LPR Results: Developer's API

Updated:	**2018-02-15**
Latest version always at:	**https://goo.gl/ocR1EN** *

Common

Configuration file, log files, database reside in **%ALLUSERSPROFILE%\FF\NumberOk**.

To enable TCP server in NumberOk set configuration parameter "NetServer.Enable" to 1 (by default 0).
To enable TCP server in NumberOk3 you can use GUI interface
To enable get recognized screenshots from NumberOk set configuration parameter "NetServer_AttachScreenshotsToEvents" to 1 (by default 0).
To set quality of screenshots change configuration parameter "NetServer_AttachedScreenshotsQuality" (allow value from 10 to 100, by default 50). **Be aware: Max queue events limit to 1024. If you have slow link or overloaded network we suggest do not attach screenshots to events.**
NumberOk is waiting for connection on TCP port 7700.
Maximum incoming connections allowed - 4 (four).

Constants

```
#define MAX_MESSAGE_SIZE = ( 1 * 1024 * 1024 + sizeof( PacketHeader ) )
#define PACKET_TYPE_StartEventsTransmit          0x19
#define PACKET_TYPE_Event                        0x21
```

Structures

Packet header:

```
typedef struct PacketHeader
{
  UINT32  pType;            // packet type
  UINT32  len;                    // packet length
} PacketHeader, *PPacketHeader;
```

Event Packet:

```
typedef struct EventPacket {
  PacketHeader  header;                          // packet header
  unsigned char data[MAX_MESSAGE_SIZE];          // packet data
} EventPacket, * PEventPacket;
```

Packet structure

PacketHeader	Param1=Value1	0x05	ParamN=ValueN	0x00	0x00	Binary data

Procedure

1. Connect to host NumberOk running to tcp port 7700 (check firewall and IP filter settings if you can not connect). NumberOk will wait for subscribe packet during 1 second. If subscribe packet will not be received NumberOk close connection.
2. Subscribe to events:
   PacketHeader packet;
   packet.len = 0;
   packet.pType = PACKET_TYPE_StartEventsTransmit;
   // send subscribe packet
   // start receive events
3. NumberOk will start sending events (recognized events and channel status) to connected client.
4. Packet data (event text, string and binary data) consist of multiple pairs like “Param=Value”. Pairs divided with ASCII char 0x05. There is no ASCII char 0x05 after the last pair “Param=Value”.
5. If you set “NetServer_AttachScreenshotsToEvents” to 1 then after the last pair there are two chars ASCII 0x00 and attached binary data (JPG image). In this case special params added to event text: "BinaryDataCount" - count of binary data blocks, "BinaryData_%n_Size" - size of n-th data block. It is possible to send several binary blocks with one packet. Binary blocks have not delimited between each other. Binary data attached AFTER TEXT PART.
6. When you receive packet: packet header type (*pType* == PACKET_TYPE_Event), packet length (*len*) - length of transmitted packet. **Be aware: you can receive a part of event.**
7. If numberOk detects error during sending data, it will close connection and connected socket.

Event fields for LPR event:

"State"	“Detected” or “Updated”
"RuntimeID"	Internal
"MovementType"	“Undefined”, “IN”, “OUT”, “TryIN”, “TryOUT”
"MovementTypePrevious"	Previous movement (look at “MovementType”)
"LastMovementDetector"	int value [0..5]
"LastMovementDetectorText"	One of: "Undefined" - undetermined, "By movement in frame", "By single zone", "By connected zones", "By photoelements", "By fact of passing"
"IsUpdated"	if car was updated (coords) then "1", else field absent.
"IsTextUpdated"	if car was updated (number) then "1", else field absent.
"Text"	cars’ number
"Class"	always "Status"
"Type"	always "IAService.LPR"
"ZoneIndex"	int value [0..3], zone index
"ChannelIndex"	int value [0..1], channel index
"PlateStandart"	internal
"PlateTTL"	How long car will “live” in system

"KPP"	int value [0..3], checkpoint index
"FrameTimeStampMs"	frame timestamp in milliseconds
"ProcessTimeMs"	How long recognition was executed
"DetectedTimestampMS"	detection timestamp, milliseconds
"LastTimestampMS"	last detection timestamp, milliseconds
"PlateConfidence"	Plate recognition confidence
"CoordsX"	Plate's coordinates in frame, absolute values
"CoordsY"	
"CoordsWidth"	
"CoordsHeight"	
"SymbolsCount"	int value, how many symbols was recognized
"SymbolThreshold"	configuration parameter: threshold for symbol, float
"SymbolConfidence_%d"	float value. %d - [1..SymbolsCount]. Confidences of symbols.
"isVariableLength"	int value [0,1], configuration parameter.
========== NumberOk2 section begin ==========	
"EtalonId"	ID from table 'etalon'
"Owner"	Field 'description' of table 'etalon'
"PassMode"	Field 'passMode'' of table 'etalon'
"PassIntervalStart"	Field 'iBeginMs' of table 'etalon'
"PassIntervalEnd"	Field 'iEndMs'' of table 'etalon'
========== NumberOk2 section end ==========	
========== NumberOk3 section begin ==========	
"PlateColor"	int value[0..7]. Plate's color (from enum:

```
{
   COLOR_BLACK=0,
   COLOR_WHITE,
   COLOR_GRAY,
   COLOR_RED,
   COLOR_YELLOW,
   COLOR_GREEN,
   COLOR_BLUE,
   NUM_COLORS
};
```

"PlateColorConfidence"	float value[0..1]
"MovementType"	int values from enum:

```
{
     MovementUndetermined    0x00
     MovementTryIn           0x01
     MovementIN              0x02
     MovementTryOut          0x04
     MovementOUT             0x08
     MovementIN2Nd           0x10
     MovementAny             0x1F    Internal usage
     MovementNone            0xFF    Internal usage
};
```

"EtalonId" unsigned long long value. ID from table ‘cars’
"Owner" Field ‘description’ of table ‘cars’
"PassMode" Calculated value from:

```
#define  PassTypeNone      0x0000
#define  PassAutoDeny      0x0001
#define  PassAutoAllow     0x0002
#define  PassByCard        0x0004
#define  PassByOperator    0x0008
#define  Passed_Ok         0x0010
#define  Passed_violator   0x0020
#define  PassNoParkPlaces  0x0040
#define  PassParkingPlace  0x0080
#define  DenyByDate        0x0100
#define  DenyByCount       0x0200
#define  DenyByDuration    0x0400
```

"GroupId" unsigned long long value. ID from table ‘Groups’
"Group" text value. Group name from table ‘Groups’
"RecognizeCounter" int value. Detections counter.
"BinaryDataCount" int value[1..N]. Count of binary data in packet.
"BinaryData_%d_Size" unsigned long value. The size of %d-th part of data.
========== NumberOk3 section end ==========

Event fields for channel status:
"Class"always "Status"
"Type" always "VideoSignal";
"Channel" int value, [0,1]. Channel number from zero.
"State" One of: "Present", “Lost”. Channel state.
"IntState" integer state: 1 == "Present", 0 == “Lost”

NumberOk’s Log messages

NumberOk writes information about connecting clients and sending data to log file with log level 0xFF:

Client opens connection:
EventsSource: socket: HEX_SOCKET, new connection from a.b.c.d:port
HEX_SOCKET - socket hex value returned by accept();
a.b.c.d - remote client ip address;
port - remote client port.

Client closes connection:
EventsSource: socket HEX_SOCKET, close due NdNrNcUaRT. Clients count: N
HEX_SOCKET - socket hex value returned by accept()..
NdNrNcUaRT:
(Nd) Net down
(Nr) Net reset
(Nc) Not connected (send() error)

(U) Host unreach
(a) Connection aborted (by remote client)
(R) Connection reset (by remote client)
(T) Timeout
Clients count: N - current clients count
Information about successful data sending:
EventsSource: socket HEX_SOCKET, sent xxxxxxxx bytes. Ok.

Example linux TCP client for NumberOk

```
/*
  Example client for NumberOk
 */
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netinet/tcp.h>
#include <arpa/inet.h>
#include <sys/errno.h>
#include <sys/poll.h>
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>
#include <sys/time.h>

#define SERVER "10.0.1.168"
#define PORT   7700

#define PACKET_TYPE_StartEventsTransmit                0x19
// NumberOk3 supports connection without synchronization packet. Configured on NumberOk.
#define PACKET_TYPE_Event                              0x21

#define CONNECT_TIMEOUT_S 2

#define POLL_TIMEOUT_US  200000 // 0.2s

#define MIN(X,Y) ( (X) < (Y) ? (X) : (Y) )

typedef struct PacketHeader
{
    int32_t pType;
    int32_t len;
} PacketHeader, *PPacketHeader;

// 1 Mb
#define MAX_MESSAGE_SIZE ( 10 * 1024 * 1024 + sizeof( PacketHeader ) )

typedef struct Packet {
    PacketHeader  header;
    unsigned char data[MAX_MESSAGE_SIZE];
} Packet, * PPacket;

Packet staticPacket;
```

```
// Set socket options
void SetSocketOptions( int fd )
{
    if ( fd == -1 ) return;
    int res = -1;
    long fl = fcntl( fd, F_GETFL, 0 );
    if ( fl < 0 )
    {
         printf( "failed to get F_GETFL to socket %d | errno : %d\n", fd, errno );
    }
    else
    {
        if ( fcntl( fd,F_SETFL, fl | O_NONBLOCK ) < 0 )
        {
             printf( "failed to get F_SETFL to socket %d | errno : %d\n", fd, errno );
        }
    }

    struct timeval to = { CONNECT_TIMEOUT_S, 0 };
    res = setsockopt( fd, SOL_SOCKET, SO_SNDTIMEO, &to, sizeof( to ) );

    int flag = 1;
    if ( setsockopt( fd, IPPROTO_TCP, TCP_NODELAY, &flag, sizeof(int)) < 0 )
    {
         printf( "failed to set TCP_NODELAY to socket %d | errno : %d\n", fd, errno );
    }

    flag = 1;
    if ( setsockopt( fd, SOL_SOCKET, SO_KEEPALIVE, &flag, sizeof(int) ) < 0 )
    {
         printf( "failed to set SO_KEEPALIVE to socket %d | errno : %d\n", fd, errno );
    }

    flag = 5; // time to send keepalive after last data packet
    if ( setsockopt( fd, SOL_TCP, TCP_KEEPIDLE, &flag, sizeof(int) ) < 0 )
    {
         printf( "failed to set TCP_KEEPIDLE to socket %d | errno : %d\n", fd, errno );
    }

    flag = 3; // failed keepalive probes before connection would be marked as dead
    if ( setsockopt( fd, SOL_TCP, TCP_KEEPCNT, &flag,sizeof(int) ) < 0 )
    {
         printf( "failed to set TCP_KEEPCNT to socket %d | errno : %d\n", fd, errno );
    }

    flag = 3; // interval between probes
    if ( setsockopt( fd, SOL_TCP,TCP_KEEPINTVL, &flag, sizeof(int) ) < 0 )
    {
         printf( "failed to set TCP_KEEPINTVL to socket %d | errno : %d\n", fd, errno );
    }
}

int SelectSocket( int fd, unsigned int timeoutUs )
{
    if ( fd <= 0 )
    {
        return -1;
    }

    struct timeval timet;
```

```
    memset( &timet, 0, sizeof( timet ) );

    timet.tv_sec = timeoutUs / 1000 / 1000;
    timet.tv_usec= timeoutUs  - (timet.tv_sec * 1000 * 1000);
    int res = 0;

    unsigned int timeoutMs = timeoutUs < 1000 ? 1 : timeoutUs / 1000;

    while (1)
    {
        struct pollfd pollfd;
        memset( &pollfd, 0, sizeof( pollfd ) );
        pollfd.fd = fd;
        pollfd.events = POLLIN | POLLPRI;

        int eventsCount = poll( &pollfd, 1, timeoutMs );
        if ( eventsCount == 1 )
        {
            if ( ( pollfd.revents & ( POLLERR | POLLHUP | POLLNVAL | POLLPRI ) ) != 0x00 )
            {
                res = -1;
            }
            else if ( ( ( pollfd.revents & ( POLLIN ) ) != 0x00 ) )
            {
                res = 1;
            }
        }
        else
        {
            res = 0;
        }
        break;
     }
     return res;
}

int main( int argc, char ** argv )
{
    struct sockaddr_in nomerokaddr;
    int fd = socket( AF_INET, SOCK_STREAM, IPPROTO_TCP );
    int res;
    // create socket
    if ( fd > 0 )
    {
        SetSocketOptions( fd );

        // set address:port
        memset( &nomerokaddr, 0, sizeof( nomerokaddr ) );
        nomerokaddr.sin_family      = AF_INET;
        nomerokaddr.sin_addr.s_addr = inet_addr( SERVER );
        nomerokaddr.sin_port        = htons( PORT );

        socklen_t len = sizeof( nomerokaddr );

        // switch to blocking mode until connect
        long fl = fcntl( fd, F_GETFL, 0 );
        if ( fl >= 0 )
        {
            if ( fcntl( fd,F_SETFL, fl & ( ~O_NONBLOCK ) ) < 0 )
            {
```

```
                printf( "failed to set F_SETFL | errno : %d", errno );
            }
        }
        else
        {
            printf( "failed to get F_GETFL | errno : %d\n", errno );
        }
        // connect
        res = connect( fd,(struct sockaddr*)&nomerokaddr, len );
        if ( res == 0 )
        {
            // switch to non blocking mode
            fl = fcntl( fd, F_GETFL, 0 );
            if ( fl >= 0 )
            {
                if ( fcntl( fd, F_SETFL, fl | O_NONBLOCK ) < 0 )
                {
                    printf( "failed to set F_SETFL | errno : %d", errno );
                }
            }
            else
            {
                printf( "failed to get F_GETFL | errno : %d\n", errno );
            }
            // subscribe to events
            PacketHeader packet;
            packet.len   = 0;
            packet.pType = PACKET_TYPE_StartEventsTransmit;
            // send subscribe packet
            res = write( fd, (char *)&packet, sizeof( packet ) );
            if ( res == sizeof( PacketHeader ) )
            {
                res = 0;
                int receivd = 0;
//                              FILE * f = fopen( "./dump.bin", "ab" );
               int jpg_count = 0;
                while ( 1 )
                {
                    // wait for event
                    res = SelectSocket( fd, POLL_TIMEOUT_US );
                    if ( res == -1 )
                    {
                        break;
                    }
                    else if ( res == 1 )
                    {
                        // receive data
                        res = recv( fd, (char*)&staticPacket + receivd, MAX_MESSAGE_SIZE + sizeof(
PacketHeader ) - receivd, MSG_DONTWAIT );
                        if ( res > 0 )
                        {
//                                fwrite( (char*)&staticPacket + receivd, res, 1, f );
//                                fflush( f );
                            receivd += res;
                            printf( "received %d from %d\n", receivd, staticPacket.header.len + sizeof(
PacketHeader ) );
                            // is this event?
                            if ( staticPacket.header.pType == PACKET_TYPE_Event && staticPacket.header.len >
0 )
                            {
```

```
                                // we have all data here ?
                                if ( staticPacket.header.len <= ( receivd - sizeof( PacketHeader ) ) )
                                {
                                                                printf( "process received %d from %d\n", receivd,
staticPacket.header.len + sizeof( PacketHeader ) );
                                    char * stringData = (char *)staticPacket.data;
                                    int stringDataLen = strlen( stringData );
                                    char * strEnd = stringData + stringDataLen;
                                    // print to stdout
                                    const char * delim = "\05";
                                    char * token = strtok( stringData, delim );
                                    char * saveptr = 0;
                                                                    while ( token != 0 && token < strEnd )
                                                                    {
//                                                                      printf( "%s\n", token );
                                                                        if ( strstr( token, "BinaryData_1_Size"
) != 0  )

                                                                        {
                                                                                printf( "%s ======== ", token
);

                                                                                token += strlen(
"BinaryData_1_Size=" );

                                                                                saveptr = token;
                                                                                while ( *token  != *delim &&
*token != 0x00  )

                                                                                {
                                                                                    ++token;
                                                                                }
                                                                                *token = 0x00;
                                                                                int jpgLen = atoi( saveptr );
                                                                                *token = 0x05;
                                                                           char * jpgDta = stringData +
stringDataLen + 2;

                                                                           char jpgFileName[128];
                                                                           snprintf( jpgFileName, 127,
"./%d.jpg", jpg_count );

                                                                                printf( " %d %s\n", jpgLen,
jpgFileName );

                                                                           ++jpg_count;
                                                                           FILE * f = fopen( jpgFileName, "wb"
);

                                                                           fwrite( jpgDta, jpgLen, 1, f );
                                                                           fclose( f );
                                                                        }
                                                                        token = strtok( 0, delim );
                                                                    }

                                    // copy the rest of other packet if present
                                    int diff = receivd - ( staticPacket.header.len + sizeof( PacketHeader )
);
                                    printf( "diff %d, strlen %d\n============================\n", diff,
stringDataLen );
                                    if ( diff > 0 )
                                    {
                                        memcpy( (char *)&staticPacket, stringData + staticPacket.header.len,
diff );

                                        receivd = diff;
                                        memset( (char *)&staticPacket + receivd, 0x00, MAX_MESSAGE_SIZE -
receivd - 1 );
                                    }
```

```
                                else
                                {
                                    receivd = 0;
                                }
                                                        }
                            }
                        }
                        else
                        {
                            if ( res == 0 || ( ( res < 0 ) && ( errno == EFAULT ) ) )
                            {
                                if ( errno == EINPROGRESS )
                                {
                                    sleep( 1 );
                                    continue;
                                } // else break it
                            } // EINPROGRESS
                            // error
                            break;
                        }
                    }
                }
            }
            shutdown( fd, SHUT_RDWR );
        }
        else
        {
            printf( "connect(...) failed | errno : %d | %s\n", errno, strerror( errno ) );
        }
        close( fd );
    }
    else
    {
        printf( "socket(...) failed | errno : %d | %s\n", errno, strerror( errno ) );
    }
}
```