camera object.
rowNum	Row Number in the parameterset
name	String with the module name
methodName	String with the name of method
methodCode	Pointer to Code number of the method
nextName	String with the name of the next module
inst	Pointer to the Module instance
numModule	Pointer to the module number

6.11 SR_API_SetParamProfile

Set the parameters for capturing profile.

Prototype:

```
int SR_API_SetParamProfile (CAMDESC* cd, int inst, int packetSize, int numProfiles);
```

Arguments:

cd	Camera object.
inst	Module Instance use 0 to 4
packetSize	Size of a profile packet to capture
numProfiles	Total number of profiles to be captured Example : If packetSize = 10 and numProfiles = 100, the sensor captures 100 profiles in packets of 10 each

6.12 SR_API_GetParamProfile

Get the parameters for capturing profile.

Prototype:

```
int SR_API_GetParamProfile (CAMDESC* cd, int inst, int* packetSize, int* numProfiles);
```

Arguments:

cd	Camera object.
inst	Module Instance use 0 to 4
packetSize	Pointer to profile packet size
numProfiles	Pointer to total number of profiles captured

7 Return value and Error codes

Each function provided by the API returns an integer value. This return value can be used to identify if the function call is success or if it is associated with an error. Different errors associated with the function call are identified by unique error codes. The error codes are defined in the file sr_api_errorcodes.h.

8 Revision History

File Version	Date	Remakes
1.1.0	21-02-13	Base Document
1.2.0	23-04-13	Added section 7 error codes and SR_API_GetSerial function
1.3.0	24-04-13	Added sections 6.4 to 6.8, changed sections 4.13.1 /2/3 SR_API_GetAPIVersion return & argument values changed SR_API_SendParsToCam return value changed SR_API_GetMACAdr return value changed SR_API_GetSerial return value changed
1.4.0	30-04-13	Added sections 6.9 and 6.10
1.5.0	07-05-13	Added section 5.7
1.6.0	18-07-13	Added Sections 6.11, 6.12, 4.15, 5.8 and 5.9
1.7.0	26-07-13	Added Section 4.20