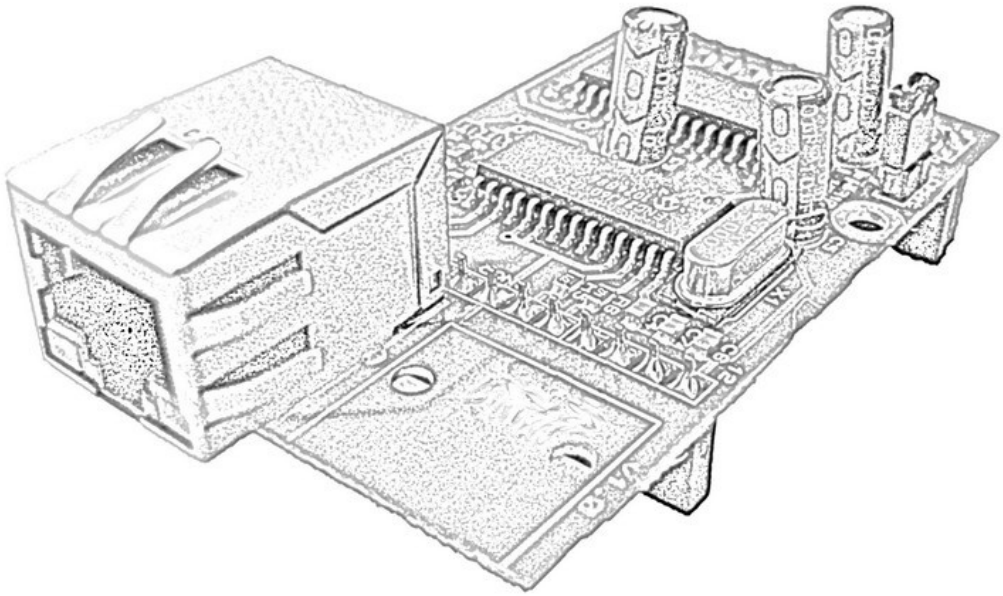
sch-remote.com

SR01E12
Library Functions



Ethernet to UART / SPI / I2C / IO controller

List of Functions

Connection negotiation

SR_IPLIST Linked list structure of discovered devices used by sr_discover

```
typedef struct sr_iplist
  {
  unsigned long ip;
  unsigned char mac[6];
  struct sr_iplist *next;
  } SR_IPLIST
```

ip IP address of discovered device. Big endian format, for example 192.168.1.2 is 0x0201a8c0
mac MAC address of discovered device
next Pointer to next SR_IPLIST element. Last in the list is NULL

sr_discover Search for connected devices

```
SR_FUNC SR_IPLIST* sr_discover(const char *ip, unsigned short port = 3101)
```

Parameters:
ip Zero terminated string to broadcast IP address of the queried network
port Port number of the target devices. Default configuration uses port 3101

Return:
NULL on failure. Pointer to SR_IPLIST structure

Remarks:
When returned list is no more used, sr_discover_free() must be called to free it.

sr_discover_free Free the resources used by sr_discover() function

```
SR_FUNC void sr_discover_free(SR_IPLIST *list)
```

Parameters:
list Pointer to SR_IPLIST structure

Return: none

sr_open_eth Open a connection trough a ethernet interface

```
SR_FUNC SR_HANDLE sr_open_eth(const char *ip, unsigned short port = 3101)
```

Parameters:
ip Zero terminated string to IP address or hostname of the target device
port Port number of the target device. Default configuration uses port 3101

Return:
NULL on failure. Handle on success

Remarks:
When the connection is no more used, sr_close() must be called.

sr_close Close a active connection

```
SR_FUNC void sr_close(SR_HANDLE srh)
```

Parameters:
srh Handle to connection

Return: none

sr_nop Send a NO OPERATION command to device

```
SR_FUNC int sr_nop(SR_HANDLE srh)
```

Parameters:
srh Handle to connection

Return:
0 on failure. Non zero on success.

sr_error_info Return a message describing occurred error

```
SR_FUNC const char* sr_error_info(SR_HANDLE srh)
```

Parameters:
srh Handle to connection

Return:
Zero terminated string

sr_version Return library version

```
SR_FUNC unsigned short sr_version()
```

Parameters: none

Return:
version word

Remarks:
Major version is in high order byte and minor version is in low order byte. Example: 0x0102 means version 1.2.

Pins IO and type configuration

sr_pin_type Possible pins configuration types:

```
enum sr_pin_type {sr_pt_analog_in=0, sr_pt_din, sr_pt_din_pullup, sr_pt_dout_low,
sr_pt_dout_high, sr_pt_dout_opendrain_open, sr_pt_dout_opendrain_short}
```

sr_pt_analog_in analog input

sr_pt_din	digital input
sr_pt_din_pullup	digital input with a integrated pullup resistor
sr_pt_dout_low	digital output, low level at startup
sr_pt_dout_high	digital output, high level at startup
sr_pt_dout_opendrain_open	open drain output, hi impedance on startup
sr_pt_dout_opendrain_short	open drain output, conducting at startup

sr_pin_setup Configure a pin type

```
SR_FUNC int sr_pin_setup(SR_HANDLE srh, int pin, sr_pin_type pin_type)
```

Parameters:

srh	Handle to connection
pin	Pin number 0..11 to configure
pin_type	Type to assign

Return:
0 on failure. Non zero on success

Remarks:
Only pins 4-9 can be analog input. Trying to set any of 1-3,10,11 pins to analog input, will be set as digital input. Setup of any individual pin, change all digital output pins to their default level.

sr_pin_status Get type of a pin

```
SR_FUNC int sr_pin_status(SR_HANDLE srh, int pin, sr_pin_type *pin_type)
```

Parameters:

srh	Handle to connection
pin	Pin number 0..11 to configure
pin_type	Pointer where to store the type

Return:
0 on failure. Non zero on success

sr_pins_setup Configure the type of all pins

```
SR_FUNC int sr_pins_setup(SR_HANDLE srh, sr_pin_type pin_type[12])
```

Parameters:

srh	Handle to connection
pin_type	Array of pin types to assign

Return:
0 on failure. Non zero on success

Remarks:
Only pins 4-9 can be analog input. Trying to set any of 1-3,10,11 pins to analog input, will be set as digital input.

sr_pins_status Get type of all pins

```
SR_FUNC int sr_pins_status(SR_HANDLE srh, sr_pin_type pin_type[12])
```

Parameters:
srh Handle to connection
pin_type Pointer where to store the types

Return:
0 on failure. Non zero on success

GPIO and Analog Interface

sr_pin_set Set the state of digital out / open drain pin

```
SR_FUNC int sr_pin_set(SR_HANDLE srh, int pin, bool b)
```

Parameters:
srh Handle to connection
pin Pin number 0..11 to configure
b 0/1 state

Return:
0 on failure. Non zero on success

Remarks:
On pins setup as analog or digital input or used for communication this function have not effect.

sr_pin_get Get the state of a pin

```
SR_FUNC int sr_pin_get(SR_HANDLE srh, int pin, bool *b)
```

Parameters:
srh Handle to connection
pin Pin number 0..11 to configure
b Pointer where to store the state

Return:
0 on failure. Non zero on success

Remarks:
On pins setup as analog input result is undetermined.

sr_pins_set Set the state of digital out / open drain pin

```
SR_FUNC int sr_pins_set(SR_HANDLE srh, unsigned short b)
```

Parameters:
srh Handle to connection
b Each bit 0..11 correspond to 0/1 state of related pin

Return:

0 on failure. Non zero on success

Remarks:
On pins setup as analog or digital input or used for communication this have not effect.

sr_pins_get Get the state of a pin

```
SR_FUNC int sr_pins_get(SR_HANDLE srh, unsigned short *b)
```

Parameters:
srh Handle to connection
b Pointer where to store the state. Each bit 0..11 correspond to 0/1 state of related pin.

Return:
0 on failure. Non zero on success

Remarks:
On pins setup as analog input result is undetermined.

sr_pin_get_analog Get the value of analog pin

```
SR_FUNC int sr_pin_get_analog(SR_HANDLE srh, int pin, unsigned short *val)
```

Parameters:
srh Handle to connection
pin Pin number 4..9 to read
val Pointer where to store the value. Range 0..1024.

Return:
0 on failure. Non zero on success

Remarks:
On only digital pins function return error. On pins not setup as analog input result is undetermined.

Serial Interface

UART_MODE_XXX Possible modes of UART communication

```
#define UART_MODE_PARITY_NO   0x0000
#define UART_MODE_PARITY_EVEN 0x0002
#define UART_MODE_PARITY_ODD  0x0004
#define UART_MODE_STOP_ONE    0x0000
#define UART_MODE_STOP_TWO    0x0001
```

UART_MODE_PARITY_NO data frame is with 8 data bits and no parity check
UART_MODE_PARITY_EVEN data frame is with 8 data bits and 1 even parity check bit
UART_MODE_PARITY_ODD data frame is with 8 data bits and 1 odd parity check bit
UART_MODE_STOP_ONE data frame is with 1 stop bit
UART_MODE_STOP_TWO data frame is with 2 stop bits

sr_uart_enable Setup the UART interface

```
SR_FUNC int sr_uart_enable(SR_HANDLE srh, int port, int mode, int baud, int rx_pin,
int tx_pin)
```

Parameters:
srh Handle to connection
port Uart module to query (0 or 1)
mode Combination of UART_MODE_XXX flags
baud UART baud rate
rx_pin Pin number used for data receive
tx_pin Pin number used for data transmit

Return:
0 on failure. Non zero on success

Remarks:
Function will accept any baudrade, but will setup the device to the range of 16 .. 4 000 000. Actual baud rate is: *actual* = 1 000 000 / (X+1), where X in 0 .. 65535 for a baudrates up to 1 000 000 and *actual* = 4 000 000 / (X+1), where X in 0 .. 3 for above.
As *rx_pin* and *tx_pin* can be selected from pins 0..2, 4..10 (pins 3 and 11 cannot be used). Function will return error if try to use unallowed pin. Selected pins should be previously set as digital input, digital output, open drain output. Trying to use analog pin will not change the behavior of the pin. Trying to use same pins for several functions have the effect of: last change override the previous purpose of the pin.

sr_uart_status Read the setting of UART interface

```
SR_FUNC int sr_uart_status(SR_HANDLE srh, int port, int *mode, int *baud, int
*rx_pin, int *tx_pin)
```

Parameters:
srh Handle to connection
port Uart module to query (0 or 1)
mode Pointer where to store combination of UART_MODE_XXX flags
baud Pointer where to store UART baud rate
rx_pin Pointer where to store receive data pin number
tx_pin Pointer where to store transmit pin number

Return:
0 on failure. Non zero on success

Remarks:
If UART functionality is not enabled, returned data is: mode = 0, baud = 0, rx_pin = -1, tx_pin = -1

sr_uart_disable Disable UART interface

```
SR_FUNC int sr_uart_disable(SR_HANDLE srh, int port)
```

Parameters:
srh Handle to connection
port Uart module to query (0 or 1)

Return:

0 on failure. Non zero on success

Remarks:
Assigned pins are turned to their state set by sr_pin_setup or sr_pins_setup.

sr_uart_write Write a byte to the UART interface

```
SR_FUNC int sr_uart_write(SR_HANDLE srh, int port, unsigned char data)
```

Parameters:
srh Handle to connection
port Uart module to query (0 or 1)
data A byte to be written

Return:
0 on failure. Non zero on success

sr_uart_read Read a byte from the UART interface

```
SR_FUNC int sr_uart_read(SR_HANDLE srh, int port, unsigned char *data, unsigned short
*readed = NULL)
```

Parameters:
srh Handle to connection
port Uart module to query (0 or 1)
data A byte pointer where to store readed data
readed Pointer where to store actual number of readed bytes

Return:
0 on failure. Non zero on success

Remarks:
If *readed* is NULL, function will wait for the data byte, else if not pending data *readed* will be 0

sr_uart_write_arr Write a array of bytes to the UART interface

```
SR_FUNC int sr_uart_write_arr(SR_HANDLE srh, int port, unsigned char *arr, unsigned
short cnt)
```

Parameters:
srh Handle to connection
port Uart module to query (0 or 1)
arr A byte pointer to the data for write
cnt Number if bytes to write

Return:
0 on failure. Non zero on success

sr_uart_read_arr Read a array of bytes from the UART interface

```
SR_FUNC int sr_uart_read_arr(SR_HANDLE srh, int port, unsigned char *arr, unsigned
short cnt, unsigned short *readed = NULL)
```

Parameters:
srh Handle to connection
port Uart module to query (0 or 1)
arr A byte pointer where to store readed array
cnt Number if bytes to read

Return:
0 on failure. Non zero on success

Remarks:
If *readed* is NULL, function will wait for cnt data bytes to be available, else will return up to cnt bytes that are available, *readed* contain the actual number.

SPI Interface

SPI_MODE_XXX Possible modes of SPI communication

```
#define SPI_MODE_IDLE_LOW           0x0000
#define SPI_MODE_IDLE_HIGH          0x0040
#define SPI_MODE_CLK_NODELAY        0x0000
#define SPI_MODE_CLK_HALFDELAY      0x0100
#define SPI_MODE_IN_MIDDLE          0x0000
#define SPI_MODE_IN_END             0x0200
```

SPI_MODE_IDLE_LOW clock signal is low level at idle state and high level in active state
SPI_MODE_IDLE_HIGH clock signal is hogh level at idle state and low level in active state
SPI_MODE_CLK_NODELAY clock change to active state at the beginning of bit window*
SPI_MODE_CLK_HALFDELAY clock change to active state at the middle of bit window*
SPI_MODE_IN_MIDDLE input data in sampled at the middle of bit window*
SPI_MODE_IN_END input data in sampled at the end of bit window*
*bit window - output data is changed at the beginning of bit window and stay unchanged to the beginning of following one. Refer to Datasheet for more details.

sr_spi_enable Setup the SPI interface

```
SR_FUNC int sr_spi_enable(SR_HANDLE srh, unsigned short mode, int baud, int clk_pin,
int out_pin, int in_pin)
```

Parameters:
srh Handle to connection
mode Combination of SPI_MODE_XXX flags
baud SPI clock baud rate
clk_pin Pin number used for clock out
out_pin Pin number used for data out
in_pin Pin number used for data in

Return:
0 on failure. Non zero on success

Remarks:
Function will accept any baudrade, but will setup the device to the nearest of the following baudrates: 8

000 000, 5 333 333, 4 000 000, 3 200 000, 2 666 667, 2 285 714, 2 000 000, 1 333 333, 1 000 000, 800 000, 666 667, 571 429, 500 000, 333 333, 250 000, 200 000, 166 667, 142 857, 125 000, 83 333, 62 500, 50 000, 41 667, 35 714, 31 250.
As *clk_pin*, *out_pin*, *in_pin* can be selected from pins 0..2, 4..10 (pins 3 and 11 cannot be used). Function will return error if try to use unallowed pin. Selected pins should be previously set as digital input, digital output, open drain output. Trying to use analog pin will not change the behavior of the pin. Trying to use same pins for several functions have the effect of: last change override the previous purpose of the pin.

sr_spi_status Read the setting of SPI interface

```
SR_FUNC int sr_spi_status(SR_HANDLE srh, unsigned short *mode, int *baud, int
*clk_pin, int *out_pin, int *in_pin)
```

Parameters:

srh	Handle to connection
mode	Pointer where to store combination of SPI_MODE_XXX flags
baud	Pointer where to store SPI clock baud rate
clk_pin	Pointer where to store clock out pin number
out_pin	Pointer where to store data out pin number
in_pin	Pointer where to store data in pin number

Return:
0 on failure. Non zero on success

Remarks:
If SPI functionality is not enabled, returned data is: mode = 0, baud = 0, clk_pin = -1, out_pin = -1, in_pin = -1

sr_spi_disable Disable SPI interface

```
SR_FUNC int sr_spi_disable(SR_HANDLE srh)
```

Parameters:

srh	Handle to connection

Return:
0 on failure. Non zero on success

Remarks:
Assigned pins are turned to their state set by sr_pin_setup or sr_pins_setup.

sr_spi_write Write a byte to the SPI interface

```
SR_FUNC int sr_spi_write(SR_HANDLE srh, unsigned char data)
```

Parameters:

srh	Handle to connection
data	A byte to be written

Return:
0 on failure. Non zero on success

Remarks:
Typical SPI communication require a CS (chip select) signal. For that purpose can be used digital out pin and sr_pin_set, before and after this function is invoked.

sr_spi_read Read a byte from the SPI interface

```
SR_FUNC int sr_spi_read(SR_HANDLE srh, unsigned char *data)
```

Parameters:
srh Handle to connection
data A byte pointer where to store readed data

Return:
0 on failure. Non zero on success

Remarks:
Typical SPI communication require a CS (chip select) signal. For that purpose can be used digital out pin and sr_pin_set, before and after this function is invoked.

sr_spi_write_arr Write a array of bytes to the SPI interface

```
SR_FUNC int sr_spi_write_arr(SR_HANDLE srh, unsigned char *arr, unsigned short cnt)
```

Parameters:
srh Handle to connection
arr A byte pointer to the data for write
cnt Number if bytes to write

Return:
0 on failure. Non zero on success

Remarks:
Typical SPI communication require a CS (chip select) signal. For that purpose can be used digital out pin and sr_pin_set, before and after this function is invoked.

int sr_spi_read_arr Read a array of bytes from the SPI interface

```
SR_FUNC int sr_spi_read_arr(SR_HANDLE srh, unsigned char *arr, unsigned short cnt)
```

Parameters:
srh Handle to connection
arr A byte pointer where to store readed array
cnt Number if bytes to read

Return:
0 on failure. Non zero on success

Remarks:
Typical SPI communication require a CS (chip select) signal. For that purpose can be used digital out pin and sr_pin_set, before and after this function is invoked.

I2C Interface

sr_i2c_enable Setup the I2C interface

```
SR_FUNC int sr_i2c_enable(SR_HANDLE srh, int port, int baud)
```

Parameters:
srh Handle to connection
port I2C module to setup: 0 - pin 0:SDA, pin 1:SCL or 1 - pin 8:SDA, pin 9:SCL
baud I2C clock baud rate

Return:
0 on failure. Non zero on success

Remarks:
Function will accept any baudrade, but will setup the device to the range of 62 112 .. 2 857 143. Actual baud rate is: *actual* = 16 000 000 / (X+2.6), where X in 3 .. 255. Baud rate slightly depend from the I2C pullup resistors value.

sr_i2c_status Read the setting of I2C interface

```
SR_FUNC int sr_i2c_status(SR_HANDLE srh, int port, int *baud)
```

Parameters:
srh Handle to connection
port I2C module to query
baud Pointer where to store I2C clock baud rate

Return:
0 on failure. Non zero on success

Remarks:
If I2C functionality is not enabled, returned data is: baud = 0

sr_i2c_disable Disable I2C interface

```
SR_FUNC int sr_i2c_disable(SR_HANDLE srh, int port)
```

Parameters:
srh Handle to connection
port I2C module to query

Return:
0 on failure. Non zero on success

Remarks:
Used pins are turned to their state set by sr_pin_setup or sr_pins_setup.

sr_i2c_write Write a byte to the I2C interface

```
SR_FUNC int sr_i2c_write(SR_HANDLE srh, int port, unsigned char i2c_addr, unsigned
char data)
```

Parameters:

srh	Handle to connection
port	I2C module to query
i2c_addr	Address of target device
data	A byte to be written

Return:
0 on failure. Non zero on success

Remarks:
I2C address is shifted 1 bit left: A6(MSB) A5 A4 A3 A2 A1 A0 <don't care bit>(LSB)

sr_i2c_read Read a byte from the I2C interface

```
SR_FUNC int sr_i2c_read(SR_HANDLE srh, int port, unsigned char i2c_addr, unsigned
char *data)
```

Parameters:

srh	Handle to connection
port	I2C module to query
i2c_addr	Address of target device
data	A byte pointer where to store readed data

Return:
0 on failure. Non zero on success

Remarks:
I2C address is shifted 1 bit left: A6(MSB) A5 A4 A3 A2 A1 A0 <don't care bit>(LSB)

sr_i2c_write_read Write a byte to the I2C interface, restart and read a byte from same I2C address

```
SR_FUNC int sr_i2c_write_read(SR_HANDLE srh, int port, unsigned char i2c_addr,
unsigned char data_write, unsigned char *data_read)
```

Parameters:

srh	Handle to connection
port	I2C module to query
i2c_addr	Address of target device
data_write	A byte to be written
data_read	A byte pointer where to store readed data

Return:
0 on failure. Non zero on success

Remarks:
I2C address is shifted 1 bit left: A6(MSB) A5 A4 A3 A2 A1 A0 <don't care bit>(LSB)

sr_i2c_write_arr Write a array of bytes to the I2C interface

```
SR_FUNC int sr_i2c_write_arr(SR_HANDLE srh, int port, unsigned char i2c_addr,
unsigned char *arr, unsigned short cnt)
```

Parameters:

srh	Handle to connection
port	I2C module to query
i2c_addr	Address of target device
arr	A byte pointer to the data for write
cnt	Number if bytes to write

Return:
0 on failure. Non zero on success

Remarks:
I2C address is shifted 1 bit left: A6(MSB) A5 A4 A3 A2 A1 A0 <don't care bit>(LSB)
Maximum size of arr is 128 bytes. If bigger, function return a error.

sr_i2c_read_arr Read a array of bytes from the I2C interface

```
SR_FUNC int sr_i2c_read_arr(SR_HANDLE srh, int port, unsigned char i2c_addr, unsigned
char *arr, unsigned short cnt)
```

Parameters:

srh	Handle to connection
port	I2C module to query
i2c_addr	Address of target device
arr	A byte pointer where to store readed array
cnt	Number if bytes to read

Return:
0 on failure. Non zero on success

Remarks:
I2C address is shifted 1 bit left: A6(MSB) A5 A4 A3 A2 A1 A0 <don't care bit>(LSB)
Maximum size of arr is 128 bytes. If bigger, function return a error.

sr_i2c_write_read_arr Write a array of bytes to the I2C interface, restart and read a array of bytes from same I2C address

```
SR_FUNC int sr_i2c_write_read_arr(SR_HANDLE srh, int port, unsigned char i2c_addr,
unsigned char *arr_write, unsigned short cnt_write, unsigned char *arr_read, unsigned
short cnt_read)
```

Parameters:

srh	Handle to connection
port	I2C module to query
i2c_addr	Address of target device
arr_write	A byte pointer to the data for write
cnt_write	Number if bytes to write
arr_read	A byte pointer where to store readed array
cnt_read	Number if bytes to read

Return:
0 on failure. Non zero on success

Remarks:
I2C address is shifted 1 bit left: A6(MSB) A5 A4 A3 A2 A1 A0 <don't care bit>(LSB)
Maximum size of arr_write is 128 bytes. If bigger, function return a error.
Maximum size of arr_read is 128 bytes. If bigger, function return a error.

Counter Interface

sr_cnt_enable Setup a counter module

```
SR_FUNC int sr_cnt_enable(SR_HANDLE srh, int port, int in_pin)
```

Parameters:
srh Handle to connection
port Counter module to query (range from 0 to 3)
in_pin Pin number used for clock signals

Return:
0 on failure. Non zero on success

Remarks:
As in_pin can be selected from pins 0..2, 4..10 (pins 3 and 11 cannot be used). Function will return error if try to use unallowed pin. Selected pin should be previously set as digital input. Trying to use analog pin will not change the behavior of the pin.

sr_cnt_status Read the setting of counter module

```
SR_FUNC int sr_cnt_status(SR_HANDLE srh, int port, int *in_pin)
```

Parameters:
srh Handle to connection
port Counter module to query (range from 0 to 3)
in_pin Pointer where to store in pin number

Return:
0 on failure. Non zero on success

Remarks:
If counter module is not enabled, returned data is: in_pin = -1

sr_cnt_disable Disable counter module

```
SR_FUNC int sr_cnt_disable(SR_HANDLE srh, int port)
```

Parameters:
srh Handle to connection
port Counter module to query (range from 0 to 3)

Return:

0 on failure. Non zero on success

sr_cnt_reset Reset current count value to 0 of counter module

```
SR_FUNC int sr_cnt_reset(SR_HANDLE srh, int port)
```

Parameters:
srh Handle to connection
port Counter module to query (range from 0 to 3)

Return:
0 on failure. Non zero on success

sr_cnt_read Read current count value to 0 of counter module

```
SR_FUNC int sr_cnt_read(SR_HANDLE srh, int port, unsigned short *cnt)
```

Parameters:
srh Handle to connection
port Counter module to query (range from 0 to 3)
cnt Pointer where to store current count (0 to 65535)

Return:
0 on failure. Non zero on success

Ethernet and startup configuration

sr_store_config Store current setup as default (start up) configuration

```
SR_FUNC int sr_store_config(SR_HANDLE srh)
```

Parameters:
srh Handle to connection

Return:
0 on failure. Non zero on success

sr_store_ip Store the default (start up) network configuration

```
SR_FUNC int sr_store_ip(SR_HANDLE srh, unsigned long ip, unsigned long mask, unsigned
long gateway, bool dhcp)
```

Parameters:
srh Handle to connection
ip Ip address
mask Network mask
gateway Gataway
dchp Enable dhcp client (0 or 1)

Return:
0 on failure. Non zero on success

Remarks:
All addresses are in big endian format, for example 192.168.1.2 is 0x0201a8c0.
If dhcp client is enabled, device try to obtain the data from it and only if fail use the static address setup.

sr_read_ip Read default (start up) network configuration

```
SR_FUNC int sr_read_ip(SR_HANDLE srh, unsigned long *ip, unsigned long *mask,
unsigned long *gateway, bool *dhcp)
```

Parameters:

srh	Handle to connection
ip	Pointer where to store ip address
mask	Pointer where to store network mask
gateway	Pointer where to store gateway
dchp	Pointer where to store is dhcp client in use

Return:
0 on failure. Non zero on success