API for the NUDP based Ethernet camera.
(*CEthCamera* + *NudpSocket*)

This API is based on C++ Sockets Library 1.9.7 (Linux) and 2.0.4a (Windows) (www.alhem.net/Sockets).
Source codes with an example: www.ire.pw.edu.pl/~rsulej/CEthCamera (Linux) and www.ire.pw.edu.pl/~rsulej/CEthCamera/win (Windows).

Contact: rsulej@elka.pw.edu.pl

1. Main interface: **CEthCamera** class.

Manages whole communications with a device. Two local UDP ports are occupied (using *NudpSocket* objects) – one for sending NUDP commands and listening for the camera responses and RAW data transfers; camera watchdog refresh commands are sent automaticaly through the second UDP port. Main goal is not to loose any of the camera packets, even if they are unexpected in a given state of the *CEthCamera* object (strict implementation of NUDP protocol). Schema of a typical usage is as follows:

- Create a *CEthCamera* object – local network interface must be specified (by IP number); default local port number for commands and data transfer is #23100 (other may be specified); watchdog refresh is set to use the "local port" + 1 (def.: #23101). Camera state byte is 0 (`CAM_CONNECTION_OPEN=0, CAM_REFRESH_RUNNING=0, CAM_DATA_TAKING=0`). See also: Constructor description.
- Set up log files. See methods description.
- Initialize the connection with `Open()` method. If connection is estabilished (method returns `true`), flags `CAM_CONNECTION_OPEN` and `CAM_REFRESH_RUNNING` are set.
- Send commands to the camera – `SendCmd()`. This method blocks the current thread execution until the correct camera response is received or timeouts and retries are expired. If `SendCmd()` returns false – call `Close()` method and reopen the connection.
- When camera is ready for taking a picture – set up a memory buffer where image data will be stored (important: do it before each shot; for safety reasons – accidental overwriting previous image for example – this setting is cleared when transfer is completed) with `SetRawDataBuffer()` method and send `0x08h` command with `SendCmd()` method.
  If `0x08h` command is sent correctly, method returns `true` and `CAM_DATA_TAKING` flag is set. Data transfer is done in the background (it takes approximately 0.8s at 100Mb/s). Other commands may be sent to the camera while transfer is pending.
- Call `WaitForRawData()`. This will block the execution until data transfer is finished (or abnormally stopped). Method returns true if all image packets has been received, otherwise (false returned) lost packets should be retransmitted with `ReadMissing()`.
- Send other commands, take pictures and finally call `Close()` and destroy *CEthCamera* object.

Complete list of *CEthCamera* methods:

Constructor:

```
CEthCamera( const std::string &local_interface,
            port_t local_port = DEF_LOCAL_PORT )
```

`local_interface` - obligatory - IP address of LOCAL eth device as string: "192.168.123.126";

`local_port` - LOCAL UDP port number, if default is not suitable or there are many CEthCamera objects;

Remember: refresh communication is done using the UDP port number: **`local_port+1`**, so if there are many cameras in the system, all they must be assigned even port numbers (or all odd…).

Destructor:

```
~CEthCamera( void );
```

Destructor will wait until RAW data transfer is finished, but it will not wait for camera responses for commands.

Connection open:

```
bool Open( const std::string &remote_host,
           port_t remote_port = DEF_REMOTE_PORT );
```

`remote_host` - obligatory - IP address of the REMOTE device as string: "192.168.123.154";
`remote_port` - REMOTE UDP port number, if standard (#1234) is not for you;

Returns `true` and sets `CAM_CONNECTION_OPEN` and `CAM_REFRESH_RUNNING` flag if succeeded. That means 2 threads are running in the background from now: *refresh* and *listener*. Refresh is quite independent, it just sends `0xFC` command every 5s (by default); it will block `SendCmd()` method if connection is lost (no response for `0xFC` command) – check `CAM_REFRESH_RUNNING` flag in case of `SendCmd() == false`; refresh is suspended during RAW data transfers. Listener thread collects all packets from the camera; unexpected packets will be reported in log files.

Connection close:

```
bool Close( void );
```

Waits for RAW data, tries to close NUDP communication socket and if succeeded - shuts down listener and refresh threads and returns `true`.

Send command:

```
bool SendCmd(
          const char src_number[4],
          const char* src_data = NULL,
          const size_t src_data_length = 0,
          const char src_top = 0,
          char* dst_number = NULL,
          char* dst_data = NULL,
          size_t* dst_data_length = NULL,
          char* dst_top = NULL );
bool SendCmd(
          const unsigned int src_number,
          const char* src_data = NULL,
          const size_t src_data_length = 0,
          const char src_top = 0,
          unsigned int* dst_number = NULL,
          char* dst_data = NULL,
          size_t* dst_data_length = NULL,
          char* dst_top = NULL );
```

packet-to-send parameters:

`src_number`	- number field, obligatory (in char array: Lo byte first, Hi byte last);
`src_data`	- pointer to data field, `NULL` if there is no data to send;
`src_data_length`	- length of the data field;
`src_top`	- type of packet, 0 (command) by default;

pointers to store camera response:

`dst_number`	- address of 4-byte long buffer to store the returned NUDP number field;
`dst_data`	- address of the returned data, there must be enough space allocated (max. length is limited by NUDP specification: `NUDP_DATA_LENGTH`);
`dst_data_length`	- address of the data field length of the received packet;

Sends an NUDP command to the camera, gets the response. Method blocks current thread execution until camera response is received or error has occurred. If camera sends RAW data packets while we are waiting for command response - it's OK, they will be collected properly (and reported in logs).

Returns `true` if command is sent correctly and appropriate answer is obtained.
Returns `false` if:
- connection is closed (`CAM_CONNECTION_OPEN` flag not set)
- refresh thread is not running (`CAM_REFRESH_RUNNING` flag not set)
- command to send is `0x08` while `CAM_DATA_TAKING` flag is still set
- problem with mutex has occurred (that would be strange...)
- NUDP socket reported an error (`NUDP_CLOSED`, `NUDP_WAITING` states)

or:
- timeouts and damaged packets limit exceeded (connection will be closed in this case).

Access to the last camera response:

```
bool GetLastResponse(
          char* dst_number,
          char* dst_data = NULL,
          size_t* dst_data_length = NULL,
          char* dst_top = NULL );
bool GetLastResponse(
          unsigned int* dst_number,
          char* dst_data = NULL,
          size_t* dst_data_length = NULL,
          char* dst_top = NULL );
```

Get the last camera response packet. Do the parameter names look familiar? Yes, it is the "dst_*" part of `SendCmd()` and the meaning is the same. Methods access the last packet sent by camera as a command response (any RAW data transmissions do not interfere). It is your responsibility to know what was the last command sent to the camera.
Returns `true` if NUDP socket state is `NUDP_READY` and `NUDP_CMD_ERR` flag is cleared.

Set address for the next image from camera:

```
bool SetRawDataBuffer( char* buf_addr );
```

Sets pointer to an external buffer where RAW data should be stored. Must be called before each `0x08h` command, or data will not be recorded. Buffer length that you must allocate on your own:

`NUDP_DATA_LENGTH · NUDP_DATA_PACKETS`

Returns `false` if called before previous data transfer is finished (`CAM_DATA_TAKING` flag is set) – in this case method has no effect.

Wait for RAW data:

```
bool WaitForRawData( void );
```

Blocks execution until RAW data transmission is finished.
Returns: `true` if all packed collected; `false` otherwise (timeout occurred, some packets should be retransmitted).

Check image packets:

```
bool FragmentReceivedByOffset( int offset );
bool FragmentReceivedByIndex( int index );
```

Check if particular RAW packet is received. It is also safe to call these methods during data transfer.

Get the number of received packets:

```
int GetDataCount( void );
```

Get the number of RAW data packets already received, also safe during the transfer.

Retransmit lost packets:

```
bool ReadMissing( char* buf_addr );
```

Reads missing packets - uses `GetDataCount()` and `FragmentReceivedByIndex()` to get missing packets from last transmission (one by one, top=6).
Returns `true` if all packets has been received.
Returns `false` if some packets are still missing (command timeout or other error occurred).

Read/change defaults:

```
void SetRefreshTime( int sec );
```

Sets watchdog refresh interval in seconds (default 5s is used if not changed).

```
void SetDataTimeout( int msec );
```

Sets timeout BETWEEN RAW data packets in milliseconds (default is 100ms). If this timeout passes without new data packets while waiting for RAW data (`CAM_DATA_TAKING` flag set), command `0x0Ah` is sent to the camera (status check). If transfer is finished according to the device status, `CAM_DATA_TAKING` flag is cleared and the background thread stops waiting for RAW data (`WaitForRawData()` will return `false`) . Otherwise (status says that camera is still sending RAW packets), background thread is still waiting. Transfer is stopped unconditionally after 3rd data timeout in a row.

```
int GetCommandTimeout();
void SetCommandTimeout( int msec );
```

Gets/sets command response timeout in milliseconds (default is 100ms). If this timeout is reached while waiting for camera response packet, command is sent again.

```
void SetCommandRetries( int n );
```

Sets number of command retries; there will be only one attempt if `n = 0`. If first attempt of sending the command failed (timeout passed or camera response arrived with errors) command will be retransmitted `n` times. If all attempts failed – `SendCmd()` returns `false`.

Camera and NUDP socket states:

```
char GetNudpState( void );
char GetCamState( void );
```

Check NUDP communication socket and camera states.

Camera state bits: 00000ABC
A - `CAM_DATA_TAKING` flag;
B - `CAM_REFRESH_RUNNING` flag;
C - `CAM_CONNECTION_OPEN` flag.

NUDP socket status bits: 000A0BCD
A - `NUDP_DATA` flag;
B - `NUDP_CMD_ERR` flag;
C - `NUDP_WAITING` flag;
D - `NUDP_READY` flag.

Log files setup:

```
bool SetLogFilename( const std::string &fname );
bool SetErrLogFilename( const std::string &fname );
```

Set file names for all logs and error logs. No logging if not called. Log files setup must take place before calling `Open()` method, otherwise some of the refresh thread logs will be lost.
Additional logging to screen is done if compiled with D_DEBUG option.

Additional infos:

```
ipaddr_t GetLocalInterface( void );
port_t GetLocalPort( void );
```

Get LOCAL interface IP and UDP port number.

```
std::string GetLocalAddrStr( void );
```

Get LOCAL interface full address as string "ip:port".

```
ipaddr_t GetRemoteHost( void );
port_t GetRemotePort( void );
```

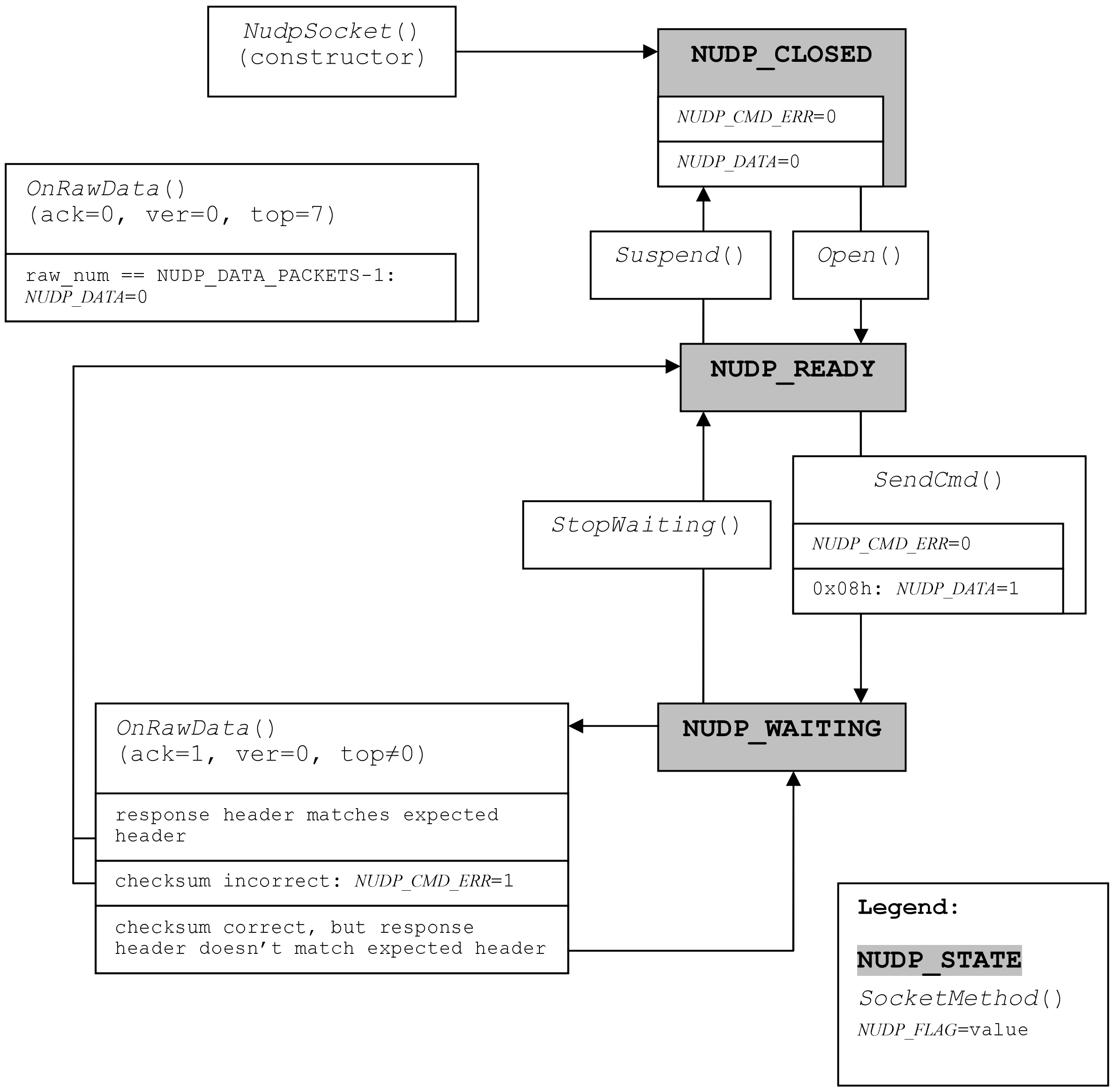
Get REMOTE camera IP and UDP port number.

```
std::string GetRemoteAddrStr( void );
```

Get REMOTE camera full address as string "ip:port".

2. **NudpSocket** – the basic class used to communicate with a camera; inherits from the *UdpSocket* (C++ Sockets Library) base class.

   Objects of this class are used internally by *CEthCamera*. Following chart shows possible state and flag changes. States and flags of the object used for sending commands and listening to the RAW transmissions may be read through `GetNudpState()` method of the *CEthCamera*.


NudpSocket() (constructor)
NUDP_CLOSED
NUDP_CMD_ERR=0
NUDP_DATA=0
OnRawData() (ack=0, ver=0, top=7)
raw_num == NUDP_DATA_PACKETS-1: NUDP_DATA=0
Suspend()
Open()
NUDP_READY
StopWaiting()
SendCmd()
NUDP_CMD_ERR=0
0x08h: NUDP_DATA=1
NUDP_WAITING
OnRawData() (ack=1, ver=0, top≠0)
response header matches expected header
checksum incorrect: NUDP_CMD_ERR=1
checksum correct, but response header doesn't match expected header
Legend:
NUDP_STATE
SocketMethod()
NUDP_FLAG=value