EDNet SDK Manual

EDNet SDK Manual

This document describes EDNet SDK and API for developers to make network client applications for DVR / NVR products.

Chapter#1.#Introduction

EDNet(DVR Network) SDK provides a network client library with ActiveX(OCX) control which can be used in Internet Explorer web browser. It is used for applications to connect and control DVR / NVR network servers.

Chapter#2.#EDNet C/C++ API Reference

Overview

EDNet library provides C/C++ API for applications to connect and control remote DVR / NT network servers.

API is thread-safe, so you can easily call any functions without multi-thread problem.

Since version 1.4.0, network server function is also supported.

Basic Types and Utilities

Basic Types and Utilities - various helper functions

Synopsis

```
#define TRUE
#define FALSE
#define EDNET_INT_TO_POINTER
#define EDNET_POINTER_TO_INT
#define EDNET_UINT_TO_POINTER
#define EDNET_POINTER_TO_UINT
#define EDNET_CHANNEL_MAX
#define EDNET_CHANNEL_AUDIO
#define E_CHANNEL_VIDEO_MAP_LEN
#define EDNET_PROPERTY_NAME_MAX

typedef EDNetFunc;
enum    EDNetLinkState;
enum    EDNetLinkHookType;
enum    EDNetConnectReplyError;
enum    EDNetRoleSystem;
enum    EDNetPropertyID;
enum    EDNetPropertyState;
enum    EDNetPropertyValue;
struct  EDNetPropertySet;
struct  EDNetSession;
enum    EDNetDeviceID;
enum    EDNetDevicePTZCommand;
enum    EDNetDeviceInfoCommand;
enum    EDNetPhoneCommand;
```

```
enum    EDNetDirType;
enum    EDNetMpegFileOpenError;
enum    EDNetMpegFileReadFormat;
enum    EDNetMpegFileReadFlag;
enum    EDNetMpegFileReadType;
enum    EDNetMpegFileSeekFlag;
enum    EDNetMpegFileSeekError;
enum    EDNetMpegFileCloseError;

int         ednet_init               (void);
void        ednet_exit               (void);
void        ednet_set_deinterlace    (int             deinterlace);
const char *ednet_get_link_state_name (EDNetLinkState  state);
const char *ednet_get_property_name   (EDNetPropertyID prop_id);
int         ednetx_init              (void);
void        ednetx_exit              (void);
int         ednet_test_bit           (const void     *addr,
                                      int             bit_nr);
void        ednet_set_bit            (void           *addr,
                                      int             bit_nr);
void        ednet_clear_bit          (void           *addr,
                                      int             bit_nr);
```

Description

These API provides basic types, constants, library initialization and miscellaneous helper functions.

Details

TRUE

```
#define TRUE                    (1)
```

FALSE

```
#define FALSE                   (0)
```

EDNET_INT_TO_POINTER

```
#define EDNET_INT_TO_POINTER (i)  ((void *) (long) (i))
```

EDNET_POINTER_TO_INT

```
#define EDNET_POINTER_TO_INT (p)  ((int) (long) (p))
```

EDNET_UINT_TO_POINTER

```
#define EDNET_UINT_TO_POINTER (i)   ((void *) (unsigned long) (i))
```

EDNET_POINTER_TO_UINT

```
#define EDNET_POINTER_TO_UINT (p)   ((unsigned int) (long) (p))
```

EDNET_CHANNEL_MAX

```
#define EDNET_CHANNEL_MAX       256
```

The number of video and audio channels used in EDNet library. Channel numbers are always between 0 and EDNET_CHANNEL_MAX - 1. Video channels are started from 0 to EDNET_CHANNEL_AUDIO - 1, audio channels are started from EDNET_CHANNEL_AUDIO to EDNET_CHANNEL_MAX - 1.

EDNET_CHANNEL_AUDIO

```
#define EDNET_CHANNEL_AUDIO     128
```

The base number of audio channels. So audio channels' number is always EDNET_CHANNEL_AUDIO + n, where n is between 0 and 127.

E_CHANNEL_VIDEO_MAP_LEN

```
#define E_CHANNEL_VIDEO_MAP_LEN (( - ) / 8)
```

EDNET_PROPERTY_NAME_MAX

```
#define EDNET_PROPERTY_NAME_MAX 64
```

EDNetFunc

```
typedef int (*EDNetFunc) (void *object, void *param, void *data);
```

Specifies the type of function which is called wihin EDNetClient hooks. The first parameter 'object' is caller client instance, the second parameter 'param' is hook information described in EDNetLinkHookType, and the last parameter 'data' is user-specific data, which has beed specified as the last parameter 'data' in ednet_client_add_hook() and ednet_client_remove_hook().

EDNetLinkState

```
typedef enum
{
  EDNET_LINK_STATE_OFFLINE = 0,
  EDNET_LINK_STATE_RESOLVE,
  EDNET_LINK_STATE_CONNECT,
  EDNET_LINK_STATE_INIT,
  EDNET_LINK_STATE_LOGIN,
  EDNET_LINK_STATE_PREPARE,
  EDNET_LINK_STATE_ONLINE,
  EDNET_LINK_STATE_MAX
} EDNetLinkState;
```

A 'link' stands for a connection between a network client object and a network server(DVR/NT). The EDNetLinkState is enumerative type for link state.

The client's current link state can be got by ednet_client_get_state() or notified by EDNET_LINK_HOOK_STATE.

EDNET_LINK_STATE_OFFLINE :
offline or disconnected

EDNET_LINK_STATE_RESOLVE :
resolving host name to IP address

EDNET_LINK_STATE_CONNECT :
connecting to the host

EDNET_LINK_STATE_INIT :
initializing the connection

EDNET_LINK_STATE_LOGIN :
authenticating with user name and password

EDNET_LINK_STATE_PREPARE :
preparing to fetch basic information from the host

EDNET_LINK_STATE_ONLINE :
ready to streaming and control remote hosts

EDNET_LINK_STATE_MAX :

EDNetLinkHookType

```
typedef enum
{
  EDNET_LINK_HOOK_STATE = 0,
  EDNET_LINK_HOOK_EVENT,
  EDNET_LINK_HOOK_FRAME,
  EDNET_LINK_HOOK_IMAGE,
  EDNET_LINK_HOOK_MAX
} EDNetLinkHookType;
```

These hook types are used with ednet_client_add_hook(), ednet_client_remove_hook() to add or remove hook functions for applications to get notified when some information or status are changed from EDNetClient objects.

EDNET_LINK_HOOK_STATE :
notified when a link state is changed. You can get client's state with ednet_client_get_state().

EDNET_LINK_HOOK_EVENT :
notified with EDNetEvent to pass information to applications.

EDNET_LINK_HOOK_FRAME :
notified with EDNetFrame to pass video / audio frames to applications.

EDNET_LINK_HOOK_IMAGE :
notified with EDNetFrame to pass video yuv images to create still images.

EDNET_LINK_HOOK_MAX :

EDNetConnectReplyError

```
typedef enum
{
  EDNET_CONNECT_REPLY_OK                    = 0,
  EDNET_CONNECT_REPLY_ERROR_RESOLVE,
  EDNET_CONNECT_REPLY_ERROR_CONNECT,
  EDNET_CONNECT_REPLY_ERROR_LOGIN,
  EDNET_CONNECT_REPLY_ERROR_PERMISSION,
  EDNET_CONNECT_REPLY_ERROR_CONNECTION_LIMIT
} EDNetConnectReplyError;
```

Describes connection errors.

EDNET_CONNECT_REPLY_OK :
success

EDNET_CONNECT_REPLY_ERROR_RESOLVE :
failed to resolve IP

EDNET_CONNECT_REPLY_ERROR_CONNECT :
failed to connect

EDNET_CONNECT_REPLY_ERROR_LOGIN :
failed to login due to invalid user and password

EDNET_CONNECT_REPLY_ERROR_PERMISSION :
failed to login due to insufficient permission

EDNET_CONNECT_REPLY_ERROR_CONNECTION_LIMIT :
failed to login due to reach connection limitation

EDNetRoleSystem

```
typedef enum
{
  EDNET_ROLE_SYSTEM_CONFIG  = (1 << 0),
  EDNET_ROLE_SYSTEM_SEARCH  = (1 << 1),
  EDNET_ROLE_SYSTEM_DEVICE  = (1 << 2),
  EDNET_ROLE_SYSTEM_NETWORK = (1 << 3),
  EDNET_ROLE_SYSTEM_BACKUP  = (1 << 4)
} EDNetRoleSystem;
```

Describes system role information.

EDNET_ROLE_SYSTEM_CONFIG :
can configure the system

EDNET_ROLE_SYSTEM_SEARCH :
can search the recorded files

EDNET_ROLE_SYSTEM_DEVICE :
can control devices

EDNET_ROLE_SYSTEM_NETWORK :
can connect from network

EDNET_ROLE_SYSTEM_BACKUP :
can backup the recorded files

EDNetPropertyID

```
typedef enum
{
  EDNET_PROPERTY_NAME           = 0,
  EDNET_PROPERTY_WIDTH,
  EDNET_PROPERTY_HEIGHT,
```

```
  EDNET_PROPERTY_CAPTURE_RATE,
  EDNET_PROPERTY_AUDIO_CHANNELS,
  EDNET_PROPERTY_SAMPLE_RATE,
  EDNET_PROPERTY_CODEC,
  EDNET_PROPERTY_BITRATE,
  EDNET_PROPERTY_GOP_SIZE,
  EDNET_PROPERTY_FRAME_RATE,
  EDNET_PROPERTY_CONTRAST,
  EDNET_PROPERTY_BRIGHTNESS,
  EDNET_PROPERTY_COLORNESS,
  EDNET_PROPERTY_HUE,
  EDNET_PROPERTY_RECORD_RATE,
  EDNET_PROPERTY_STATE,
  EDNET_PROPERTY_LINKED_AUDIO,
  EDNET_PROPERTY_MAX
} EDNetPropertyID;
```

Channel property types.

EDNET_PROPERTY_NAME :
channel name

EDNET_PROPERTY_WIDTH :
video width

EDNET_PROPERTY_HEIGHT :
video height

EDNET_PROPERTY_CAPTURE_RATE :
capture rate

EDNET_PROPERTY_AUDIO_CHANNELS :
the number of audio channels (1:mono, 2:stereo)

EDNET_PROPERTY_SAMPLE_RATE :
audio sample rate

EDNET_PROPERTY_CODEC :
codec, see EDNetCodecID

EDNET_PROPERTY_BITRATE :
video bitrate

EDNET_PROPERTY_GOP_SIZE :
video gop size

EDNET_PROPERTY_FRAME_RATE :
video frame rate

EDNET_PROPERTY_CONTRAST :
video contrast

EDNET_PROPERTY_BRIGHTNESS :
video brightness

EDNET_PROPERTY_COLORNESS :
video colorness

EDNET_PROPERTY_HUE :
video hue

EDNET_PROPERTY_RECORD_RATE :
video recording rate

EDNET_PROPERTY_STATE :
state, see EDNetPropertyState

EDNET_PROPERTY_LINKED_AUDIO :
linked audio channel

EDNET_PROPERTY_MAX :

EDNetPropertyState

```
typedef enum
{
  EDNET_PROPERTY_STATE_SIGNAL    = (1 << 0),
  EDNET_PROPERTY_STATE_PTZ_AVAIL = (1 << 1),
  EDNET_PROPERTY_STATE_HR        = (1 << 2),
  EDNET_PROPERTY_STATE_HIDDEN    = (1 << 3),
  EDNET_PROPERTY_STATE_AUDIO     = (1 << 4)
} EDNetPropertyState;
```

The state of the channel.

EDNET_PROPERTY_STATE_SIGNAL :
has valid signal (0 means loss)

EDNET_PROPERTY_STATE_PTZ_AVAIL :
has PTZ device

EDNET_PROPERTY_STATE_HR :
reserved

EDNET_PROPERTY_STATE_HIDDEN :
is hidden

EDNET_PROPERTY_STATE_AUDIO :
has linked audio

EDNetPropertyValue

```
typedef enum
{
  EDNET_PROPERTY_VALUE_INTEGER = 0,
  EDNET_PROPERTY_VALUE_STRING
} EDNetPropertyValue;
```

The type of channel property values.

EDNET_PROPERTY_VALUE_INTEGER :
integer value

EDNET_PROPERTY_VALUE_STRING :
string value

EDNetPropertySet

```
typedef struct
{
  char name[64];
  int  value[EDNET_PROPERTY_MAX];
} EDNetPropertySet;
```

char name[64] :

int value[EDNET_PROPERTY_MAX] :

EDNetSession

```
typedef struct
{
  int  sid;
  char addr[32];
  int  port;
  int  uid;
} EDNetSession;
```

int sid :
Session ID

char addr[32] :

int port :

int uid :
User ID

EDNetDeviceID

```
typedef enum
{
  EDNET_DEVICE_GENERAL        = 0x00,
  EDNET_DEVICE_PTZ            = 0x01,
  EDNET_DEVICE_MOTION         = 0x02,
  EDNET_DEVICE_ALARM          = 0x03,
  EDNET_DEVICE_SENSOR         = 0x04,
  EDNET_DEVICE_DISK           = 0x05,
  EDNET_DEVICE_VIDEO_LOSS     = 0x06,
  EDNET_DEVICE_RAID           = 0x07,
  EDNET_DEVICE_VIDEO_RECOVERY = 0x08,
  EDNET_DEVICE_EXTERNAL       = 0x40,
  EDNET_DEVICE_ANALOG         = 0x41,
  EDNET_DEVICE_CUSTOM         = 0x80
} EDNetDeviceID;
```

Event devices

EDNET_DEVICE_GENERAL :
General event

EDNET_DEVICE_PTZ :
PTZ event

EDNET_DEVICE_MOTION :
Motion detection event

EDNET_DEVICE_ALARM :
Replay event

EDNET_DEVICE_SENSOR :
Sensor event

EDNET_DEVICE_DISK :
Disk event

EDNET_DEVICE_VIDEO_LOSS :
Video signal loss event

EDNET_DEVICE_RAID :
Raid device event

EDNET_DEVICE_VIDEO_RECOVERY :
Video signal recovery event

EDNET_DEVICE_EXTERNAL :
External event

EDNET_DEVICE_ANALOG :
Analog input event

EDNET_DEVICE_CUSTOM :
User defined event

EDNetDevicePTZCommand

```
typedef enum
{
  EDNET_DEVICE_PTZ_MOVE              = (0x0001 << 16),
  EDNET_DEVICE_PTZ_MOVE_LEFT_UP      = (0x0001 << 16) | 0x0000,
  EDNET_DEVICE_PTZ_MOVE_UP           = (0x0001 << 16) | 0x0001,
  EDNET_DEVICE_PTZ_MOVE_RIGHT_UP     = (0x0001 << 16) | 0x0002,
  EDNET_DEVICE_PTZ_MOVE_LEFT         = (0x0001 << 16) | 0x0003,
  EDNET_DEVICE_PTZ_MOVE_STOP         = (0x0001 << 16) | 0x0004,
  EDNET_DEVICE_PTZ_MOVE_RIGHT        = (0x0001 << 16) | 0x0005,
  EDNET_DEVICE_PTZ_MOVE_LEFT_DOWN    = (0x0001 << 16) | 0x0006,
  EDNET_DEVICE_PTZ_MOVE_DOWN         = (0x0001 << 16) | 0x0007,
  EDNET_DEVICE_PTZ_MOVE_RIGHT_DOWN   = (0x0001 << 16) | 0x0008,
  EDNET_DEVICE_PTZ_MOVE_ZOOM_IN      = (0x0001 << 16) | 0x0009,
  EDNET_DEVICE_PTZ_MOVE_ZOOM_OUT     = (0x0001 << 16) | 0x000a,
  EDNET_DEVICE_PTZ_MOVE_FOCUS_FAR    = (0x0001 << 16) | 0x000b,
  EDNET_DEVICE_PTZ_MOVE_FOCUS_NEAR   = (0x0001 << 16) | 0x000c,
  EDNET_DEVICE_PTZ_STATE             = (0x0002 << 16),
  EDNET_DEVICE_PTZ_STATE_AUTOPAN_ON  = (0x0002 << 16) | 0x0000,
  EDNET_DEVICE_PTZ_STATE_AUTOPAN_OFF = (0x0002 << 16) | 0x0001,
  EDNET_DEVICE_PTZ_STATE_LIGHT_ON    = (0x0002 << 16) | 0x0002,
  EDNET_DEVICE_PTZ_STATE_LIGHT_OFF   = (0x0002 << 16) | 0x0003,
  EDNET_DEVICE_PTZ_PRESET_SET        = (0x0101 << 16),
  EDNET_DEVICE_PTZ_PRESET_MOVE       = (0x0102 << 16),
  EDNET_DEVICE_PTZ_PRESET_RESET      = (0x0103 << 16)
} EDNetDevicePTZCommand;
```

Commands to control PTZ devices

EDNET_DEVICE_PTZ_MOVE :
Mode

EDNET_DEVICE_PTZ_MOVE_LEFT_UP :
Left up

EDNET_DEVICE_PTZ_MOVE_UP :
Up

EDNET_DEVICE_PTZ_MOVE_RIGHT_UP :
Right up

EDNET_DEVICE_PTZ_MOVE_LEFT :
Left

EDNET_DEVICE_PTZ_MOVE_STOP :
Stop

EDNET_DEVICE_PTZ_MOVE_RIGHT :
Right

EDNET_DEVICE_PTZ_MOVE_LEFT_DOWN :
Left down

EDNET_DEVICE_PTZ_MOVE_DOWN :
Down

EDNET_DEVICE_PTZ_MOVE_RIGHT_DOWN :
Right down

EDNET_DEVICE_PTZ_MOVE_ZOOM_IN :
Zoom in

EDNET_DEVICE_PTZ_MOVE_ZOOM_OUT :
Zoom out

EDNET_DEVICE_PTZ_MOVE_FOCUS_FAR :
Focus far

EDNET_DEVICE_PTZ_MOVE_FOCUS_NEAR :
Focus near

EDNET_DEVICE_PTZ_STATE :
State

EDNET_DEVICE_PTZ_STATE_AUTOPAN_ON :
Autopan on

EDNET_DEVICE_PTZ_STATE_AUTOPAN_OFF :
Autopan off

EDNET_DEVICE_PTZ_STATE_LIGHT_ON :
Light on

EDNET_DEVICE_PTZ_STATE_LIGHT_OFF :
Light off

EDNET_DEVICE_PTZ_PRESET_SET :
Preset set

EDNET_DEVICE_PTZ_PRESET_MOVE :
Preset go

EDNET_DEVICE_PTZ_PRESET_RESET :
Preset reset

EDNetDeviceInfoCommand

```
typedef enum
{
  EDNET_DEVICE_SUB_CMD_DEVICES = 0x0001,
  EDNET_DEVICE_SUB_CMD_INFO    = 0x0002,
  EDNET_DEVICE_SUB_CMD_PERIOD  = 0x0003,
  EDNET_DEVICE_SUB_CMD_VALUE   = 0x0004
} EDNetDeviceInfoCommand;
```

Sub-commands to control PTZ devices

EDNET_DEVICE_SUB_CMD_DEVICES :
Request / Response of total device number

EDNET_DEVICE_SUB_CMD_INFO :
Information

EDNET_DEVICE_SUB_CMD_PERIOD :
Set the period of sending device status

EDNET_DEVICE_SUB_CMD_VALUE :
Request current device status

EDNetPhoneCommand

```
typedef enum
{
  EDNET_PHONE_COMMAND_CALL       = 0x0001,
  EDNET_PHONE_COMMAND_BROADCAST  = 0x0002,
  EDNET_PHONE_COMMAND_ACCEPT     = 0x0003,
  EDNET_PHONE_COMMAND_IGNORE     = 0x0004,
  EDNET_PHONE_COMMAND_LISTEN     = 0x0005,
  EDNET_PHONE_COMMAND_NOT_LISTEN = 0x0006,
  EDNET_PHONE_COMMAND_SPEAK      = 0x0007,
  EDNET_PHONE_COMMAND_NOT_SPEAK  = 0x0008
} EDNetPhoneCommand;
```

Bidirectional audio communication commands.

EDNET_PHONE_COMMAND_CALL :
Request to connect

EDNET_PHONE_COMMAND_BROADCAST :
Broadcast audio

EDNET_PHONE_COMMAND_ACCEPT :
Accept

EDNET_PHONE_COMMAND_IGNORE :
Cancel connection

EDNET_PHONE_COMMAND_LISTEN :
Listen audio

EDNET_PHONE_COMMAND_NOT_LISTEN :
Not to listen audio

EDNET_PHONE_COMMAND_SPEAK :
Speak audio

EDNET_PHONE_COMMAND_NOT_SPEAK :
Not to speak audio

EDNetDirType

```
typedef enum
{
  EDNET_DIR_TYPE_FILE           = 0x0001,
  EDNET_DIR_TYPE_DIR            = 0x0002,
  EDNET_DIR_TYPE_ROOT_PATH      = 0x0004,
  EDNET_DIR_TYPE_MASK_FULL_PATH = 0x0100
} EDNetDirType;
```

Directory types

EDNET_DIR_TYPE_FILE :
File

EDNET_DIR_TYPE_DIR :
Directory

EDNET_DIR_TYPE_ROOT_PATH :
Root of record directory

EDNET_DIR_TYPE_MASK_FULL_PATH :
Absolute path

EDNetMpegFileOpenError

```
typedef enum
{
  EDNET_MPEG_FILE_OPEN_ERROR_MPEG_DRIVER = 0xff01,
  EDNET_MPEG_FILE_OPEN_ERROR_OPEN        = 0xff02,
```

```
  EDNET_MPEG_FILE_OPEN_ERROR_GET_INFO    = 0xff03,
  EDNET_MPEG_FILE_OPEN_ERROR_INVALID     = 0xff04
} EDNetMpegFileOpenError;
```

Describe error types for opening the recorded file

EDNET_MPEG_FILE_OPEN_ERROR_MPEG_DRIVER :
driver error

EDNET_MPEG_FILE_OPEN_ERROR_OPEN :
open error

EDNET_MPEG_FILE_OPEN_ERROR_GET_INFO :
failed to get information

EDNET_MPEG_FILE_OPEN_ERROR_INVALID :
invalid file name

EDNetMpegFileReadFormat

```
typedef enum
{
  EDNET_MPEG_FILE_READ_FORMAT_YV12         = 0x01,
  EDNET_MPEG_FILE_READ_FORMAT_UYVY         = 0x02,
  EDNET_MPEG_FILE_READ_FORMAT_RGB16        = 0x03,
  EDNET_MPEG_FILE_READ_FORMAT_RGB24        = 0x04,
  EDNET_MPEG_FILE_READ_FORMAT_AUDIO_16_8K  = 0x10,
  EDNET_MPEG_FILE_READ_FORMAT_AUDIO_16_16K = 0x20,
  EDNET_MPEG_FILE_READ_FORMAT_AUDIO_16_32K = 0x30,
  EDNET_MPEG_FILE_READ_FORMAT_AUDIO_16_44K = 0x40,
  EDNET_MPEG_FILE_READ_FORMAT_AUDIO_16_48K = 0x50,
  EDNET_MPEG_FILE_READ_FORMAT_AUDIO_16_11K = 0x60,
  EDNET_MPEG_FILE_READ_FORMAT_AUDIO_8_8K   = 0x70,
  EDNET_MPEG_FILE_READ_FORMAT_AUDIO_8_11K  = 0x71,
  EDNET_MPEG_FILE_READ_FORMAT_AUDIO_8_16K  = 0x72,
  EDNET_MPEG_FILE_READ_FORMAT_AUDIO_8_32K  = 0x73,
  EDNET_MPEG_FILE_READ_FORMAT_AUDIO_8_44K  = 0x74,
  EDNET_MPEG_FILE_READ_FORMAT_AUDIO_8_48K  = 0x75
} EDNetMpegFileReadFormat;
```

Describes frame formats which are read from the remote server.

EDNET_MPEG_FILE_READ_FORMAT_YV12 :
YV12 video

EDNET_MPEG_FILE_READ_FORMAT_UYVY :
UYVY video

EDNET_MPEG_FILE_READ_FORMAT_RGB16 :
RGB16 video

EDNET_MPEG_FILE_READ_FORMAT_RGB24 :
RGB24 video

EDNET_MPEG_FILE_READ_FORMAT_AUDIO_16_8K :
16bit 8000Hz audio

EDNET_MPEG_FILE_READ_FORMAT_AUDIO_16_16K :
16bit 16000Hz audio

EDNET_MPEG_FILE_READ_FORMAT_AUDIO_16_32K :
16bit 32000Hz audio

EDNET_MPEG_FILE_READ_FORMAT_AUDIO_16_44K :
16bit 44000Hz audio

EDNET_MPEG_FILE_READ_FORMAT_AUDIO_16_48K :
16bit 48000Hz audio

EDNET_MPEG_FILE_READ_FORMAT_AUDIO_16_11K :
16bit 11000Hz audio

EDNET_MPEG_FILE_READ_FORMAT_AUDIO_8_8K :
8bit 8000Hz audio

EDNET_MPEG_FILE_READ_FORMAT_AUDIO_8_11K :
8bit 32000Hz audio

EDNET_MPEG_FILE_READ_FORMAT_AUDIO_8_16K :
8bit 16000Hz audio

EDNET_MPEG_FILE_READ_FORMAT_AUDIO_8_32K :
8bit 32000Hz audio

EDNET_MPEG_FILE_READ_FORMAT_AUDIO_8_44K :
8bit 44000Hz audio

EDNET_MPEG_FILE_READ_FORMAT_AUDIO_8_48K :
8bit 48000Hz audio

EDNetMpegFileReadFlag

```
typedef enum
{
  EDNET_MPEG_FILE_READ_FLAG_KEY_FRAME         = 0x01,
  EDNET_MPEG_FILE_READ_FLAG_ERROR_INVALID     = 0xf1,
  EDNET_MPEG_FILE_READ_FLAG_ERROR_FILE_ID     = 0xf2,
  EDNET_MPEG_FILE_READ_FLAG_ERROR_FORMAT      = 0xf3,
  EDNET_MPEG_FILE_READ_FLAG_ERROR_READ        = 0xf4,
```

```
  EDNET_MPEG_FILE_READ_FLAG_ERROR_MPEG_DRIVER = 0xf5,
  EDNET_MPEG_FILE_READ_FLAG_ERROR_MASK        = 0xf0
} EDNetMpegFileReadFlag;
```

Describes frame status flags.

EDNET_MPEG_FILE_READ_FLAG_KEY_FRAME :
is a key frame

EDNET_MPEG_FILE_READ_FLAG_ERROR_INVALID :
failed to erad due to invalid parameters

EDNET_MPEG_FILE_READ_FLAG_ERROR_FILE_ID :
failed to read due to invalid file id

EDNET_MPEG_FILE_READ_FLAG_ERROR_FORMAT :
failed to read due to unsupported format

EDNET_MPEG_FILE_READ_FLAG_ERROR_READ :
failed to read due to system read call error

EDNET_MPEG_FILE_READ_FLAG_ERROR_MPEG_DRIVER :
failed to read due to driver error

EDNET_MPEG_FILE_READ_FLAG_ERROR_MASK :
error masking value

EDNetMpegFileReadType

```
typedef enum
{
  EDNET_MPEG_FILE_READ_TYPE_BOTH  = 0,
  EDNET_MPEG_FILE_READ_TYPE_VIDEO = 1
} EDNetMpegFileReadType;
```

Describes frame types which should be read.

EDNET_MPEG_FILE_READ_TYPE_BOTH :
read video and audio frames

EDNET_MPEG_FILE_READ_TYPE_VIDEO :
read only video frames

EDNetMpegFileSeekFlag

```
typedef enum
{
  EDNET_MPEG_FILE_SEEK_FLAG_PREV_KEY_FRAME = 0x01,
```

```
  EDNET_MPEG_FILE_SEEK_FLAG_NEXT_KEY_FRAME = 0x02,
  EDNET_MPEG_FILE_SEEK_FLAG_KEY_FRAME      = 0x10
} EDNetMpegFileSeekFlag;
```

Describes frame seek options.

EDNET_MPEG_FILE_SEEK_FLAG_PREV_KEY_FRAME :
seek to previous key frame

EDNET_MPEG_FILE_SEEK_FLAG_NEXT_KEY_FRAME :
seek to next key frame

EDNET_MPEG_FILE_SEEK_FLAG_KEY_FRAME :
seek to current key frame

EDNetMpegFileSeekError

```
typedef enum
{
  EDNET_MPEG_FILE_SEEK_ERROR_INVALID     = 0xf1,
  EDNET_MPEG_FILE_SEEK_ERROR_FILE_ID     = 0xf2,
  EDNET_MPEG_FILE_SEEK_ERROR_SEEK        = 0xf3,
  EDNET_MPEG_FILE_SEEK_ERROR_MPEG_DRIVER = 0xf4
} EDNetMpegFileSeekError;
```

Describes seek status flags.

EDNET_MPEG_FILE_SEEK_ERROR_INVALID :
failed to seek due to invalid parameters

EDNET_MPEG_FILE_SEEK_ERROR_FILE_ID :
failed to seek due to invalid file id

EDNET_MPEG_FILE_SEEK_ERROR_SEEK :
failed to seek due to system seek call error

EDNET_MPEG_FILE_SEEK_ERROR_MPEG_DRIVER :
failed to seek due to driver error

EDNetMpegFileCloseError

```
typedef enum
{
  EDNET_MPEG_FILE_CLOSE_ERROR_INVALID     = 0xf1,
  EDNET_MPEG_FILE_CLOSE_ERROR_FILD_ID     = 0xf2,
  EDNET_MPEG_FILE_CLOSE_ERROR_MPEG_DRIVER = 0xf3
} EDNetMpegFileCloseError;
```

Descrubes close status flags.

EDNET_MPEG_FILE_CLOSE_ERROR_INVALID :
failed to close due to invalid parameters

EDNET_MPEG_FILE_CLOSE_ERROR_FILD_ID :
failed to close due to invalid file id

EDNET_MPEG_FILE_CLOSE_ERROR_MPEG_DRIVER :
failed to close due to driver error

ednet_init ()

```
int         ednet_init              (void);
```

Initializes EDNet library.

If you want to initialize extra display / sound modules at a time, use ednetx_init() function.

Returns :
0 if success, otherwise -1.

ednet_exit ()

```
void        ednet_exit              (void);
```

Finalizes EDNet library.

ednet_set_deinterlace ()

```
void        ednet_set_deinterlace     (int            deinterlace);
```

Determines whether deinterlace filter is applied to high resolution video frames (height 〉 288) after they are decoded from original MPEG-4 frames. If it's set to TRUE, more CPU resource will be required.

deinterlace :
TRUE if deinterlace filter should be applied

ednet_get_link_state_name ()

```
const char *ednet_get_link_state_name (EDNetLinkState  state);
```

Returns a string of the state.

state :
the state of the client

Returns :

a string, which you must not free

ednet_get_property_name ()

```
const char *ednet_get_property_name   (EDNetPropertyID prop_id);
```

Returns a string of the property name.

prop_id :

channel's property ID

Returns :

a string, which you must not free

ednetx_init ()

```
int         ednetx_init               (void);
```

Initializes EDNet library including display / sound modules.

If you want to initialize just core network module, use ednet_init() function.

Returns :

0 if success, otherwise -1.

ednetx_exit ()

```
void        ednetx_exit               (void);
```

Finalizes EDNet library including display / sound modules.

ednet_test_bit ()

```
int         ednet_test_bit            (const void     *addr,
                                       int             bit_nr);
```

ednet_set_bit ()

```
void        ednet_set_bit             (void           *addr,
                                       int             bit_nr);
```

ednet_clear_bit ()

```
void        ednet_clear_bit          (void          *addr,
                                      int            bit_nr);
```

Network Clients

Network Clients - support for network connection

Synopsis

```
#define ednet_client_destroy

typedef EDNetClient;

EDNetClient   *ednet_client_new                    (void);
void           ednet_client_ref                    (EDNetClient          *client);
void           ednet_client_unref                  (EDNetClient          *client);
int            ednet_client_add_hook               (EDNetClient          *client,
                                                    EDNetLinkHookType     hook,
                                                    EDNetFunc             func,
                                                    void                 *data);
int            ednet_client_remove_hook            (EDNetClient          *client,
                                                    EDNetLinkHookType     hook,
                                                    EDNetFunc             func,
                                                    void                 *data);
EDNetLinkState ednet_client_get_state              (EDNetClient          *client);
void           ednet_client_get_product            (EDNetClient          *client,
                                                    int                  *type,
                                                    int                  *id);
int            ednet_client_run                    (EDNetClient          *client,
                                                    int                   timeout);
int            ednet_client_main                   (EDNetClient          *client);
int            ednet_client_quit                   (EDNetClient          *client);
int            ednet_client_connect                (EDNetClient          *client,
                                                    const char           *host,
                                                    int                   port,
                                                    const char           *user,
                                                    const char           *password);
int            ednet_client_shutdown               (EDNetClient          *client);
int            ednet_client_request_channel_list   (EDNetClient          *client);
int            ednet_client_request_channel_properties (EDNetClient     *client,
                                                    int                   channel);
```

```
int             ednet_client_change_channel_property    (EDNetClient          *client,
                                                         int                   channel,
                                                         EDNetPropertyID       pid,
                                                         const char            *pstr,
                                                         int                   pval);
int             ednet_client_add_wanted_frame_format    (EDNetClient          *client,
                                                         int                   channel,
                                                         EDNetFrameFormat      fmt,
                                                         int                   width,
                                                         int                   height);
int             ednet_client_remove_wanted_frame_format (EDNetClient          *client,
                                                         int                   channel,
                                                         EDNetFrameFormat      fmt,
                                                         int                   width,
                                                         int                   height);
int             ednet_client_request_stream             (EDNetClient          *client,
                                                         int                   channel,
                                                         int                   request);
int             ednet_client_control_device             (EDNetClient          *client,
                                                         EDNetDeviceID         dev_id,
                                                         int                   dev_no,
                                                         EDNetDevicePTZCommand ptz);
int             ednet_client_phone                      (EDNetClient          *client,
                                                         EDNetPhoneCommand     cmd,
                                                         int                   src_channel,
                                                         int                   dest_channel,
                                                         int                   flags);
int             ednet_client_send_frame                 (EDNetClient          *client,
                                                         EDNetFrame            *frame);
int             ednet_client_send_channel_list          (EDNetClient          *client,
                                                         int                   total,
                                                         int                   *channels);
int             ednet_client_send_channel_property      (EDNetClient          *client,
                                                         int                   channel,
                                                         EDNetPropertyID       pid,
                                                         const char            *pstr,
                                                         int                   pval);
int             ednet_client_reply_stream               (EDNetClient          *client,
                                                         int                   channel,
                                                         int                   reply);
int             ednet_client_request_config             (EDNetClient          *client,
                                                         const char            *section,
                                                         const char            *key);
int             ednet_client_change_config              (EDNetClient          *client,
                                                         const char            *section,
```

```
                                                const char          *key,
                                                const char          *value);
int             ednet_client_request_dir        (EDNetClient        *client,
                                                 EDNetDirType        req_type,
                                                 unsigned int        req_id,
                                                 const char          *path);
int             ednet_client_request_mpeg_file_open     (EDNetClient        *client,
                                                 unsigned int        req_id,
                                                 unsigned int        file_id,
                                                 const char          *name);
int             ednet_client_request_mpeg_file_read     (EDNetClient        *client,
                                                 unsigned int        req_id,
                                                 unsigned int        file_id,
                                                 EDNetMpegFileReadType read_type);
int             ednet_client_request_mpeg_file_seek     (EDNetClient        *client,
                                                 unsigned int        req_id,
                                                 unsigned int        file_id,
                                                 unsigned int        idx,
                                                 EDNetMpegFileSeekFlag flag);
int             ednet_client_request_mpeg_file_close    (EDNetClient        *client,
                                                 unsigned int        req_id,
                                                 unsigned int        file_id);
int             ednet_client_request_file_transfer      (EDNetClient        *client,
                                                 unsigned int        req_id,
                                                 const char          *filename);
int             ednet_client_cancel_file_transfer       (EDNetClient        *client,
                                                 unsigned int        req_id);
```

Description

The EDNetClient is the core of EDNet SDK. All network transactions between a client and a DVR / NT server handled by this network client object.

You can order a client to control a remote server with ednet_client_*() functions. The video / audio frames and server informations received from remote servers can be notified by hooking network events with ednet_client_add_hook().

Multiple EDNetClient can be created and managed for control multiple servers simultaneously without no worry.

Details

ednet_client_destroy

```
#define ednet_client_destroy (client)          (client)
```

EDNetClient

```
typedef struct _EDNetClient EDNetClient;
```

A network client object to connect and control remote DVR / NT servers.

ednet_client_new ()

```
EDNetClient   *ednet_client_new                       (void);
```

Creates a new EDNetClient.

Returns :

a new EDNetClient, NULL if an error occurred.

ednet_client_ref ()

```
void          ednet_client_ref                        (EDNetClient          *client);
```

Adds a reference to the client.

client :

a EDNetClient

ednet_client_unref ()

```
void          ednet_client_unref                      (EDNetClient          *client);
```

Removes a reference from the client, deallocating the client if no references remain.

client :

a EDNetClient

ednet_client_add_hook ()

```
int           ednet_client_add_hook                   (EDNetClient          *client,
                                                       EDNetLinkHookType     hook,
                                                       EDNetFunc             func,
                                                       void                 *data);
```

Adds a hook function to be called when specified hook type occur. User data will be passed as the last parameter of the hook function.

Several hook functions can be added to the same hook type.

Hook functions are called within a thread in which the library executes, by calling ednet_client_run() or ednet_client_main().

client :
a EDNetClient

hook :
hook type

func :
callback function

data :
user data

Returns :
0 on success, -1 if an error occurred.

ednet_client_remove_hook ()

```
int           ednet_client_remove_hook              (EDNetClient          *client,
                                                     EDNetLinkHookType     hook,
                                                     EDNetFunc             func,
                                                     void                  *data);
```

Removes a hook function for the hook type which has added with user data.

client :
a EDNetClient

hook :
hook type

func :
callback function

data :
user data

Returns :
0 on success, -1 if an error occurred.

ednet_client_get_state ()

```
EDNetLinkState ednet_client_get_state                (EDNetClient          *client);
```

Returns the state of the client.

client：

a EDNetClient

Returns：

current state, see EDNetLinkState

ednet_client_get_product ()

```
void           ednet_client_get_product              (EDNetClient          *client,
                                                      int                   *type,
                                                      int                   *id);
```

Returns product information about the connected server.

client：

a EDNetClient

type：

a pointer to store product type

id：

a pointer to store product id

ednet_client_run ()

```
int            ednet_client_run                      (EDNetClient          *client,
                                                      int                    timeout);
```

Runs the main loop to process network transaction and application requests for a time.

client：

a EDNetClient

timeout：

an upper limit on the time for which it will block, in milliseconds. Specifying a negative value means an infinite timeout.

Returns：

0 on success, −1 if an error occurred.

ednet_client_main ()

```
int             ednet_client_main                       (EDNetClient          *client);
```

Runs the main loop to process network transaction and application requests until ednet_client_quit() is called.

If you want to manage network processing in a seperated thread, this function may be suitable for that situation.

client :
a EDNetClient

Returns :
0 on success, -1 if an error occurred.

ednet_client_quit ()

```
int             ednet_client_quit                       (EDNetClient          *client);
```

Makes the invocation of the main loop return.

This function can be called within other threads in which the main loop is running.

client :
a EDNetClient

Returns :
0 on success, -1 if an error occurred.

ednet_client_connect ()

```
int             ednet_client_connect                    (EDNetClient          *client,
                                                         const char           *host,
                                                         int                   port,
                                                         const char           *user,
                                                         const char           *password);
```

Starts connecting to the remote DVR / NT server.

The result of connection is returned via EDNET_EVENT_CONNECT_REPLY event within EDNET_LINK_HOOK_EVENT hook. If the connection is established, the state of the client is changed to EDNET_LINK_STATE_ONLINE. You can also track client's state with EDNET_LINK_HOOK_STATE hook.

client :
a EDNetClient

host :
host address

`port` :
port number, use default 8081 if this value is zero.

`user` :
user name

`password` :
password

Returns :
0 on normal condition, 1 if it's not off-line state.

ednet_client_shutdown ()

```
int            ednet_client_shutdown                (EDNetClient          *client);
```

Disconnects the connection from the remote DVR / NT server.

The result of a shutdown is returned via EDNET_EVENT_SHUTDOWN_REPLY event within EDNET_LINK_HOOK_EVENT hook.

`client` :
a EDNetClient

Returns :
0 on normal condition, -1 if a error occurred.

ednet_client_request_channel_list ()

```
int            ednet_client_request_channel_list       (EDNetClient          *client);
```

Requests the server to retrieve the list of channels.

The result is returned via EDNET_EVENT_CHANNEL_LIST event within EDNET_LINK_HOOK_EVENT hook.

Note that EDNET_EVENT_CHANNEL_LIST event is signaled automatically after the connection is established.

`client` :
a EDNetClient

Returns :
0 on success, -1 if a error occurred.

ednet_client_request_channel_properties ()

```
int           ednet_client_request_channel_properties (EDNetClient         *client,
                                                       int                  channel);
```

Requests the server to retrieve the properties of channels.

The result is returned via EDNET_EVENT_CHANNEL_PROPERTY event within EDNET_LINK_HOOK_EVENT hook.

Note that EDNET_EVENT_CHANNEL_PROPERTY events for all channels are signaled automatically after the connection is established.

client :
a EDNetClient

channel :
channel number, −1 means all channels.

Returns :
0 on success, −1 if a error occurred.

ednet_client_change_channel_property ()

```
int           ednet_client_change_channel_property    (EDNetClient         *client,
                                                       int                  channel,
                                                       EDNetPropertyID      pid,
                                                       const char          *pstr,
                                                       int                  pval);
```

Requests the server to change the property of a channel.

If 'pstr' is NULL, 'pval' parameter is used.

The result is returned via EDNET_EVENT_CHANNEL_PROPERTY event within EDNET_LINK_HOOK_EVENT hook.

client :
a EDNetClient

channel :
channel number

pid :
property id

pstr :
a string of the property, or NULL

pval :
a integer value of property

Returns :

0 on success, -1 if a error occurred.

ednet_client_add_wanted_frame_format ()

```
int             ednet_client_add_wanted_frame_format    (EDNetClient          *client,
                                                         int                   channel,
                                                         EDNetFrameFormat      fmt,
                                                         int                   width,
                                                         int                   height);
```

Adds a frame format in which the application want to received via EDNetFrame buffer within EDNET_LINK_HOOK_FRAME hook.

Even if you adds several formats, multiple frames in formats converted from a original received frame will be signaled. So, you must always check attributes of EDNetFrame buffers.

client :

a EDNetClient

channel :

channel number

fmt :

frame format

width :

the width of the video frame

height :

the height of the video frame

Returns :

0 on success, -1 if a error occurred.

ednet_client_remove_wanted_frame_format ()

```
int             ednet_client_remove_wanted_frame_format (EDNetClient          *client,
                                                         int                   channel,
                                                         EDNetFrameFormat      fmt,
                                                         int                   width,
                                                         int                   height);
```

Removes a frame format from the list of formats for specified channel.

See the description of ednet_client_add_wanted_frame_format().

client：
a EDNetClient

channel：
channel number

fmt：
frame format

width：
the width of the video frame

height：
the height of the video frame

Returns：
0 on success, -1 if a error occurred.

ednet_client_request_stream ()

```
int            ednet_client_request_stream            (EDNetClient          *client,
                                                       int                   channel,
                                                       int                   request);
```

Requests the server to control live streaming of a channel.

The result is returned via EDNET_EVENT_STREAM_REQUEST_REPLY event within EDNET_LINK_HOOK_EVENT hook.

client：
a EDNetClient

channel：
channel number

request：
TRUE for start, FALSE for cancel

Returns：
0 on success, -1 if a error occurred.

ednet_client_control_device ()

```
int            ednet_client_control_device            (EDNetClient          *client,
                                                       EDNetDeviceID         dev_id,
                                                       int                   dev_no,
                                                       EDNetDevicePTZCommand ptz);
```

Requests the server to control attached devices.

client :
a EDNetClient

dev_id :
device ID

dev_no :
device number

ptz :
device control command

Returns :
0 on success, -1 if a error occurred.

ednet_client_phone ()

```
int             ednet_client_phone                      (EDNetClient          *client,
                                                         EDNetPhoneCommand     cmd,
                                                         int                   src_channel,
                                                         int                   dest_channel,
                                                         int                   flags);
```

Transmits bidirectional audio communication (phone) commands.

client :
a EDNetClient

cmd :
phone command

src_channel :
a local audio channel number

dest_channel :
a remote audio channel number

flags :
reserved

Returns :
0 on success, -1 if a error occurred.

ednet_client_send_frame ()

```
int             ednet_client_send_frame                 (EDNetClient          *client,
```

```
                                                EDNetFrame          *frame);
```

Transmits audio frames to the remote DVR / NT server.

Note that the channel number of a frame buffer must be EDNET_CHANNEL_AUDIO and the other fields also should be filled with valid information.

This function is used for bidirectional audio communication.

client :
a EDNetClient

frame :
a frame buffer to send

Returns :
0 on success, -1 if a error occurred.

ednet_client_send_channel_list ()

```
int             ednet_client_send_channel_list          (EDNetClient         *client,
                                                         int                  total,
                                                         int                 *channels);
```

Transmits the list of audio channels to the remote DVR / NT server.

Note that 'total' must be 1, and channels must indicate the array which has EDNET_CHANNEL_AUDIO in the first element.

This function is used for bidirectional audio communication.

client :
a EDNetClient

total :
the number of audio channels

channels :
an array of channels

Returns :
0 on success, -1 if a error occurred.

ednet_client_send_channel_property ()

```
int             ednet_client_send_channel_property      (EDNetClient         *client,
                                                         int                  channel,
```

```
                                                    EDNetPropertyID       pid,
                                                    const char           *pstr,
                                                    int                   pval);
```

Transmits the property of a channel to the remote DVR / NT server.

If 'pstr' is NULL, 'pval' parameter is used.

This function is used for bidirectional audio communication.

client :
a EDNetClient

channel :
channel number

pid :
property id

pstr :
a string of the property, or NULL

pval :
a integer value of property

Returns :
0 on success, -1 if a error occurred.

ednet_client_reply_stream ()

```
int           ednet_client_reply_stream             (EDNetClient         *client,
                                                     int                   channel,
                                                     int                   reply);
```

Transmits the reply for live streaming request to the remote DVR / NT server.

This function is used for bidirectional audio communication.

client :
a EDNetClient

channel :
channel number

reply :
TRUE for streaming started, FALSE for streaming cancelled

Returns :
0 on success, -1 if a error occurred.

ednet_client_request_config ()

```
int             ednet_client_request_config             (EDNetClient          *client,
                                                         const char           *section,
                                                         const char           *key);
```

Requests the server to retrieve the configuration value for specified section and key.

The result is returned via EDNET_EVENT_CONFIG_REPLY event within EDNET_LINK_HOOK_EVENT hook.

client :
a EDNetClient

section :
section name

key :
key name

Returns :
0 on success, −1 if a error occurred.

ednet_client_change_config ()

```
int             ednet_client_change_config              (EDNetClient          *client,
                                                         const char           *section,
                                                         const char           *key,
                                                         const char           *value);
```

Requests the server to change the configuration value for specified section and key.

The result is returned via EDNET_EVENT_CONFIG_REPLY event within EDNET_LINK_HOOK_EVENT hook.

client :
a EDNetClient

section :
section name

key :
key name

value :
value name

Returns :

0 on success, -1 if a error occurred.

ednet_client_request_dir ()

```
int           ednet_client_request_dir              (EDNetClient         *client,
                                                     EDNetDirType         req_type,
                                                     unsigned int         req_id,
                                                     const char          *path);
```

Requests the server to retrieve the entries of specified directory.

The result is returned via EDNET_EVENT_DIR_REPLY event within EDNET_LINK_HOOK_EVENT hook. The given request id will be filled in EDNetEventDirReply:req_id in order to distinguish one from multiple requests.

This function is used for remote search and replay.

client :

a EDNetClient

req_type :

directory type

req_id :

request id

path :

directory path

Returns :

0 on success, -1 if a error occurred.

ednet_client_request_mpeg_file_open ()

```
int           ednet_client_request_mpeg_file_open   (EDNetClient         *client,
                                                     unsigned int         req_id,
                                                     unsigned int         file_id,
                                                     const char          *name);
```

Requests the server to open a recorded MPEG file.

The result is returned via EDNET_EVENT_MPEG_FILE_OPEN_REPLY event within EDNET_LINK_HOOK_EVENT hook. The given request id and file id will be filled in EDNetEventMpegFileOpenReply:req_id and EDNetEventMpegFileOpenReply:file_id in order to distinguish one from multiple requests.

This function is used for remote search and replay.

client :
a EDNetClient

req_id :
request id

file_id :
file id

name :
file name

Returns :
0 on success, -1 if a error occurred.

ednet_client_request_mpeg_file_read ()

```
int            ednet_client_request_mpeg_file_read     (EDNetClient          *client,
                                                        unsigned int          req_id,
                                                        unsigned int          file_id,
                                                        EDNetMpegFileReadType read_type);
```

Requests the server to read a frame from recorded MPEG file.

The result is returned via EDNET_EVENT_MPEG_FILE_READ_REPLY event within EDNET_LINK_HOOK_EVENT hook. The given request id will be filled in EDNetEventMpegFileReadReply:req_id in order to distinguish one from multiple requests.

This function is used for remote search and replay.

client :
a EDNetClient

req_id :
request id

file_id :
file id

read_type :
reading method

Returns :
0 on success, -1 if a error occurred.

ednet_client_request_mpeg_file_seek ()

```
int            ednet_client_request_mpeg_file_seek     (EDNetClient          *client,
```

```
                                                    unsigned int          req_id,
                                                    unsigned int          file_id,
                                                    unsigned int          idx,
                                                    EDNetMpegFileSeekFlag flag);
```

Requests the server to seek in recorded MPEG file.

The result is returned via EDNET_EVENT_MPEG_FILE_SEEK_REPLY event within EDNET_LINK_HOOK_EVENT hook. The given request id will be filled in EDNetEventMpegFileSeekReply:req_id in order to distinguish one from multiple requests.

This function is used for remote search and replay.

client :
a EDNetClient

req_id :
request id

file_id :
file id

idx :
video frame index

flag :
seek method

Returns :
0 on success, -1 if a error occurred.

ednet_client_request_mpeg_file_close ()

```
int           ednet_client_request_mpeg_file_close    (EDNetClient          *client,
                                                       unsigned int          req_id,
                                                       unsigned int          file_id);
```

Requests the server to close the recorded MPEG file.

The result is returned via EDNET_EVENT_MPEG_FILE_CLOSE_REPLY event within EDNET_LINK_HOOK_EVENT hook. The given request id will be filled in EDNetEventMpegFileCloseReply:req_id in order to distinguish one from multiple requests.

This function is used for remote search and replay.

client :
a EDNetClient

req_id :
request id

file_id :
file id

Returns :
0 on success, -1 if a error occurred.

ednet_client_request_file_transfer ()

```
int           ednet_client_request_file_transfer      (EDNetClient          *client,
                                                       unsigned int          req_id,
                                                       const char           *filename);
```

Downloads specified recorded file from the remote server.

The result is returned via EDNET_EVENT_FILE_GET event within EDNET_LINK_HOOK_EVENT hook. The given request id will be filled in EDNetEventFileGet:req_id in order to distinguish one from multiple requests.

The server will divide a file into multiple packets to send them to the client. So EDNET_EVENT_FILE_GET event will be signaled several times for a file.

client :
a EDNetClient

req_id :
request id

filename :
file name

Returns :
0 on success, -1 if a error occurred.

ednet_client_cancel_file_transfer ()

```
int           ednet_client_cancel_file_transfer       (EDNetClient          *client,
                                                       unsigned int          req_id);
```

Requests the server to cancel the requested download.

client :
a EDNetClient

req_id :
request id

Returns :
0 on success, -1 if a error occurred.

Network Servers

Network Servers – process network client connections

Synopsis

```
typedef EDNetServer;
struct  EDNetServerFuncs;
struct  EDNetServerConfig;

EDNetServer *ednet_server_new           (void);
EDNetServer *ednet_server_ref           (EDNetServer       *server);
void         ednet_server_unref         (EDNetServer       *server);
int          ednet_server_start         (EDNetServer       *server,
                                         EDNetServerConfig *config);
void         ednet_server_stop          (EDNetServer       *server);
void         ednet_server_fill_default  (EDNetServerConfig *config);
void         ednet_server_set_funcs     (EDNetServer       *server,
                                         EDNetServerFuncs  *funcs);
void         ednet_server_disconnect    (EDNetServer       *server,
                                         int                sid);
void         ednet_server_set_property  (EDNetServer       *server,
                                         int                channel,
                                         EDNetPropertyID    prop_id,
                                         int                prop_value,
                                         const char        *prop_str);
void         ednet_server_set_audio     (EDNetServer       *server,
                                         int                channel,
                                         int                has_input,
                                         int                has_output);
int          ednet_server_get_session   (EDNetServer       *server,
                                         int                sid,
                                         EDNetSession      *session);
int          ednet_server_get_sessions  (EDNetServer       *server,
                                         EDNetSession      *sessions,
                                         int               *total);
int          ednet_server_get_peer      (EDNetServer       *server,
                                         int                sid,
                                         int               *channel,
                                         EDNetPropertySet  *props);
void         ednet_server_phone         (EDNetServer       *server,
                                         int                sid,
                                         EDNetPhoneCommand  cmd,
                                         int                src_channel,
```

```
                                    int              dest_channel,
                                    int              flags);
void         ednet_server_notify_device  (EDNetServer      *server,
                                    EDNetDeviceID    dev_id,
                                    int              dev_no,
                                    const char       *data);
void         ednet_server_notify_session (EDNetServer      *server,
                                    int              sid,
                                    EDNetDeviceID    dev_id,
                                    int              dev_no,
                                    int              data);
EDNetBuf     *ednet_server_make_packets  (int              channel,
                                    EDNetCodecID     payload,
                                    int              is_keyframe,
                                    uint32_t         timestamp,
                                    const char       *prefix_data,
                                    int              prefix_len,
                                    const char       *frame_data,
                                    int              frame_len);
void         ednet_server_send_packets   (EDNetServer      *server,
                                    EDNetBuf         *packets);
void         ednet_server_send_frame     (EDNetServer      *server,
                                    int              channel,
                                    EDNetCodecID     payload,
                                    int              is_keyframe,
                                    uint32_t         timestamp,
                                    const char       *frame_data,
                                    int              frame_len);
void         ednet_server_set_buffering  (EDNetServer      *server,
                                    int              seconds);
void         ednet_server_control_stream (EDNetServer      *server,
                                    int              sid,
                                    int              channel,
                                    int              request);
```

Description

Details

EDNetServer

```
typedef struct _EDNetServer EDNetServer;
```

A server to process network connections.

EDNetServerFuncs

```
typedef struct
{
  void  (*receive_frame) (EDNetServer *server, int sid, EDNetFrame *frame);
  void  (*control_stream) (EDNetServer *server, int sid, int channel, int requests, int start_or_stop);
  int   (*set_property) (EDNetServer *server, int sid, int channel, EDNetPropertyID prop_id, const char *old_value
  void  (*control_device) (EDNetServer *server, int sid, EDNetDeviceID dev_id, int dev_no, EDNetDevicePTZCommand p
  void  (*phone) (EDNetServer *server, int sid, EDNetPhoneCommand phone_cmd, int src_channel, int dest_channel, in
  int   (*check_auth) (EDNetServer *server, int sid, const char *user, const char *enc_password);
  void  (*get_user_role) (EDNetServer *server, int sid, int user_id, char *channel_access, char *channel_streaming
  char * (*config_read) (EDNetServer *server, int sid, const char *key);
  void  (*config_write) (EDNetServer *server, int sid, const char *key, const char *value);
  void  (*session_changed) (EDNetServer *server, int sid, int action);
} EDNetServerFuncs;
```

void (*receive_frame) (EDNetServer *server, int sid, EDNetFrame *frame) :

Process frame packets received from clients.

server :

a EDNetServer

sid :

client session ID

frame :

a new frame received

void (*control_stream) (EDNetServer *server, int sid, int channel, int requests, int start_or_stop) :

Control live streaming.

server :

a EDNetServer

sid :

client session ID

channel :

channel number to control live streaming

requests :

the number of sessions for the channel

start_or_stop :

TRUE / FALSE

`int (*set_property) (EDNetServer *server, int sid, int channel, EDNetPropertyID prop_id, const char *old_value, const char *new_value)` :

Set the property of the channel.

`server` :

a EDNetServer

`sid` :

client session ID

`channel` :

channel number

`prop_id` :

property ID

`old_value` :

current property string

`new_value` :

new property string

Returns :

0 if it's allowed, or −1.

You don't need call ednet_server_set_property().

`void (*control_device) (EDNetServer *server, int sid, EDNetDeviceID dev_id, int dev_no, EDNetDevicePTZCommand ptz_cmd)` :

Control devices.

`server` :

a EDNetServer

`sid` :

client session ID

`dev_id` :

device ID

`dev_no` :

device number

`ptz_cmd` :

PTZ control data, see EDNetDevicePTZCommand

`void (*phone) (EDNetServer *server, int sid, EDNetPhoneCommand phone_cmd, int src_channel, int dest_channel, int flags)` :

Process phone signals.

`server` :

a EDNetServer

`sid` :
client session ID

`phone_cmd` :
phone command

`src_channel` :
source channel

`dest_channel` :
destination channel

`int (*check_auth) (EDNetServer *server, int sid, const char *user, const char *enc_password)` :
Check authentication and return the result.

`server` :
a EDNetServer

`sid` :
client session ID

`user` :
user name

`enc_password` :
MD5 encrypted password

Returns :
user ID, or −1 if no user, −2 if invalid password, −3 if no access right.

`void (*get_user_role) (EDNetServer *server, int sid, int user_id, char *channel_access, char *channel_streaming, char *channel_device, uint32_t *system_role)` :
Get user role information.

`server` :
a EDNetServer

`sid` :
client session ID

`user_id` :
user ID

`channel_access` :
channel access access to be stored

`channel_streaming` :
channel streaming access to be stored

`channel_device` :
channel device access to be stored

system :
system access to be stored

char * (*config_read) (EDNetServer *server, int sid, const char *key) :
Read the configuration value for the given key.

server :
a EDNetServer

sid :
client session ID

key :
a key string which is separated with slash(/) characters

Returns :
newly allocated value string which must be freed with free() or NULL.

void (*config_write) (EDNetServer *server, int sid, const char *key, const char *value) :
Write the configuration value for the given key.

server :
a EDNetServer

sid :
client session ID

key :
a key string which is separated with slash(/) characters

value :
a value string

void (*session_changed) (EDNetServer *server, int sid, int action) :
Notify session list change.

server :
a EDNetServer

sid :
client session ID

action :
1:added, 0:removed

EDNetServerConfig

```
typedef struct
{
  int  session_max;
```

```
  int  video_total;
  int  audio_total;
  int  listen_port;
  int  product_type;
  int  product_id;
  int  dio_inputs;
  int  dio_outputs;
  char disk_host_name[16];
  int  sndbuf_size;
} EDNetServerConfig;
```

int session_max :

int video_total :

int audio_total :

int listen_port :

int product_type :

int product_id :

int dio_inputs :

int dio_outputs :

char disk_host_name[16] :

int sndbuf_size :

ednet_server_new ()

```
EDNetServer *ednet_server_new           (void);
```

ednet_server_ref ()

```
EDNetServer *ednet_server_ref           (EDNetServer       *server);
```

ednet_server_unref ()

```
void         ednet_server_unref         (EDNetServer       *server);
```

ednet_server_start ()

```
int          ednet_server_start          (EDNetServer       *server,
                                          EDNetServerConfig *config);
```

ednet_server_stop ()

```
void         ednet_server_stop           (EDNetServer       *server);
```

ednet_server_fill_default ()

```
void         ednet_server_fill_default   (EDNetServerConfig *config);
```

ednet_server_set_funcs ()

```
void         ednet_server_set_funcs      (EDNetServer       *server,
                                          EDNetServerFuncs  *funcs);
```

ednet_server_disconnect ()

```
void         ednet_server_disconnect     (EDNetServer       *server,
                                          int                sid);
```

ednet_server_set_property ()

```
void         ednet_server_set_property   (EDNetServer       *server,
                                          int                channel,
                                          EDNetPropertyID    prop_id,
                                          int                prop_value,
                                          const char        *prop_str);
```

ednet_server_set_audio ()

```
void         ednet_server_set_audio      (EDNetServer       *server,
                                          int                channel,
                                          int                has_input,
                                          int                has_output);
```

ednet_server_get_session ()

```
int         ednet_server_get_session    (EDNetServer       *server,
                                         int                sid,
                                         EDNetSession       *session);
```

ednet_server_get_sessions ()

```
int         ednet_server_get_sessions   (EDNetServer       *server,
                                         EDNetSession       *sessions,
                                         int                *total);
```

ednet_server_get_peer ()

```
int         ednet_server_get_peer       (EDNetServer       *server,
                                         int                sid,
                                         int                *channel,
                                         EDNetPropertySet   *props);
```

ednet_server_phone ()

```
void        ednet_server_phone          (EDNetServer       *server,
                                         int                sid,
                                         EDNetPhoneCommand  cmd,
                                         int                src_channel,
                                         int                dest_channel,
                                         int                flags);
```

ednet_server_notify_device ()

```
void        ednet_server_notify_device  (EDNetServer       *server,
                                         EDNetDeviceID      dev_id,
                                         int                dev_no,
                                         const char         *data);
```

ednet_server_notify_session ()

```
void        ednet_server_notify_session (EDNetServer       *server,
                                         int                sid,
                                         EDNetDeviceID      dev_id,
                                         int                dev_no,
                                         int                data);
```

ednet_server_make_packets ()

```
EDNetBuf    *ednet_server_make_packets   (int               channel,
                                         EDNetCodecID       payload,
                                         int                is_keyframe,
                                         uint32_t           timestamp,
                                         const char        *prefix_data,
                                         int                prefix_len,
                                         const char        *frame_data,
                                         int                frame_len);
```

ednet_server_send_packets ()

```
void        ednet_server_send_packets   (EDNetServer       *server,
                                         EDNetBuf          *packets);
```

ednet_server_send_frame ()

```
void        ednet_server_send_frame     (EDNetServer       *server,
                                         int                channel,
                                         EDNetCodecID       payload,
                                         int                is_keyframe,
                                         uint32_t           timestamp,
                                         const char        *frame_data,
                                         int                frame_len);
```

ednet_server_set_buffering ()

```
void        ednet_server_set_buffering  (EDNetServer       *server,
                                         int                seconds);
```

ednet_server_control_stream ()

```
void        ednet_server_control_stream (EDNetServer       *server,
                                         int                sid,
                                         int                channel,
                                         int                request);
```

Network Events

Network Events - information from network clients

Synopsis

```
enum    EDNetEventType;
struct  EDNetEvent;
struct  EDNetEventConnect;
struct  EDNetEventConnectReply;
struct  EDNetEventShutdownReply;
struct  EDNetEventSessionList;
struct  EDNetEventUserRole;
struct  EDNetEventChannelList;
struct  EDNetEventChannelProperty;
struct  EDNetEventChannelPropertyRequest;
struct  EDNetEventStreamRequest;
struct  EDNetEventStreamRequestReply;
struct  EDNetEventDeviceInfo;
struct  EDNetEventDeviceControl;
struct  EDNetEventDeviceEvent;
struct  EDNetEventPhone;
struct  EDNetEventDirRequest;
struct  EDNetEventDirReply;
struct  EDNetEventConfig;
struct  EDNetEventMpegFileOpen;
struct  EDNetEventMpegFileOpenReply;
struct  EDNetEventMpegFileRead;
struct  EDNetEventMpegFileReadReply;
struct  EDNetEventMpegFileSeek;
struct  EDNetEventMpegFileSeekReply;
struct  EDNetEventMpegFileClose;
struct  EDNetEventMpegFileCloseReply;
struct  EDNetEventFileRequest;
struct  EDNetEventFileGet;
struct  EDNetEventFileRequestCancel;
struct  EDNetEventFrame;
struct  EDNetEventSignal;
struct  EDNetEventWantedFrame;
```

```
void ednet_event_ref   (EDNetEvent *event);
void ednet_event_unref (EDNetEvent *event);
```

Description

Network events are signaled by network client objects in order to notify information about the remote server.

Some events are used in library internal, and they are not notified to applications.

Details

EDNetEventType

```
typedef enum
{
  EDNET_EVENT_CONNECT                 = 0,
  EDNET_EVENT_CONNECT_REPLY           = 1,
  EDNET_EVENT_SHUTDOWN                = 2,
  EDNET_EVENT_SHUTDOWN_REPLY          = 3,
  EDNET_EVENT_SESSION_LIST            = 4,
  EDNET_EVENT_USER_ROLE               = 5,
  EDNET_EVENT_CHANNEL_LIST            = 6,
  EDNET_EVENT_CHANNEL_LIST_REQUEST    = 7,
  EDNET_EVENT_CHANNEL_PROPERTY        = 8,
  EDNET_EVENT_CHANNEL_PROPERTY_SET    = 9,
  EDNET_EVENT_CHANNEL_PROPERTY_REQUEST = 10,
  EDNET_EVENT_STREAM_REQUEST          = 11,
  EDNET_EVENT_STREAM_REQUEST_REPLY    = 12,
  EDNET_EVENT_DEVICE_INFO             = 13,
  EDNET_EVENT_DEVICE_CONTROL          = 14,
  EDNET_EVENT_DEVICE_EVENT            = 15,
  EDNET_EVENT_PHONE                   = 16,
  EDNET_EVENT_DIR_REQUEST             = 17,
  EDNET_EVENT_DIR_REPLY               = 18,
  EDNET_EVENT_CONFIG_GET              = 19,
  EDNET_EVENT_CONFIG_SET              = 20,
  EDNET_EVENT_CONFIG_REPLY            = 21,
  EDNET_EVENT_MPEG_FILE_OPEN          = 22,
  EDNET_EVENT_MPEG_FILE_OPEN_REPLY    = 23,
  EDNET_EVENT_MPEG_FILE_READ          = 24,
  EDNET_EVENT_MPEG_FILE_READ_VIDEO    = 25,
  EDNET_EVENT_MPEG_FILE_READ_REPLY    = 26,
  EDNET_EVENT_MPEG_FILE_SEEK          = 27,
```

```
  EDNET_EVENT_MPEG_FILE_SEEK_REPLY     = 28,
  EDNET_EVENT_MPEG_FILE_CLOSE          = 29,
  EDNET_EVENT_MPEG_FILE_CLOSE_REPLY    = 30,
  EDNET_EVENT_FILE_REQUEST             = 31,
  EDNET_EVENT_FILE_GET                 = 32,
  EDNET_EVENT_FILE_REQUEST_CANCEL      = 33,
  EDNET_EVENT_FRAME_TRANSMIT           = 34,
  EDNET_EVENT_SIGNAL                   = 35,
  EDNET_EVENT_WANTED_FRAME             = 36,
  EDNET_EVENT_QUIT                     = 37,
  EDNET_EVENT_MAX
} EDNetEventType;
```

Event types.

EDNET_EVENT_CONNECT :
EDNetEventConnect

EDNET_EVENT_CONNECT_REPLY :
EDNetEventConnectReply

EDNET_EVENT_SHUTDOWN :
EDNetEvent

EDNET_EVENT_SHUTDOWN_REPLY :
EDNetEventShutdownReply

EDNET_EVENT_SESSION_LIST :
EDNetEventSessionList

EDNET_EVENT_USER_ROLE :
EDNetEventUserRole

EDNET_EVENT_CHANNEL_LIST :
EDNetEventChannelList

EDNET_EVENT_CHANNEL_LIST_REQUEST :
EDNetEvent

EDNET_EVENT_CHANNEL_PROPERTY :
EDNetEventChannelProperty

EDNET_EVENT_CHANNEL_PROPERTY_SET :
EDNetEventChannelProperty

EDNET_EVENT_CHANNEL_PROPERTY_REQUEST :
EDNetEventChannelPropertyRequest

EDNET_EVENT_STREAM_REQUEST :
EDNetEventStreamRequest

EDNET_EVENT_STREAM_REQUEST_REPLY :
EDNetEventStreamRequestReply

EDNET_EVENT_DEVICE_INFO :
EDNetEventDeviceInfo

EDNET_EVENT_DEVICE_CONTROL :
EDNetEventDeviceControl

EDNET_EVENT_DEVICE_EVENT :
EDNetEventDeviceEvent

EDNET_EVENT_PHONE :
EDNetEventPhone

EDNET_EVENT_DIR_REQUEST :
EDNetEventDirRequest

EDNET_EVENT_DIR_REPLY :
EDNetEventDirReply

EDNET_EVENT_CONFIG_GET :
EDNetEventConfig

EDNET_EVENT_CONFIG_SET :
EDNetEventConfig

EDNET_EVENT_CONFIG_REPLY :
EDNetEventConfig

EDNET_EVENT_MPEG_FILE_OPEN :
EDNetEventMpegFileOpen

EDNET_EVENT_MPEG_FILE_OPEN_REPLY :
EDNetEventMpegFileOpenReply

EDNET_EVENT_MPEG_FILE_READ :
EDNetEventMpegFileRead

EDNET_EVENT_MPEG_FILE_READ_VIDEO :
EDNetEventMpegFileRead

EDNET_EVENT_MPEG_FILE_READ_REPLY :
EDNetEventMpegFileReadReply

EDNET_EVENT_MPEG_FILE_SEEK :
EDNetEventMpegFileSeek

EDNET_EVENT_MPEG_FILE_SEEK_REPLY :
EDNetEventMpegFileSeekReply

EDNET_EVENT_MPEG_FILE_CLOSE :
EDNetEventMpegFileClose

EDNET_EVENT_MPEG_FILE_CLOSE_REPLY :
EDNetEventMpegFileCloseReply

EDNET_EVENT_FILE_REQUEST :
EDNetEventFileRequest

EDNET_EVENT_FILE_GET :
EDNetEventFileGet

EDNET_EVENT_FILE_REQUEST_CANCEL :
EDNetEventFileRequestCancel

EDNET_EVENT_FRAME_TRANSMIT :
EDNetEventFrame

EDNET_EVENT_SIGNAL :
EDNetEventSignal

EDNET_EVENT_WANTED_FRAME :
EDNetEventWantedFrame

EDNET_EVENT_QUIT :
EDNetEvent

EDNET_EVENT_MAX :

EDNetEvent

```
typedef struct
{
  const char    *obj_name;
  void         (*finalize) (void *object);
  int            _ref;
  EDNetEventType type;
} EDNetEvent;
```

The common structure for network events.

const char *obj_name :

void (*finalize) (void *object) :

int _ref :

EDNetEventType type :
event type

EDNetEventConnect

```
typedef struct
{
  const char    *obj_name;
  void         (*finalize) (void *object);
  int            _ref;
  EDNetEventType type;
  char          *host;
  int            port;
  char          *user;
  char          *password;
} EDNetEventConnect;
```

const char *obj_name :

void (*finalize) (void *object) :

int _ref :

EDNetEventType type :
event type

char *host :

int port :

char *user :

char *password :

EDNetEventConnectReply

```
typedef struct
{
  const char            *obj_name;
  void                 (*finalize) (void *object);
  int                    _ref;
  EDNetEventType         type;
  EDNetConnectReplyError error;
} EDNetEventConnectReply;
```

Describes the result of trying to connect to the remote server.

const char *obj_name :

void (*finalize) (void *object) :

int _ref :

EDNetEventType type :
event type

EDNetConnectReplyError error :
error status

EDNetEventShutdownReply

```
typedef struct
{
  const char            *obj_name;
  void                 (*finalize) (void *object);
  int                    _ref;
  EDNetEventType         type;
  EDNetConnectReplyError error;
} EDNetEventShutdownReply;
```

Describes the result of trying to disconnect from the remote server.

const char *obj_name :

void (*finalize) (void *object) :

int _ref :

EDNetEventType type :
event type

EDNetConnectReplyError error :
error status

EDNetEventSessionList

```
typedef struct
{
  const char    *obj_name;
  void         (*finalize) (void *object);
  int            _ref;
  EDNetEventType type;
  int            total;
} EDNetEventSessionList;
```

Describes the number of client sessions in the remote server.

const char *obj_name :

void (*finalize) (void *object) :

int _ref :

EDNetEventType type :
event type

int total :
the number of sessions

EDNetEventUserRole

```
typedef struct
{
  const char    *obj_name;
  void         (*finalize) (void *object);
  int            _ref;
  EDNetEventType type;
  int            uid;
  char           channel_access[ ((256-128)/8)];
  char           channel_streaming[ ((256-128)/8)];
  char           channel_device[ ((256-128)/8)];
  int            system;
} EDNetEventUserRole;
```

Describes connected account information.

const char *obj_name :

void (*finalize) (void *object) :

int _ref :

EDNetEventType type :
event type

int uid :
user id

char channel_access[((256-128)/8)] :
channel access permission (bitwised)

char channel_streaming[((256-128)/8)] :
channel streaming permission (bitwised)

char channel_device[((256-128)/8)] :
channel device contol permission (bitwised)

int system :
system permission, see EDNetRoleSystem.

EDNetEventChannelList

```
typedef struct
{
  const char    *obj_name;
  void         (*finalize) (void *object);
  int            _ref;
  EDNetEventType type;
  int            total;
  int           *channels;
} EDNetEventChannelList;
```

Describes the list of channels in the remote server.

const char *obj_name :

void (*finalize) (void *object) :

int _ref :

EDNetEventType type :
event type

int total :
the number of channels

int *channels :
the array of channel numbers

EDNetEventChannelProperty

```
typedef struct
{
  const char                          *obj_name;
  void                               (*finalize) (void *object);
  int                                  _ref;
  EDNetEventType                       type;
  int                                  channel;
  EDNetPropertyID                      id;
  EDNetPropertyValue                   vtype;
  int                                  val;
  char                                *str;
  union _EDNetEventChannelProperty::@2 value;
} EDNetEventChannelProperty;
```

Describes the property of the channel.

const char *obj_name :

void (*finalize) (void *object) :

int _ref :

EDNetEventType type :
event type

int channel :
channel number

EDNetPropertyID id :
property id

EDNetPropertyValue vtype :
property type

int val :

char *str :

union _EDNetEventChannelProperty::@2 value :
property value

EDNetEventChannelPropertyRequest

```
typedef struct
{
  const char    *obj_name;
  void         (*finalize) (void *object);
  int            _ref;
  EDNetEventType type;
  int            channel;
} EDNetEventChannelPropertyRequest;
```

Describes the request for channel properties.

const char *obj_name :

void (*finalize) (void *object) :

int _ref :

EDNetEventType type :
event type

int channel :
channel number

EDNetEventStreamRequest

```
typedef struct
{
  const char    *obj_name;
  void         (*finalize) (void *object);
  int            _ref;
  EDNetEventType type;
  int            channel;
  int            request;
} EDNetEventStreamRequest;
```

Describes the request for live streaming.

const char *obj_name :

void (*finalize) (void *object) :

int _ref :

EDNetEventType type :
event type

int channel :
channel number

int request :
TRUE:start, FALSE:cancel

EDNetEventStreamRequestReply

```
typedef struct
{
  const char    *obj_name;
  void         (*finalize) (void *object);
  int            _ref;
  EDNetEventType type;
  int            channel;
  int            reply;
} EDNetEventStreamRequestReply;
```

Describes the response of live streaming requests.

const char *obj_name :

void (*finalize) (void *object) :

int _ref :

EDNetEventType type :
event type

int channel :
channel number

int reply :
TRUE:started, FALSE:stopped

EDNetEventDeviceInfo

```
typedef struct
{
  const char            *obj_name;
  void                 (*finalize) (void *object);
  int                    _ref;
  EDNetEventType         type;
  EDNetDeviceID          dev_id;
  int                    dev_no;
  EDNetDeviceInfoCommand cmd;
  int                    cmd_data;
} EDNetEventDeviceInfo;
```

Describes event device information.

const char *obj_name :

void (*finalize) (void *object) :

int _ref :

EDNetEventType type :
event type

EDNetDeviceID dev_id :
device type

int dev_no :
device number

EDNetDeviceInfoCommand cmd :
a command

int cmd_data :
data

EDNetEventDeviceControl

```
typedef struct
{
  const char           *obj_name;
  void                (*finalize) (void *object);
  int                   _ref;
  EDNetEventType        type;
  EDNetDeviceID         dev_id;
  int                   dev_no;
  EDNetDevicePTZCommand ptz;
} EDNetEventDeviceControl;
```

Describes event device control event.

const char *obj_name :

void (*finalize) (void *object) :

int _ref :

EDNetEventType type :
event type

EDNetDeviceID dev_id :
device type

int dev_no :
device number

EDNetDevicePTZCommand ptz :
control command

EDNetEventDeviceEvent

```
typedef struct
{
  const char    *obj_name;
  void         (*finalize) (void *object);
  int            _ref;
  EDNetEventType type;
  EDNetDeviceID  dev_id;
  int            dev_no;
  unsigned int   timestamp;
  char           data[ ((256-128)/8)];
} EDNetEventDeviceEvent;
```

Generated when some events occurred.

const char *obj_name :

void (*finalize) (void *object) :

int _ref :

EDNetEventType type :
event type

EDNetDeviceID dev_id :
device type

int dev_no :
device number

unsigned int timestamp :
timestamp

char data[((256-128)/8)] :
data

EDNetEventPhone

```
typedef struct
{
  const char       *obj_name;
  void            (*finalize) (void *object);
  int               _ref;
  EDNetEventType    type;
  EDNetPhoneCommand cmd;
  int               src_channel;
  int               dest_channel;
  int               flags;
} EDNetEventPhone;
```

Describes the status of bidirectional audio communication.

const char *obj_name :

void (*finalize) (void *object) :

int _ref :

EDNetEventType type :
event type

EDNetPhoneCommand cmd :
phone command

int src_channel :
local audio channel

int dest_channel :
remote audio channel

int flags :
flags

EDNetEventDirRequest

```
typedef struct
{
  const char    *obj_name;
  void         (*finalize) (void *object);
  int            _ref;
  EDNetEventType type;
  EDNetDirType   req_type;
  unsigned int   req_id;
  char          *path;
} EDNetEventDirRequest;
```

Describes the request for directory listing.

const char *obj_name :

void (*finalize) (void *object) :

int _ref :

EDNetEventType type :
event type

EDNetDirType req_type :
directory type

unsigned int req_id :
request ID

char *path :

EDNetEventDirReply

```
typedef struct
{
  const char    *obj_name;
  void         (*finalize) (void *object);
  int            _ref;
  EDNetEventType type;
```

```
  EDNetDirType   req_type;
  unsigned int   req_id;
  unsigned int   total;
  unsigned int   count;
  char          *names;
} EDNetEventDirReply;
```

Describes the respone of directory listing requests.

const char *obj_name :

void (*finalize) (void *object) :

int _ref :

EDNetEventType type :
event type

EDNetDirType req_type :
directory type

unsigned int req_id :
request ID

unsigned int total :
the number of entries

unsigned int count :
the number of entries in this event

char **names :
the array of entry names

EDNetEventConfig

```
typedef struct
{
  const char    *obj_name;
  void         (*finalize) (void *object);
  int            _ref;
  EDNetEventType type;
  char          *section;
  char          *key;
  char          *value;
} EDNetEventConfig;
```

Describes the configuration request or response.

const char *obj_name :

void (*finalize) (void *object) :

int _ref :

EDNetEventType type :
event type

char *section :
section name

char *key :
key name

char *value :
value

EDNetEventMpegFileOpen

```
typedef struct
{
  const char    *obj_name;
  void         (*finalize) (void *object);
  int           _ref;
  EDNetEventType type;
  unsigned int   req_id;
  unsigned int   file_id;
  char          *name;
} EDNetEventMpegFileOpen;
```

Describes the request for opening the recorded file.

const char *obj_name :

void (*finalize) (void *object) :

int _ref :

EDNetEventType type :
event type

unsigned int req_id :
request id

unsigned int file_id :
file id

char *name :
file name (full path)

EDNetEventMpegFileOpenReply

```
typedef struct
{
  const char            *obj_name;
  void                 (*finalize) (void *object);
  int                    _ref;
  EDNetEventType         type;
  unsigned int           req_id;
  unsigned int           file_id;
  EDNetMpegFileOpenError error;
  unsigned int           vtotal;
  unsigned int           width;
  unsigned int           height;
  unsigned int           frame_interval;
  unsigned int           channels;
  unsigned int           sample_rate;
  unsigned short         video_codec_type;
  unsigned short         audio_codec_type;
} EDNetEventMpegFileOpenReply;
```

Describes the response of the recorded file opening request.

const char *obj_name :

void (*finalize) (void *object) :

int _ref :

EDNetEventType type :
event type

unsigned int req_id :
request id

unsigned int file_id :
file id

EDNetMpegFileOpenError error :
error status, 0 on success

unsigned int vtotal :
the number of video frames

unsigned int width :
the width of video

unsigned int height :
the height of video

unsigned int frame_interval :
frame interval

unsigned int channels :
the number of audio channels (1:mono, 2:stereo)

unsigned int sample_rate :
audio sample rate

unsigned short video_codec_type :
video codec

unsigned short audio_codec_type :
audio codec

EDNetEventMpegFileRead

```
typedef struct
{
  const char           *obj_name;
  void                (*finalize) (void *object);
  int                   _ref;
  EDNetEventType        type;
  unsigned int          req_id;
  unsigned int          file_id;
  EDNetMpegFileReadType read_type;
} EDNetEventMpegFileRead;
```

Describes the request for reading recorded mpeg file.

const char *obj_name :

void (*finalize) (void *object) :

int _ref :

EDNetEventType type :
event type

unsigned int req_id :
request id

unsigned int file_id :
file id

EDNetMpegFileReadType read_type :
reading method

EDNetEventMpegFileReadReply

```
typedef struct
{
  const char             *obj_name;
  void                  (*finalize) (void *object);
  int                     _ref;
  EDNetEventType          type;
  unsigned int            req_id;
  EDNetMpegFileReadFormat format;
  EDNetMpegFileReadFlag   flag;
  unsigned int            index;
  unsigned int            total;
  unsigned int            offset;
  unsigned int            frame_data_len;
  char                   *frame_data;
} EDNetEventMpegFileReadReply;
```

Describes the response of reading mpeg file requests.

const char *obj_name :

void (*finalize) (void *object) :

int _ref :

EDNetEventType type :
event type

unsigned int req_id :
request id

EDNetMpegFileReadFormat format :
frame format

EDNetMpegFileReadFlag flag :
status flags

unsigned int index :
frame index

unsigned int total :
the length of the frame

unsigned int offset :
the offset of data in the frame

unsigned int frame_data_len :
the length of data

char *frame_data :
frame data

EDNetEventMpegFileSeek

```
typedef struct
{
  const char           *obj_name;
  void                (*finalize) (void *object);
  int                   _ref;
  EDNetEventType        type;
  unsigned int          req_id;
  unsigned int          file_id;
  unsigned int          index;
  EDNetMpegFileSeekFlag flag;
} EDNetEventMpegFileSeek;
```

Describes the request for seeking recorded mpeg file.

const char *obj_name :

void (*finalize) (void *object) :

int _ref :

EDNetEventType type :
event type

unsigned int req_id :
request id

unsigned int file_id :
file id

unsigned int index :
frame index

EDNetMpegFileSeekFlag flag :
seek method

EDNetEventMpegFileSeekReply

```
typedef struct
{
  const char            *obj_name;
  void                 (*finalize) (void *object);
  int                    _ref;
  EDNetEventType         type;
  unsigned int           req_id;
```

```
  EDNetMpegFileSeekFlag  flag;
  EDNetMpegFileSeekError error;
  unsigned int           index;
} EDNetEventMpegFileSeekReply;
```

Describes the response of seeking mpeg file requests.

const char *obj_name :

void (*finalize) (void *object) :

int _ref :

EDNetEventType type :
event type

unsigned int req_id :
request id

EDNetMpegFileSeekFlag flag :
seek method

EDNetMpegFileSeekError error :
error status

unsigned int index :

EDNetEventMpegFileClose

```
typedef struct
{
  const char    *obj_name;
  void         (*finalize) (void *object);
  int            _ref;
  EDNetEventType type;
  unsigned int   req_id;
  unsigned int   file_id;
} EDNetEventMpegFileClose;
```

Describes the request for closing recorded mpeg file.

const char *obj_name :

void (*finalize) (void *object) :

int _ref :

EDNetEventType type :
event type

unsigned int req_id :
request id

unsigned int file_id :
file id

EDNetEventMpegFileCloseReply

```
typedef struct
{
  const char             *obj_name;
  void                  (*finalize) (void *object);
  int                     _ref;
  EDNetEventType          type;
  unsigned int            req_id;
  EDNetMpegFileCloseError error;
} EDNetEventMpegFileCloseReply;
```

Describes the response of closing mpeg file requests.

const char *obj_name :

void (*finalize) (void *object) :

int _ref :

EDNetEventType type :
event type

unsigned int req_id :
request id

EDNetMpegFileCloseError error :
status

EDNetEventFileRequest

```
typedef struct
{
  const char    *obj_name;
  void         (*finalize) (void *object);
  int            _ref;
  EDNetEventType type;
  unsigned int   req_id;
  char          *filename;
} EDNetEventFileRequest;
```

Describes the request for download files.

const char *obj_name :

void (*finalize) (void *object) :

int _ref :

EDNetEventType type :
event type

unsigned int req_id :
request id

char *filename :
file name (full path)

EDNetEventFileGet

```
typedef struct
{
  const char    *obj_name;
  void         (*finalize) (void *object);
  int            _ref;
  EDNetEventType type;
  unsigned int   req_id;
  unsigned int   offset;
  unsigned int   size;
  unsigned int   total;
  unsigned int   len;
  char          *data;
} EDNetEventFileGet;
```

Describes the response of download files.

const char *obj_name :

void (*finalize) (void *object) :

int _ref :

EDNetEventType type :
event type

unsigned int req_id :
request id

unsigned int offset :
the offset of data in the file

unsigned int size :

unsigned int total :
the length of the file

unsigned int len :
the length of data

char *data :
file data

EDNetEventFileRequestCancel

```
typedef struct
{
  const char    *obj_name;
  void         (*finalize) (void *object);
  int            _ref;
  EDNetEventType type;
  unsigned int   req_id;
} EDNetEventFileRequestCancel;
```

Describes the request for cancel download.

const char *obj_name :

void (*finalize) (void *object) :

int _ref :

EDNetEventType type :
event type

unsigned int req_id :
request id

EDNetEventFrame

```
typedef struct
{
  const char    *obj_name;
  void         (*finalize) (void *object);
  int            _ref;
  EDNetEventType type;
  EDNetFrame    *frame;
} EDNetEventFrame;
```

Describes the request for transferring frames.

const char *obj_name :

void (*finalize) (void *object) :

int _ref :

EDNetEventType type :
event type

EDNetFrame *frame :
an audio frame

EDNetEventSignal

```
typedef struct
{
  const char       *obj_name;
  void            (*finalize) (void *object);
  int               _ref;
  EDNetEventType    type;
  int               connect;
  EDNetLinkHookType signal;
  EDNetFunc         func;
  void             *data;
} EDNetEventSignal;
```

Describes the request for connecting hook signals.

const char *obj_name :

void (*finalize) (void *object) :

int _ref :

EDNetEventType type :
event type

int connect :
should be connected or disconnected

EDNetLinkHookType signal :
hook type

EDNetFunc func :
hook callback function

void *data :
user data

EDNetEventWantedFrame

```
typedef struct
{
  const char      *obj_name;
  void           (*finalize) (void *object);
  int              _ref;
  EDNetEventType   type;
  int              add;
  int              channel;
  EDNetFrameFormat fmt;
  int              width;
  int              height;
} EDNetEventWantedFrame;
```

Describes the request for adding or removing wanted frame formats.

const char *obj_name :

void (*finalize) (void *object) :

int _ref :

EDNetEventType type :
event type

int add :
should be added or removed

int channel :
channel number

EDNetFrameFormat fmt :
the format of frames

int width :
the width of video frames

int height :
the height of video frames

ednet_event_ref ()

```
void ednet_event_ref   (EDNetEvent *event);
```

Add a reference to the event object.

event :
a event object

ednet_event_unref ()

```
void ednet_event_unref (EDNetEvent *event);
```

Removes a reference from the event object, deallocating the event if no references remain.

event :
a event object

Display Operations

Display Operations – draw video frame images

Synopsis

```
#define ednet_display_destroy

typedef EDNetDisplay;
struct  EDNetRect;

EDNetDisplay    *ednet_display_new                (EDNetWindow   window);
EDNetDisplay    *ednet_display_ref                (EDNetDisplay *d);
void             ednet_display_unref              (EDNetDisplay *d);
void             ednet_display_put_video          (EDNetDisplay *d,
                                                   EDNetFrame   *frame);
void             ednet_display_draw               (EDNetDisplay *d);
void             ednet_display_set_visible        (EDNetDisplay *d,
                                                   int           visible);
void             ednet_display_set_area           (EDNetDisplay *d,
                                                   EDNetRect    *area);
void             ednet_display_set_src_area       (EDNetDisplay *d,
                                                   EDNetRect    *area);
void             ednet_display_set_dest_area      (EDNetDisplay *d,
                                                   EDNetRect    *area);
void             ednet_display_set_border_thickness (EDNetDisplay *d,
                                                   int           thickness);
void             ednet_display_set_border_color   (EDNetDisplay *d,
                                                   uint32_t      color);
int              ednet_display_get_screen_depth   (EDNetDisplay *d);
EDNetFrameFormat ednet_display_get_frame_format   (EDNetDisplay *d);
void             ednet_display_get_window_size    (EDNetDisplay *d,
                                                   EDNetRect    *area);
```

```
void            ednet_display_set_bg            (EDNetFrame   *frame);
void            ednet_display_init              (void);
void            ednet_display_exit              (void);
int             ednet_rect_intersect            (EDNetRect    *src1,
                                                 EDNetRect    *src2,
                                                 EDNetRect    *dest);
```

Description

You can draw video frame images received from EDNetClient with these functions.

Details

ednet_display_destroy

```
#define ednet_display_destroy (d)     (d)
```

EDNetDisplay

```
typedef struct _EDNetDisplay EDNetDisplay;
```

A display object to draw video frame images

EDNetRect

```
typedef struct
{
  int x;
  int y;
  int width;
  int height;
} EDNetRect;
```

Defines the position and size of a retangle.

int x :
the x coordinate of the left edge of the rectangle

int y :
the y coordinate of the top of the rectangle

int width :
the width of the rectangle

int height :
the height of the rectangle

ednet_display_new ()

```
EDNetDisplay    *ednet_display_new                (EDNetWindow   window);
```

Creates a new display object within specified window handle as a parent window.

window :
a window handle (HWND in Windows, Drawable in X Window)

Returns :
a new EDNetDisplay, NULL if an error occurred.

ednet_display_ref ()

```
EDNetDisplay    *ednet_display_ref                (EDNetDisplay *d);
```

Adds a reference to the display.

d :
a EDNetDisplay

ednet_display_unref ()

```
void            ednet_display_unref               (EDNetDisplay *d);
```

Removes a reference from the display, deallocating the display if no references remain.

d :
a EDNetDisplay

ednet_display_put_video ()

```
void            ednet_display_put_video           (EDNetDisplay *d,
                                                   EDNetFrame   *frame);
```

Draws a video frame on the screen.

d :
a EDNetDisplay

frame :
a video frame, NULL if you want to draw background image.

ednet_display_draw ()

```
void            ednet_display_draw              (EDNetDisplay *d);
```

Redraws the last video frame on the screen.

d :
a EDNetDisplay

ednet_display_set_visible ()

```
void            ednet_display_set_visible       (EDNetDisplay *d,
                                                 int           visible);
```

Determines whether video images should be visible or invisible.

d :
a EDNetDisplay

visible :
if TRUE, video images should be visible

ednet_display_set_area ()

```
void            ednet_display_set_area          (EDNetDisplay *d,
                                                 EDNetRect    *area);
```

Sets the position and size of the display window in the parent window.

d :
a EDNetDisplay

area :
a EDNetRect

ednet_display_set_src_area ()

```
void            ednet_display_set_src_area      (EDNetDisplay *d,
                                                 EDNetRect    *area);
```

Sets the position and size of the video image to draw on the window.

d :
a EDNetDisplay

area :
a EDNetRect

ednet_display_set_dest_area ()

```
void            ednet_display_set_dest_area       (EDNetDisplay *d,
                                                   EDNetRect    *area);
```

Sets the position and size of the display window on which it draws video images.

d :
a EDNetDisplay

area :
a EDNetRect

ednet_display_set_border_thickness ()

```
void            ednet_display_set_border_thickness (EDNetDisplay *d,
                                                    int           thickness);
```

Set the thickness of borders within display area.

d :
a EDNetDisplay

thickness :
thickness, 0 if no border needed.

ednet_display_set_border_color ()

```
void            ednet_display_set_border_color    (EDNetDisplay *d,
                                                   uint32_t      color);
```

Set the color of borders within display area.

d :
a EDNetDisplay

color :
a color value in 'rrggbb' format

ednet_display_get_screen_depth ()

```
int             ednet_display_get_screen_depth     (EDNetDisplay *d);
```

Returns the depth of current display screen.

It may be 8, 16, 24 or 32.

d :
a EDNetDisplay

Returns :
the depth of display screen

ednet_display_get_frame_format ()

```
EDNetFrameFormat ednet_display_get_frame_format     (EDNetDisplay *d);
```

Returns the frame format suitable for current display screen.

d :
a EDNetDisplay

Returns :
the EDNetFrameFormat

ednet_display_get_window_size ()

```
void            ednet_display_get_window_size      (EDNetDisplay *d,
                                                    EDNetRect    *area);
```

Retrieves the size of internal display window.

d :
a EDNetDisplay

area :
a EDNetRect to store information

ednet_display_set_bg ()

```
void            ednet_display_set_bg               (EDNetFrame   *frame);
```

Changes the background image with a specified frame.

The background image is used to draw images for signal loss or non-exist cameras.

frame :
a EDNetFrame

ednet_display_init ()

```
void            ednet_display_init              (void);
```

Initializes display module.

ednet_display_exit ()

```
void            ednet_display_exit              (void);
```

Finalizes display module.

ednet_rect_intersect ()

```
int             ednet_rect_intersect            (EDNetRect    *src1,
                                                 EDNetRect    *src2,
                                                 EDNetRect    *dest);
```

Calculates the intersection of two rectangles. It is allowed for 'dest' to be the same as either 'src1' or 'src2'. If the rectangles do not intersect, dest's width and height is set to 0 and its x and y values are undefined. If you are only interested in whether the rectangles intersect, but not in the intersecting area itself, pass NULL for 'dest'.

src1 :
a EDNetRect

src2 :
a EDNetRect

dest :
return location for the intersection of src1 and src2, or NULL

Returns :
TRUE if the rectangles intersect.

Sound Operations

Sound Operations - play and capture audio frames

Synopsis

```
#define EDNET_SOUND_VOLUME_MAX
```

```
#define EDNET_SOUND_SAMPLE_RATE
#define EDNET_SOUND_SAMPLE_BITS
#define EDNET_SOUND_SAMPLE_CHANNELS
#define ednet_sound_destroy

typedef EDNetSound;
typedef EDNetSoundCaptureFunc;

EDNetSound *ednet_sound_new_play         (int                   skip_delay);
EDNetSound *ednet_sound_new_capture      (int                   encode);
void        ednet_sound_ref              (EDNetSound           *s);
void        ednet_sound_unref            (EDNetSound           *s);
void        ednet_sound_play             (EDNetSound           *s,
                                          EDNetBuf             *frame);
void        ednet_sound_clear_play_queue (EDNetSound           *s);
void        ednet_sound_set_volume       (EDNetSound           *s,
                                          int                   volume);
void        ednet_sound_set_mute         (EDNetSound           *s,
                                          int                   mute);
void        ednet_sound_start_capture    (EDNetSound           *s);
void        ednet_sound_stop_capture     (EDNetSound           *s);
void        ednet_sound_set_capture_func (EDNetSound           *s,
                                          EDNetSoundCaptureFunc callback,
                                          void                 *callback_data);
```

Description

You can play audio frames received from EDNetClient with these functions, and capture audio frames in order to send them via EDNetSoundStream.

Details

EDNET_SOUND_VOLUME_MAX

```
#define EDNET_SOUND_VOLUME_MAX      128
```

EDNET_SOUND_SAMPLE_RATE

```
#define EDNET_SOUND_SAMPLE_RATE     8000
```

EDNET_SOUND_SAMPLE_BITS

```
#define EDNET_SOUND_SAMPLE_BITS     16
```

EDNET_SOUND_SAMPLE_CHANNELS

```
#define EDNET_SOUND_SAMPLE_CHANNELS 1
```

ednet_sound_destroy

```
#define ednet_sound_destroy (s)     (s)
```

EDNetSound

```
typedef struct _EDNetSound EDNetSound;
```

A sound object to play or capture audio frames

EDNetSoundCaptureFunc

```
typedef int (*EDNetSoundCaptureFunc) (EDNetFrame *frame, void *user_data);
```

ednet_sound_new_play ()

```
EDNetSound *ednet_sound_new_play        (int                 skip_delay);
```

Creates a new EDNetSound for playing audio frames.

skip_delay :
TRUE if skip delayed frames, used for live streaming.

Returns :
a new EDNetSound, NULL if errors occurred.

ednet_sound_new_capture ()

```
EDNetSound *ednet_sound_new_capture     (int                 encode);
```

Creates a new EDNetSound for capturing audio frames.

encode :
TRUE to get encoded frames

Returns :
a new EDNetSound, NULL if errors occurred.

ednet_sound_ref ()

```
void        ednet_sound_ref             (EDNetSound          *s);
```

Adds a reference to the sound object.

s :
a EDNetSound

ednet_sound_unref ()

```
void        ednet_sound_unref           (EDNetSound          *s);
```

Removes a reference from the sound object, deallocating the sound object if no references remain.

s :
a EDNetSound

ednet_sound_play ()

```
void        ednet_sound_play            (EDNetSound          *s,
                                         EDNetBuf            *frame);
```

Enqueues a frame to the internal play queue for playing.

s :
a EDNetSound

frame :
an audio frame

ednet_sound_clear_play_queue ()

```
void        ednet_sound_clear_play_queue (EDNetSound          *s);
```

Clears the internal play queue.

s :
a EDNetSound

ednet_sound_set_volume ()

```
void        ednet_sound_set_volume      (EDNetSound          *s,
                                         int                  volume);
```

Adjusts the volume of the sound.

s :
a EDNetSound

volume :
between 0 and EDNET_SOUND_VOLUME_MAX

ednet_sound_set_mute ()

```
void        ednet_sound_set_mute        (EDNetSound          *s,
                                         int                  mute);
```

Determines whether it should mute the sound.

s :
a EDNetSound

mute :
TRUE if mute should be on

ednet_sound_start_capture ()

```
void        ednet_sound_start_capture   (EDNetSound          *s);
```

Starts to capture audio frames.

s :
a EDNetSound

ednet_sound_stop_capture ()

```
void        ednet_sound_stop_capture    (EDNetSound          *s);
```

Stops to capture audio frames.

s :
a EDNetSound

ednet_sound_set_capture_func ()

```
void        ednet_sound_set_capture_func (EDNetSound          *s,
                                          EDNetSoundCaptureFunc callback,
                                          void                *callback_data);
```

Sets the callback function which should be called when audio frames are captured from devices.

s :
a EDNetSound

callback :
callback function

callback_data :
user data

Sound Streaming

Sound Streaming - support for bidirectional audio comminucation

Synopsis

```
#define ednet_sound_stream_destroy

typedef EDNetSoundStream;

EDNetSoundStream *ednet_sound_stream_new          (int              channel,
                                                   const char      *name);
void              ednet_sound_stream_ref          (EDNetSoundStream *s);
void              ednet_sound_stream_unref        (EDNetSoundStream *s);
const char       *ednet_sound_stream_get_name     (EDNetSoundStream *s);
void              ednet_sound_stream_set_name     (EDNetSoundStream *s,
                                                   const char      *name);
void              ednet_sound_stream_add_client   (EDNetSoundStream *s,
                                                   EDNetClient     *client);
void              ednet_sound_stream_remove_client (EDNetSoundStream *s,
                                                   EDNetClient     *client);
void              ednet_sound_stream_do_event     (EDNetSoundStream *s,
                                                   const EDNetEvent *e,
                                                   EDNetClient     *client);
```

Description

The EDNetSoundStream acts as a audio streaming server. You don't need to create a EDNetSound for capturing or send reply to the server for requesting audio streaming.

Details

ednet_sound_stream_destroy

```
#define ednet_sound_stream_destroy (s)     (s)
```

EDNetSoundStream

```
typedef struct _EDNetSoundStream EDNetSoundStream;
```

A sound stream object to send audio frames to remote servers.

ednet_sound_stream_new ()

```
EDNetSoundStream *ednet_sound_stream_new           (int               channel,
                                                    const char       *name);
```

Creates a new EDNetSoundStream object.

channel :
audio channel number (0 is default)

name :
audio device name

Returns :
a new EDNetSoundStream, NULL if errors occurred.

ednet_sound_stream_ref ()

```
void              ednet_sound_stream_ref           (EDNetSoundStream *s);
```

Adds a reference to the sound streaming object.

s :
a EDNetSoundStream

ednet_sound_stream_unref ()

```
void            ednet_sound_stream_unref        (EDNetSoundStream *s);
```

Removes a reference from the sound streaming object, deallocating the object if no references remain.

s :
a EDNetSoundStream

ednet_sound_stream_get_name ()

```
const char      *ednet_sound_stream_get_name     (EDNetSoundStream *s);
```

Returns the name of the audio device used for sound streaming.

s :
a EDNetSoundStream

ednet_sound_stream_set_name ()

```
void            ednet_sound_stream_set_name     (EDNetSoundStream *s,
                                                 const char      *name);
```

Sets the name of the audio device used for sound streaming.

s :
a EDNetSoundStream

name :
audio device name

ednet_sound_stream_add_client ()

```
void            ednet_sound_stream_add_client   (EDNetSoundStream *s,
                                                 EDNetClient     *client);
```

Adds a EDNetClient to the sound streaming object, which will transmit captured audio frames to the server connected with the client.

s :
a EDNetSoundStream

client :
a EDNetClient

ednet_sound_stream_remove_client ()

```
void             ednet_sound_stream_remove_client (EDNetSoundStream *s,
                                                   EDNetClient      *client);
```

Removes a EDNetClient from the sound streaming object.

s :
a EDNetSoundStream

client :
a EDNetClient

ednet_sound_stream_do_event ()

```
void             ednet_sound_stream_do_event      (EDNetSoundStream *s,
                                                   const EDNetEvent *e,
                                                   EDNetClient      *client);
```

Hands over network events for the sound stream object to process requests like EDNET_EVENT_CHANNEL_LIST_REQUEST, EDNET_EVENT_CHANNEL_PROPERTY_REQUEST, EDNET_EVENT_STREAM_REQUEST automatically.

s :
a EDNetSoundStream

e :
a EDNetEvent

client :
a EDNetClient

Buffers

Buffers - support for buffer objects

Synopsis

```
struct  EDNetBuf;

EDNetBuf *ednet_buf_new   (unsigned int len);
EDNetBuf *ednet_buf_ref   (EDNetBuf    *buf);
void      ednet_buf_unref (EDNetBuf    *buf);
```

Description

Buffer objects are widely used within EDNet SDK internal. It's a base object of EDNetFrame object, so you can call ednet_buf_*() functions with EDNetFrame objects. But It's better to use ednet_frame_*() functions for EDNetFrame objects.

Details

EDNetBuf

```
typedef struct
{
  const char  *dummy;
  void       (*finalize) (void *object);
  int          _ref;
  unsigned int len;
  char        *data;
} EDNetBuf;
```

A Buffer used to contain large data chunks. It supports basic reference counting mechanism for efficient memory mananament in multi-thread environment.

const char *dummy :
private data. NEVER MODIFY THIS!

void (*finalize) (void *object) :
private data. NEVER MODIFY THIS!

int _ref :
private data. NEVER MODIFY THIS!

unsigned int len :
the length of buffer data.

char *data :
the pointer to buffer data.

ednet_buf_new ()

```
EDNetBuf *ednet_buf_new   (unsigned int len);
```

Creates a new buffer, whose data space will be allocated with specified size.

len :
the length of buffer data

Returns :

a new EDNetBuf, NULL if an error occurred.

ednet_buf_ref ()

```
EDNetBuf *ednet_buf_ref   (EDNetBuf    *buf);
```

Adds a reference to the buffer.

buf :

a EDNetBuf

ednet_buf_unref ()

```
void      ednet_buf_unref (EDNetBuf    *buf);
```

Removes a reference from the buffer, deallocating the buffer if no references remain.

buf :

a EDNetBuf

Frame Buffers

Frame Buffers – support for multimedia frame buffers

Synopsis

```
#define EDNET_FRAME_HEADER_LEN

enum    EDNetCodecID;
typedef EDNetCodec;
struct  EDNetFrame;
struct  EDNetFrameHeader;
enum    EDNetFrameFlag;
enum    EDNetFrameFormat;

EDNetCodec *ednet_codec_new                    (EDNetCodecID id,
                                                int          is_encoder,
                                                int          width,
                                                int          height,
                                                int          fps,
                                                int          bitrate,
                                                int          gop_size);
```

```
EDNetCodec *ednet_codec_ref                    (EDNetCodec  *codec);
void        ednet_codec_unref                  (EDNetCodec  *codec);
int         ednet_codec_decode                 (EDNetCodec  *codec,
                                                char        *dest,
                                                int         *dest_len,
                                                const char  *src,
                                                int          src_len);
int         ednet_codec_encode                 (EDNetCodec  *codec,
                                                char        *dest,
                                                int         *dest_len,
                                                const char  *src,
                                                int          src_len,
                                                int         *is_keyframe);
int         ednet_codec_get_max_audio_frame_size (void);
void        ednet_codec_set_deinterlace        (int          deinterlace);
EDNetFrame *ednet_frame_new                    (unsigned int len,
                                                uint32_t     timestamp,
                                                uint8_t      channel,
                                                uint8_t      ptype,
                                                uint16_t     flags);
EDNetFrame *ednet_frame_ref                    (EDNetFrame  *frame);
void        ednet_frame_unref                  (EDNetFrame  *frame);
```

Description

The EDNetFrame is used to pass video and audio frames received from network servers to applications. It may contains frame data in MPEG-4 / H.264 / ADPCM formats or decoded data in YV12 / RGB32 / PCM formats.

Details

EDNET_FRAME_HEADER_LEN

```
#define EDNET_FRAME_HEADER_LEN (sizeof ())
```

EDNetCodecID

```
typedef enum
{
  EDNET_CODEC_VIDEO = 0x00,
  EDNET_CODEC_MPEG4 = 0x01,
  EDNET_CODEC_MJPEG = 0x02,
  EDNET_CODEC_H264  = 0x04,
```

```
  EDNET_CODEC_AUDIO = 0x80,
  EDNET_CODEC_ADPCM = 0x83,
  EDNET_CODEC_G726  = 0x84,
  EDNET_CODEC_PCMU  = 0x86
} EDNetCodecID;
```

Codec types. (Payload)

EDNET_CODEC_VIDEO :
Start number of video codec

EDNET_CODEC_MPEG4 :
MPEG-4

EDNET_CODEC_MJPEG :
MJPEG

EDNET_CODEC_H264 :
H.264

EDNET_CODEC_AUDIO :
Start number of audio codec

EDNET_CODEC_ADPCM :
16bits/8000Hz ADPCM_IMA_WAV

EDNET_CODEC_G726 :
16bits/8000Hz G.726

EDNET_CODEC_PCMU :
8bits/8000Hz G.711 PCMu Law

EDNetCodec

```
typedef struct _EDNetCodec EDNetCodec;
```

Coder and Decoder object.

EDNetFrame

```
typedef struct
{
  EDNetBuf buf;
  uint32_t timestamp;
  uint8_t  channel;
  uint8_t  ptype;
```

```
  uint16_t flags;
  int      fmt;
  int      width;
  int      height;
} EDNetFrame;
```

A Buffer used for multimedia video / audio frames, which supports basic reference counting mechanism and frame information.

EDNetBuf buf :
a buffer, the base object

uint32_t timestamp :
timestamp

uint8_t channel :
channel number

uint8_t ptype :
payload, see EDNetCodecID

uint16_t flags :
flags, see EDNetFrameFlag

int fmt :
format, see EDNetFrameFormat

int width :
The width of video frame

int height :
The height of video frame

EDNetFrameHeader

```
typedef struct
{
  uint32_t skey;
  uint32_t timestamp;
  uint32_t size;
  uint32_t offset;
  uint8_t  channel;
  uint8_t  ptype;
  uint16_t flags;
} EDNetFrameHeader;
```

uint32_t skey :
Session key

uint32_t timestamp :
Timestamp

uint32_t size :
Frame size

uint32_t offset :
Offset

uint8_t channel :
Channel number

uint8_t ptype :
see EDNetCodecID

uint16_t flags :
see EDNetFrameFlag

EDNetFrameFlag

```
typedef enum
{
  EDNET_FRAME_FLAG_KEY_FRAME = 1 << 0,
  EDNET_FRAME_FLAG_VIDEO     = 1 << 1,
  EDNET_FRAME_FLAG_AUDIO     = 1 << 2
} EDNetFrameFlag;
```

Frame flags

EDNET_FRAME_FLAG_KEY_FRAME :
it's a key frame.

EDNET_FRAME_FLAG_VIDEO :
it contains video data.

EDNET_FRAME_FLAG_AUDIO :
it contains audio data.

EDNetFrameFormat

```
typedef enum
{
  EDNET_FRAME_FORMAT_ENCODED = 0,
  EDNET_FRAME_FORMAT_YV12,
  EDNET_FRAME_FORMAT_RGB32,
  EDNET_FRAME_FORMAT_RGB24,
  EDNET_FRAME_FORMAT_RGB16,
```

```
  EDNET_FRAME_FORMAT_PCM
} EDNetFrameFormat;
```

Frame formats

EDNET_FRAME_FORMAT_ENCODED :
encoded, see EDNetCodecID

EDNET_FRAME_FORMAT_YV12 :
YV12

EDNET_FRAME_FORMAT_RGB32 :
RGB32

EDNET_FRAME_FORMAT_RGB24 :
RGB24

EDNET_FRAME_FORMAT_RGB16 :
RGB16

EDNET_FRAME_FORMAT_PCM :
PCM (16 bits, 8000Hz, Mono)

ednet_codec_new ()

```
EDNetCodec *ednet_codec_new                    (EDNetCodecID id,
                                                int          is_encoder,
                                                int          width,
                                                int          height,
                                                int          fps,
                                                int          bitrate,
                                                int          gop_size);
```

새로운 코덱 객체를 생성합니다.

id :
codec ID

is_encoder :
TRUE or FALSE

width :
the width of video frames

height :
the height of video frames

fps :
frame rates

bitrate :
bit rates

gop_size :
key frame interval

Returns :
On success a newly created codec object, otherwise NULL.

ednet_codec_ref ()

```
EDNetCodec *ednet_codec_ref                    (EDNetCodec  *codec);
```

Increases the reference count on a codec by 1.

ednet_codec_unref ()

```
void        ednet_codec_unref                  (EDNetCodec  *codec);
```

Decreases the reference count on a codec by 1.

ednet_codec_decode ()

```
int         ednet_codec_decode                 (EDNetCodec  *codec,
                                                char        *dest,
                                                int         *dest_len,
                                                const char  *src,
                                                int          src_len);
```

코덱을 이용하여 비디오 영상을 복호화(Decoding) 합니다.

codec :
EDNetCodec 객체

dest :
복호화 한 프레임을 넣을 버퍼

dest_len :
복호화 한 프레임을 넣을 버퍼의 크기

src :
복호화 할 데이터

src_len :
복호화 할 데이터 크기

Returns :
정상적으로 복호화 했을 경우 0, 그렇지 않으면 -1

ednet_codec_encode ()

```
int         ednet_codec_encode                  (EDNetCodec  *codec,
                                                 char        *dest,
                                                 int         *dest_len,
                                                 const char  *src,
                                                 int          src_len,
                                                 int         *is_keyframe);
```

코덱을 이용하여 부호화(Encoding) 합니다.

codec :
EDNetCodec 객체

dest :
부호화 한 데이터를 넣을 버퍼

dest_len :
부호화 한 데이터를 넣을 버퍼의 크기

src :
부호화 할 데이터

src_len :
부호화 할 데이터 크기

is_keyframe :
a pointer to variable to be stored with it's key frame

Returns :
정상적으로 부호화 했을 경우 0, 그렇지 않으면 -1

ednet_codec_get_max_audio_frame_size ()

```
int         ednet_codec_get_max_audio_frame_size (void);
```

오디오 프레임의 최대 크기를 얻어옵니다.

Returns :
AVCODEC_MAX_AUDIO_FRAME_SIZE

ednet_codec_set_deinterlace ()

```
void        ednet_codec_set_deinterlace         (int         deinterlace);
```

디인터레이스 필터를 사용할지 여부를 설정합니다.

deinterlace :
디인터레이스 필터를 사용하면 TRUE

ednet_frame_new ()

```
EDNetFrame *ednet_frame_new                     (unsigned int len,
                                                 uint32_t     timestamp,
                                                 uint8_t      channel,
                                                 uint8_t      ptype,
                                                 uint16_t     flags);
```

Creates a new frame buffer object whose data space will be allocated with specified size, and filled with specified information.

len :
the length of a frame data

timestamp :
timestamp

channel :
channel number (see EDNET_CHANNEL_MAX, EDNET_CHANNEL_AUDIO)

ptype :
payload (see EDNetCodecID)

flags :
flags (see EDNetFrameFlag)

Returns :
a new EDNetFrame, NULL if an error occurred.

ednet_frame_ref ()

```
EDNetFrame *ednet_frame_ref                     (EDNetFrame  *frame);
```

Adds a reference to the frame.

frame :
a EDNetFrame

ednet_frame_unref ()

```
void        ednet_frame_unref                   (EDNetFrame  *frame);
```

Removes a reference from the frame, deallocating the buffer if no references remain.

frame :
a EDNetFrame

Message Logging

Message Logging - support for logging messages

Synopsis

```
enum    EDNetLogLevel;
typedef EDNetLogFunc;

void ednet_log             (EDNetLogLevel level,
                            void         *object,
                            const char   *fmt,
                            ...);
void ednet_log_set_handler (EDNetLogFunc  func,
                            void         *data);
void ednet_log_set_level   (EDNetLogLevel level);
```

Description

These functions provide support for logging error messages or messages used for debugging.

There are several built-in levels of messages, defined in EDNetLogLevel.

Details

EDNetLogLevel

```
typedef enum
{
  EDNET_LOG_DEBUG,
  EDNET_LOG_INFO,
  EDNET_LOG_WARNING,
  EDNET_LOG_ERROR
} EDNetLogLevel;
```

Log message level.

EDNET_LOG_DEBUG :
debug messages (lowest)

EDNET_LOG_INFO :
informative message

EDNET_LOG_WARNING :
non-critical warning messages

EDNET_LOG_ERROR :
critical error messages (highest)

EDNetLogFunc

```
typedef void (*EDNetLogFunc) (const char *buf, void *data);
```

Specifies the type of function which is called when log messages are printed.

ednet_log ()

```
void ednet_log             (EDNetLogLevel level,
                            void         *object,
                            const char   *fmt,
                            ...);
```

Prints log messages with specified level. If there is no handler registered via ednet_log_set_handler(), messages will be printed on standard error terminal output (stderr). Log messages whose level is lower than one specified with ednet_log_set_level() will be ignored.

If log message level is EDNET_LOG_ERROR, program will be stopped by calling abort().

level :
log message level

object :
log message sender, NULL is acceptable.

fmt :
printf-style message format

ednet_log_set_handler ()

```
void ednet_log_set_handler (EDNetLogFunc  func,
                            void         *data);
```

Registers log message handler function which called when log messages are printed by ednet_log(). User data is passed as the second parameter on calling handler function.

func :
log message handler

data :
user data

ednet_log_set_level ()

```
void ednet_log_set_level   (EDNetLogLevel level);
```

Determines the log message level, which log messages whose level is lower than will not be printed.

Default log message level is EDNET_LOG_INFO.

level :
log message level

Chapter#3.#EDNet Plugin (ActiveX) API Reference

Overview

EDNet plugin ActiveX(EDNet OCX) provides a OCX control which can be used to view and control remote DVR / NT servers in Visual Basic or Internet Explorer. It also provides more convenient environment to develop network client applications than EDNet C/C++ API in Windows.

Basic Usage

Applicaions written in Javascript or Visual Basic communicate with EDNet OCX by using Events and Attributes. Applications should use events to not only send commands but also receive notification from the network. The value of attributes can be fetched or changed with events.

EDNet OCX provides just two API functions for applications:

```
sendEvent (event, data);
getAttr (attr, value);
```

`sendEvent()` function is used to send events to EDNet OCX. The first parameter `event` is a integer value and stands for event type. The second parameter `data` is a string and contains event-related additional parameters.

`getAttr()` function is used to retrieve the value of a attribute. The first parameter `attr` is a integer value and stands for attribute type. The second parameter `value` is string variable to be stored value.

If events for the application occur, they are notified by calling following event handler of EDNet OCX. Applications should register callback function for this event handler in order to process events.

```
onEvent (event, data)
```

Like `sendEvent()` function, The first parameter `event` is a integer value and stands for event type. The second parameter `data` is a string and contains event-related additional parameters.

Events

The type of event is integer number. (You can lookup these numbers in `ednetp-const.js` in web client javascript examples) The parameter `data` is a string and event-specific.

EDNETP_EVENT_STATE (0)

Describes the state of connection to the remote server. Data format is `'state'`, where `state` is a integer value (see EDNETP_ATTR_STATE(2))

This event is receive-only.

EDNETP_EVENT_ATTR (1)

Changes the value of a attribute, or notified when it is changed. Data format is `'attr:value'`, where `attr` is a integer value and `value` is a string. (see Attributes)

EDNETP_EVENT_LOGIN (2)

Sets login information. Data format is `host:port:user:password`, where `host` is the remote host address, `port` is a port number, `user` is user name and `password` is password.

For example, `'demo.nvrsw.com:8081:guest:guest'` is valid login information.

This event is send-only.

EDNETP_EVENT_CONNECT (3)

Starts to connect to the remote server with login information.

This event is send-only.

EDNETP_EVENT_SHUTDOWN (4)

Disconnects from the remote server.

This event is send-only.

EDNETP_EVENT_HOST_EVENT (5)

Sends device events, or notified from the remote server. Date format is `'event:device'`, where `event` is a event type, and `device` is a device number.

For example, the motion detection of camera 2 event is `'1:2'`.

The list of event types is following:

EDNETP_DEVICE_GENERAL (0)

EDNETP_DEVICE_PTZ (1)
Pan/Tilt/Zoom for the camera device

EDNETP_DEVICE_MOTION (2)
Motion detection for the camera device

EDNETP_DEVICE_ALARM (3)
Relay devices

EDNETP_DEVICE_SENSOR (4)
Sensor devices

EDNETP_DEVICE_DISK (5)
Disk error

EDNETP_DEVICE_VIDEO_LOSS (6)
Video signal loss for the camera device

EDNETP_DEVICE_RAID (7)
RAID device error

EDNETP_DEVICE_VIDEO_RECOVER (8)
Video signal recover for the camera device

EDNETP_DEVICE_EXTERNAL (0x40)
External events

EDNETP_DEVICE_ANALOG (0x41)
Analog input devices

EDNETP_DEVICE_CUSTOM (0x80)

EDNETP_EVENT_PRESET (6)

Sends preset information for the camera. Data format is `'camera:command:preset-no'`, where `camera` is a camera number, `command` is a preset command type, `preset-no` is a preset number.

For example, to set the preset 2 of camera 3 with the current position, data is `'2:0:2'`.

This event is send-only.

The list of available preset commands is following:

EDNETP_PRESET_COMMAND_SET (0)
Set current camera position to the preset number

EDNETP_PRESET_COMMAND_MOVE (1)
Move camera to the preset number

EDNETP_PRESET_COMMAND_RESET (2)
Reset the preset number

EDNETP_EVENT_PTZ (7)

Controls Pan/Tilt/Zoom devices on the remote server. Data format is `'camera:command'`, where `camera` is a camera number, and `command` is a command type.

For example, zoom in camera 1 is `'0:10'`.

This event is send-only.

The list of available PTZ commands is following:

EDNETP_PTZ_COMMAND_LEFT_UP (0)
Moves the camera left and up

EDNETP_PTZ_COMMAND_UP (1)
Moves the camera up

EDNETP_PTZ_COMMAND_RIGHT_UP (2)
Moves the camera right and up

EDNETP_PTZ_COMMAND_LEFT (3)
Moves the camera left

EDNETP_PTZ_COMMAND_STOP (4)
Stops the camera moving

EDNETP_PTZ_COMMAND_RIGHT (5)
Moves the camera right

EDNETP_PTZ_COMMAND_LEFT_DOWN (6)
Moves the camera left and down

EDNETP_PTZ_COMMAND_DOWN (7)
Moves the camera down

EDNETP_PTZ_COMMAND_RIGHT_DOWN (8)
Moves the camera right and down

EDNETP_PTZ_COMMAND_ZOOM_IN (9)
Zoom in

EDNETP_PTZ_COMMAND_ZOOM_OUT (10)
Zoom out

EDNETP_PTZ_COMMAND_FOCUS_FAR (11)
Sets focus far

EDNETP_PTZ_COMMAND_FOCUS_NEAR (12)
Sets focus near

EDNETP_PTZ_COMMAND_AUTOPAN_ON (13)
Turns on auto-pan

EDNETP_PTZ_COMMAND_AUTOPAN_OFF (14)
Turns off auto-pan

EDNETP_PTZ_COMMAND_LIGHT_ON (15)
Turns on the light

EDNETP_PTZ_COMMAND_LIGHT_OFF (16)
Turns off the light

EDNETP_EVENT_CHANNEL_PROPERTY (8)

Describes the property of the channel. Date format is `'channel-no:property-name:value'`, where `channel` is a channel number, `property-name` is a property number and `value` is the value of the property.

For example, the name of audio channel 128 is `'128:name:audio1'`.

This event is receive-only.

EDNETP_EVENT_CHANNEL_STATE (9)

Describes the state of the channel. Date format is `'channel-no:state'`, where `channel-no` is a channel number, and `state` is a bitwise integer value.

This event is receive-only.

EDNETP_EVENT_BACKGROUND (10)

Not yet supported.

EDNETP_EVENT_PHONE_COMMAND (11)

Sends phone (bidirectional audio communication) commands to the remote server. Data format is `'audio-channel:command-no'`, where `audio-channel` is a audio channel number, and `command-no` is a phone command.

This event is send-only.

The list of available phone commands is following:

EDNETP_PHONE_COMMAND_CALL (0)
Calls the audio channel. When the remote accepts the call, real communication is established.

EDNETP_PHONE_COMMAND_BROADCAST (1)
Makes the audio channel speakable forcibly.

EDNETP_PHONE_COMMAND_ACCEPT (2)
Accepts the call from the remote.

EDNETP_PHONE_COMMAND_IGNORE (3)
Ignores the call from the remote.

EDNETP_PHONE_COMMAND_LISTEN (4)
Starts to listen to the remote audio channel.

EDNETP_PHONE_COMMAND_NOT_LISTEN (5)
Stops to listen to the remote audio channel.

EDNETP_PHONE_COMMAND_SPEAK (6)
Make the audio channel speakable.

EDNETP_PHONE_COMMAND_NOT_SPEAK (7)
Make the audio channel not speakable.

EDNETP_EVENT_PHONE_STATE (12)

Descirbes the phone status. Data format is `'audio-channel:state-no:boolean-value'`, where `audio-chnannel` is a audio channel number, `state-no` is a state type, and `boolean-value` is the value of the state.

This event is receive-only.

The list of phone states is following:

EDNETP_PHONE_STATE_NORMAL (0)
None

EDNETP_PHONE_STATE_CALL (1)
Requsted the call to the remote audio channel

EDNETP_PHONE_STATE_REQUESTED (2)
Received the call from the remote audio channel

EDNETP_PHONE_STATE_IGNORED (3)
Ignored the call from the remote audio channel

EDNETP_PHONE_STATE_REMOTE_LISTEN (4)
The remote audio channel is listening to the local channel

EDNETP_PHONE_STATE_REMOTE_SPEAK (5)
The remote audio channel is speaking to the local channel

EDNETP_PHONE_STATE_LISTEN (6)
Listening to the remote audio channel

EDNETP_PHONE_STATE_SPEAK (7)
Speaking to the remote audio channel

EDNETP_EVENT_SEARCH_COMMAND (13)

Sends requests for searching recorded file on the remote server. Date format is `'camera:command:parameter'`, where `camera` is a camera number, `command` is the search command, and `parameter` is the parameter of the command.

This event is send-only.

The list of available search commands is following:

EDNETP_SEARCH_COMMAND_DATE (0)
Requests for recorded date list.

EDNETP_SEARCH_COMMAND_TIME (1)
Requests for recorded time list in the specified day. `parameter` must contain the day in `YYYY-MM-DD` format.

EDNETP_EVENT_SEARCH_INFO (14)

Describes the response of searching recorded file on the remote server. Date format is `'camera:search-info:parameter'`, where `camera` is a camera number, `search-info` is the search information, and `parameter` is the parameter of the information.

This event is receive-only.

The list of the search information is following:

EDNETP_SEARCH_INFO_DATE_DATA (0)
Describes the date information. `parameter` contains the day in `YYYY-MM-DD` format.

EDNETP_SEARCH_INFO_TIME_DATA (1)
Describes the time information. `parameter` contains the time in `YYYYMMDD:YYYYMMDD:hhmmss:hhmmss` format, which consists of start and finish time of a recorded file.

EDNETP_EVENT_REPLAY_COMMAND (15)

Sends the requests for controlling replay of a recorded file. Date format is `'camera:command:parameter'`, where `camera` is a camera number, `command` is the replay command, and `parameter` is the additional parameter of the command.

This event is send-only.

The list of available replay commands is following:

EDNETP_REPLAY_COMMAND_START (0)
Starts to replay the recorded file. The `parameter` must contain the start time and initial pause state (1 or 0) in `YYYY-MM-DD:hhmmss:pause` format.

EDNETP_REPLAY_COMMAND_PAUSE (1)
Makes a pause to replay.

EDNETP_REPLAY_COMMAND_RESUME (2)
Resumes the replay which are previously paused.

EDNETP_REPLAY_COMMAND_STOP (3)
Stops the replay.

EDNETP_REPLAY_COMMAND_SEEK (4)
Seeks to the specified time in `hh:mm:ss` format.

EDNETP_REPLAY_COMMAND_PREV_FRAME (5)
Seeks backward in a frame.

EDNETP_REPLAY_COMMAND_NEXT_FRAME (6)
Seeks forward in a frame.

EDNETP_REPLAY_COMMAND_MULTI_SPEED (7)
Changes the speed of the replay. The `parameter` must contain a integer value in `speed` format like:

- `EDNETP_REPLAY_PLAY_SPEED_HALF (0)`
- `EDNETP_REPLAY_PLAY_SPEED_X1 (1)`
- `EDNETP_REPLAY_PLAY_SPEED_X2 (2)`
- `EDNETP_REPLAY_PLAY_SPEED_X4 (3)`

EDNETP_REPLAY_COMMAND_BACKUP (8)

Downloads the file which are played. The `parameter` must contain start(1) or cancel(0) downloading command in `boolean` format.

EDNETP_EVENT_REPLAY_INFO (16)

Describes current replay status. Data format is `'camera:replay-info:parameter'`, where `camera` is a camera number, `replay-info` is the replay information, and `parameter` is the additional parameter of the information.

This event is receive-only.

The list of the replay information is following:

EDNETP_REPLAY_INFO_START_STOP (0)

Describes the state of start or stop of the replay. If `parameter` is 1, it's started, otherwise it's stopped.

EDNETP_REPLAY_INFO_RESUME_PAUSE (1)

Describes the state of pause or resume of the replay. If `parameter` is 1, it's resumed, otherwise it's paused.

EDNETP_REPLAY_INFO_TIME_LINE (2)

Describes the ellapsed time of replaying the current file. `parameter` contains a time string in `hh:mm:ss` format.

EDNETP_REPLAY_INFO_MULTI_SPEED (3)

Describes the current replaying speed. `parameter` contains integer value. (see EDNETP_REPLAY_COMMAND_MULTI_SPEED(7))

EDNETP_REPLAY_INFO_BACKUP (4)

Describes the download status. `parameter` contains download information in `filename:progress:state` format, where the last value is the current state like:

- `EDNETP_REPLAY_BACKUP_STATE_NONE (0)`
- `EDNETP_REPLAY_BACKUP_STATE_PREPARE (1)`
- `EDNETP_REPLAY_BACKUP_STATE_START (2)`
- `EDNETP_REPLAY_BACKUP_STATE_CANCEL (3)`
- `EDNETP_REPLAY_BACKUP_STATE_COMPLETE (4)`
- `EDNETP_REPLAY_BACKUP_STATE_ERROR (5)`

EDNETP_EVENT_STILL_IMAGE (17)

Save the still images of the camera which are displayed currently. Data format is `'command:parameter'`, where `command` is the action type, and `parameter` is the additional parameter of the action.

EDNETP_ATTR_STILL_IMAGE_NAME (16) is used for the format of file name and EDNETP_ATTR_STILL_IMAGE_DIR (15) is used for the directory to save them.

This event is send-only.

The list of available commands is following:

EDNETP_STILL_IMAGE_DIALOG (0)

Saves still images after opening the dialog for users to select the directory to save images. `parameter` may contain default directory to select.

EDNETP_STILL_IMAGE_SAVE (1)

Saves still images directly. `parameter` must contains camera number to save in `camera` format.

EDNETP_EVENT_LOG_LEVEL (98)

Changes the level to receive log messages. Data format is `'log-level'`, where `log-level` is integer value, if it's 1, it will print only information but if it's 0, it will print debug messages also.

This event is send-only.

EDNETP_EVENT_LOG_MESSAGE (99)

Describes log messages from EDNet OCX. Data contains log message string.

This event is receive-only.

Attributes

Attributes can be retrieved with getAttr() [106] and some attributes may be changed with EDNETP_EVENT_ATTR (1) by applications. (You can lookup attribute numbers in `ednetp-const.js` in web client javascript examples)

EDNETP_ATTR_VERSION (0)

Describes the version of EDNet OCX. Data format is `'major:minor'`, where `major` is a major version number, and `minor` is a minotr version number.

This attribute is read-only.

EDNETP_ATTR_ERROR_CODE (1)

Describes the error status for the last action. Data format is `'error'`, where `error` is a error code.

This attribute is read-only.

EDNETP_ERROR_NONE (0)

No error occured

EDNETP_ERROR_RESOLVE (1)

Failed to resolve IP address

EDNETP_ERROR_CONNECT (2)

Failed to connect to the remote server

EDNETP_ERROR_LOGIN (3)

Failed to login due to invalid login information

EDNETP_ERROR_PERMISSION (4)
Failed to login due to insufficient permission

EDNETP_ERROR_CONNECTION_LIMIT (5)
Failed to login due to reach connection limitation

EDNETP_ATTR_STATE (2)

Describes the state of the connection to the remote server. Data format is `'state'`, where `state` is a status code.

This attribute is read-only.

EDNETP_STATE_OFFLINE (0)
Offline or Disconnected

EDNETP_STATE_RESOLVE (1)
Resolving host name to IP address

EDNETP_STATE_CONNECT (2)
Connecting to the remote server

EDNETP_STATE_INIT (3)
Initializing the connection

EDNETP_STATE_LOGIN (4)
Authenticating with user name and password

EDNETP_STATE_PREPARE (5)
Preparing to fetch basic information from the remote server

EDNETP_STATE_ONLINE (6)
Ready to streaming and control the remote server

EDNETP_ATTR_MODE (3)

Describes the operation mode. Data format is `'mode'`, where if `mode` is 0 it's live display, otherwise it's search replay.

EDNETP_ATTR_CAMERA_TOTAL (4)

Describes the number of cameras on the remote server. Data format is `'total'`, where `total` is a integer value.

This attribute is read-only.

EDNETP_ATTR_CAMERA (5)

Describes selected camera number. Data format is `'camera'`, where `camera` is a integer value.

EDNETP_ATTR_DISPLAY_MODE (6)

Describes display screen mode. Data format is `'rowsxcols'`, where `rows` is the number of rows in display screen, and `cols` is the number of columns in display screen.

EDNETP_ATTR_DISPLAY_RATIO (7)

Describes display ratio. Data format is `'ratio'`, where `ratio` is the ratio type.

EDNET_PLUGIN_DISPLAY_RATIO_NORMAL (0)
Fits to display screen.

EDNET_PLUGIN_DISPLAY_RATIO_4_3 (1)
Displays in 4:3 ratio.

EDNET_PLUGIN_DISPLAY_RATIO_ORIGINAL (2)
Displays in original ratio.

EDNETP_ATTR_DISPLAY_ACCEL (8)

Not supported any more.

EDNETP_ATTR_DISPLAY_BORDER (9)

Describes the display border color. Data format is `'size:bg-color:selected-color'`, where `size` is the thickness in pixel units, `bg-color` is the default border color described in '#RRGGBB' format, and `selected-color` is the selected border color described in '#RRGGBB' format.

EDNETP_ATTR_MUTE (10)

Describes the status of the mute. Data format is `'mute'`, where if `mute` is 0 mute is off, otherwise mute is on.

EDNETP_ATTR_VOLUME (11)

Describes the size of the volume. Data format is `'volume'`, where `volume` is a integer value between 0 and 128.

EDNETP_ATTR_ALARM_TOTAL (12)

Describes the number of relay devices on the remote server. Data format is `'relays'`, where `relays` is a integer value.

This attribute is read-only.

EDNETP_ATTR_AUDIO_TOTAL (13)

Describes the number of audio channels on the remote server. Data format is `'audios'`, where `audios` is a integer value.

This attribute is read-only.

EDNETP_ATTR_AUDIO_NAME (14)

Describes the name of the local audio channel. Data contains a string used for bidirectional audio communication.

EDNETP_ATTR_STILL_IMAGE_DIR (15)

Describes the directory to save still images. Data contains a full path string. Default is the dekstop folder of the user.

EDNETP_ATTR_STILL_IMAGE_NAME (16)

Describes the format of still image file names. Data is a string which consists of reserved format keywords like `%h` for host name, `%c` for camera number, `%d` for the date and `%t` for the time. Default is `%h-%c-%d-%t`.

EDNETP_ATTR_BACKUP_DIR (17)

Describes the directroy to download recorded mpeg files. Data contains a full path string. Default is the dekstop folder of the user.

Appendix#A.#Appendix

Change Log

This section describes the change log for EDNet DVR Network Library releases.

EDNet 1.4.1 (2016.11.29)

- Fix multiple ActiveX don't work properly [#3]
- Add Utonix SDK sample code [#2]

EDNet 1.4.0 (2015.06.16)

- Add support for 64-bit Windows [5162]
- Fix ActiveX display size not changed even though the window size is changed [4830]
- Fix ActiveX still image doesn't work [4819]

EDNet 1.3.7 (2014.04.10)

- Fix some MJPEG video color broken [4695]
- Remove dependency for DirectX library

EDNet 1.3.6 (2014.01.08)

- Fix ActiveX web client search replay doesn't work [4661]
- Add ActiveX search replay / backup sample code

EDNet 1.3.5 (2013.05.10)

- Fix example project build error

EDNet 1.3.4 (2013.04.02)

- Plug ActiveX memory leaks [4434]
- Fix ActiveX crash when the window size is changed in 1x1 mode [4386]

EDNet 1.3.3 (2012.08.18)

- Fix OCX crash with no sound cards [4297]

EDNet 1.3.2 (2012.03.05)

- Upgrade codec library [4141]

EDNet 1.3.1 (2012.02.13)

- Add support for the host with multiple cameras up to 100 channels [4162]

EDNet 1.3.0 (2009.07.31)

- Add support for H.264 codec [3752]
- Refactor internal structure to support various video / audio codecs [3765]

EDNet 1.2.6 (2009.05.18)

- Add english translation for EDNet SDK manual [3333]

EDNet 1.2.5 (2009.03.21)

- Add EDNet C/C++ API Reference to SDK manual [3585]
- Upgrade internal codec library [3668]

EDNet 1.2.4 (2008.12.12)

- Fix a 'Speak' bug of the phone (bidirectional audio communication) in web client [3509]
- Fix a bug when downloading big files in web client [3546]
- Fix a remote replay problem when a file has non-integer frame rates. [3548]

EDNet 1.2.3 (2008.10.17)

- Modify web client phone (bidirectional audio communication) GUI [3472]
- Add download(backup) support to web client [3473]

EDNet 1.2.2 (2008.10.08)

- Modify ActiveX plugin API to save still images [3477]
- Make the system not restart after install ActiveX components [3476]
- Add Linux ARM / MIPS platform supports [3481]
- Fix the problem where the audio sound of the camera which is not visible to the screen is audible unconditionally [3483]

EDNet 1.2.1 (2008.09.12)

- Add a Visual C++ OCX sample prgram [3442]
- Fix the high usage of CPUs when it plays audio [3457]
- Fix the memory reference error when bidirectional audio communication is used [3459]
- Make it invisible for ActiveX to show 4x4 screens at the time of web client start [3455]
- Make tooltips more faster in web client [3471]
- Fix the wrong calculation of total play time in web client search [3452]
- Fix the flash symptom when it moves in a frame in web client search [3468]
- Fix the empty area for 1 second in web client search [3469]
- Fix replay speed is reset to x1 in the middle of replay in web client search [3470]
- Fix the symptom replay stops somewhere in web client search [3474]
- Fix path name inconsistency for SDK examples [3451]

EDNet 1.2.0 (2008.08.22)

- Add remote search and replay support to web client [60]
- Add a phone (bidirectional audio communication) example to SDK [3441]

EDNet 1.1.2 (2008.07.02)

- Add volume control support to ActiveX control [3405]
- Fix the bug when it connections another host again [3375]

EDNet 1.1.1 (2008.05.09)

- Add a support for ActiveX in Windows Vista [3331]
- Fix playing audio frames from Sentry24NT not working [3340]
- Fix the example application doesn't draw images [3332]

EDNet 1.1.0 (2008.04.25)

- Add a ActiveX API reference to SDK manual [3296]
- Add a phone (bidirectional audio communication) support to web client [3162]
- Fix the screen mode problem when it connects the same host again [3275]
- Fix the high usage of CPU when web clients draw images [3274]

EDNet 1.0.0 (2008.03.12)

- Rewrite and Upgrade ActiveX(OCX) component [2640]