ONVIF Client Library

Interface Documentation

Happytimesoft Corporation | Version 1.0 | Copyright 2014-2026

Table of Contents

1. Introduction

This document describes the public C interfaces of the ONVIF Client Library. The library implements the ONVIF client protocol stack and provides device discovery, device management, media, PTZ, imaging, event and system maintenance functions, plus logging and buffer management helpers. All interfaces are exported as plain C functions so they can be called from both C and C++.

The interfaces are organized in the following header files under the OnvifClientLibrary directory:

Header file	Contents
onvif/onvif_probe.h	ONVIF WS-Discovery device discovery (probe / hello / bye)
onvif/onvif_api.h	High-level ONVIF device API (capabilities, media, PTZ, imaging, events, system)
onvif/onvif_event.h	Event notification handling (subscription, notify server, timer renew)
bm/sys_log.h	Logging facility (log levels, file rotation, atomic print)
bm/sys_buf.h	Network buffer pool management (sys_buf_init / sys_buf_deinit)
http/http_parse.h	HTTP message buffer pool management (http_msg_buf_init / http_msg_buf_deinit)

2. Common Definitions and Data Types

2.1 BOOL and HT_API

BOOL is defined as a signed char in bm/sys_inc.h (TRUE=1, FALSE=0). HT_API is the export macro: on Windows it expands to __declspec(dllexport/dllimport) depending on HT_EXPORTS; on Linux/iOS/macOS it expands to nothing. When building a static library on Windows, define HT_STATIC to make HT_API empty.

```
typedef signed char  BOOL;
#define TRUE  1
#define FALSE 0
#define HT_API  (empty / __declspec(dllexport|dllimport))
```

2.2 Key data structures

The following structures are frequently used by the API functions in this document.

DEVICE_BINFO (onvif/onvif.h) - device basic information used by discovery:

Field	Description
int type	Device type: ODT_UNKNOWN(0), ODT_NVT(1), ODT_NVD(2), ODT_NVS(3), ODT_NVA(4)
char EndpointReference[100]	Unique endpoint reference of the device
int MetadataVersion	Metadata version
uint32 sizeScopes	Number of scopes
onvif_Scope scopes[MAX_SCOPE_NUMS]	Device scopes

onvif_XAddr XAddr	Device XAddr (host, port and URL)

ONVIF_DEVICE (onvif/onvif.h) - the main device handle:

Field group	Description
binfo (DEVICE_BINFO)	Device basic information (filled by discovery / GetEndpointReference)
username / password	Authentication user name and password, set by the user
timeout	Request timeout in milliseconds
needAuth / authFailed	Authentication flags (ws-username-token vs digest)
errCode / fault	Last error code and ONVIF fault returned by the stack
curProfile	Current profile pointer (media service 1), user selectable
v_src / a_src	Video / audio source lists
profiles (ONVIF_PROFILE)	Profile list for media service 1
media_profiles	Media profile list for media service 2
v_src_cfg / a_src_cfg / v_enc_cfg / a_enc_cfg	Video/audio source and encoder configuration lists (media 1)
v_enc_cfg2 / a_enc_cfg2	Encoder configuration lists for media service 2
ptz_node / ptz_cfg	PTZ node and PTZ configuration lists
events (ONVIF_EVENT)	Event subscription information
DeviceInformation	Device information (manufacturer, model, firmware, ...)
Capabilities	Device capability set (media, ptz, image, event, ...)

ONVIF_PROFILE (onvif/onvif.h) - a media profile:

Each profile carries its name, token, the RTSP stream URI (filled by GetStreamUris) and pointers to the video/audio source, encoder, PTZ, metadata and analytics configurations bound to the profile. Profiles are linked with a 'next' pointer.

onvif_TransportProtocol (onvif/onvif_cm.h):

```
typedef enum {
    TransportProtocol_Invalid = -1,
    TransportProtocol_UDP     = 0,
    TransportProtocol_TCP     = 1,
    TransportProtocol_RTSP    = 2,
    TransportProtocol_HTTP    = 3
} onvif_TransportProtocol;
```

Notify_REQ (onvif/onvif_req.h) - event notification message:

Field	Description
char PostUrl[200]	URL the event message was posted to (HTTP notify endpoint)
NotificationMessageList * notify	List of notification messages

tev_PullMessages_RES (onvif/onvif_res.h) - PullMessages result:

Field	Description
time_t CurrentTime	Date/time when the messages were delivered by the server
time_t TerminationTime	Date/time when the pull point shuts down without further pulls
NotificationMessageList * NotifyMessages	List of pulled notification messages

3. Device Discovery Interfaces (onvif_probe.h)

These interfaces implement ONVIF WS-Discovery: the library periodically sends Probe messages over UDP multicast and reports matching devices (Probe Match), device hello (Hello) and device byebye (Bye) events through a callback. A separate monitor callback can be registered for a specific device endpoint reference.

3.1 Constants

Name	Value	Description
PROBE_MSGTYPE_MATCH	0	Probe match - a device responded to the probe request
PROBE_MSGTYPE_HELLO	1	Hello - a device announced itself
PROBE_MSGTYPE_BYE	2	Bye - a device announced it is leaving
MAX_PROBE_FD	32	Maximum number of probe sockets (one per address family/interface)

3.2 Callback and data types

```
typedef void (* onvif_probe_cb)(DEVICE_BINFO * p_res, int msgtype, void * pdata);

typedef struct { int family; SOCKET fd; } ONVIF_PROBE_FD;

typedef struct {
    onvif_probe_cb  probe_cb;
    void *          probe_cb_data;
    void *          probe_mutex;
    pthread_t       probe_thread;
    int             probe_interval;
    BOOL            probe_running;
    char            monitor_reference[100];
    int             monitor_msgtype;
    onvif_probe_cb  monitor_cb;
    void *          monitor_cb_data;
    void *          monitor_mutex;
    ONVIF_PROBE_FD  probe_fd[MAX_PROBE_FD];
} ONVIF_PROBE_CLS;
```

onvif_probe_cb is the discovery callback. msgtype is one of PROBE_MSGTYPE_MATCH, PROBE_MSGTYPE_HELLO or PROBE_MSGTYPE_BYE. For PROBE_MSGTYPE_BYE only the p_res-

>EndpointReference field is valid; all other fields are invalid. The pdata pointer is the user data passed when registering the callback.

3.3 Functions

```
HT_API void set_monitor_cb(char * reference, int msgtype, onvif_probe_cb cb, void
* pdata);
```

Sets a monitor callback that only reports discovery messages for a specific device endpoint reference.

Parameter	Description
reference	Endpoint reference of the device to monitor (see DEVICE_BINFO.EndpointReference)
msgtype	Probe message type to monitor: PROBE_MSGTYPE_MATCH / HELLO / BYE
cb	Callback function
pdata	User data passed back to the callback

```
HT_API void set_probe_cb(onvif_probe_cb cb, void * pdata);
```

Sets the global probe callback that reports all discovered devices.

Parameter	Description
cb	Callback function
pdata	User data passed back to the callback

```
HT_API void set_probe_interval(int interval);
```

Sets the probe sending interval. The default interval is 30 seconds.

Parameter	Description
interval	Probe interval in seconds

```
HT_API BOOL start_probe(const char * ip, int interval);
```

Starts the probe thread. The thread sends a Probe request and processes responses periodically.

Parameter	Description
ip	IP address to bind the probe sockets to. If NULL, the default IP address is used
interval	Probe interval in seconds (default 30)

Return value: TRUE on success, FALSE on failure.

- Call set_probe_cb before start_probe if you want to receive discovered devices.
- Call stop_probe to stop the discovery thread.

```
HT_API void stop_probe();
```

Stops the probe thread and closes the probe sockets.

```
HT_API void send_probe_req();
```

Manually sends a WS-Discovery Probe request (for example to trigger an immediate discovery scan).

4. High-level Device API (onvif_api.h)

All functions in this chapter take an ONVIF_DEVICE * p_dev as the first parameter. Before calling them, initialize the device with onvif_initDevice() (onvif/onvif_cln.h), which sets the host, port and HTTPS flag, and optionally set authentication with onvif_SetAuthInfo(). Every function performs a synchronous ONVIF request and returns TRUE on success or FALSE on failure. On failure, p_dev->errCode and p_dev->fault contain the error information and onvif_GetErrString(p_dev) returns a human readable error string.

4.1 Device management (capabilities, time, information)

```
HT_API BOOL GetCapabilities(ONVIF_DEVICE * p_dev);
```

ONVIF 1.0 compatible interface. Calls GetCapabilities to obtain the device capability set, which is saved in p_dev->Capabilities. The obtained capability set is merged with the existing capability set.

Return value: TRUE on success, FALSE on failure.

```
HT_API BOOL GetServices(ONVIF_DEVICE * p_dev);
```

Calls GetServices to obtain the device capability set and saves it in p_dev->Capabilities. The obtained capability set is merged with the existing capability set.

Return value: TRUE on success, FALSE on failure.

```
HT_API BOOL GetSystemDateAndTime(ONVIF_DEVICE * p_dev);
```

Gets the device date and time. The result is saved in p_dev->devTime. If p_dev->timeType is TIME_TYPE_LOCAL(1), p_dev->devTime holds the local time; if it is TIME_TYPE_UTC(2), p_dev->devTime holds the UTC time.

Return value: TRUE on success, FALSE on failure.

```
HT_API BOOL SetSystemDateAndTime(ONVIF_DEVICE * p_dev);
```

Sets the device date and time using the current local system time.

Return value: TRUE on success, FALSE on failure.

```
HT_API BOOL GetDeviceInformation(ONVIF_DEVICE * p_dev);
```

Retrieves device information (manufacturer, model, firmware version, serial number, hardware ID) and saves it in p_dev->DeviceInformation.

Return value: TRUE on success, FALSE on failure.

```
HT_API BOOL GetEndpointReference(ONVIF_DEVICE * p_dev);
```

Retrieves the device endpoint reference and saves it in p_dev->binfo.EndpointReference.

Return value: TRUE on success, FALSE on failure.

4.2 Media service (ONVIF Media 1)

```
HT_API BOOL GetProfiles(ONVIF_DEVICE * p_dev);
```

Calls GetProfiles to retrieve all device media profiles and saves them in p_dev->profiles. Existing profiles are released first.

Return value: TRUE on success, FALSE on failure.

- This is the ONVIF Media 1 service interface.

```
HT_API BOOL GetStreamUris(ONVIF_DEVICE * p_dev, onvif_TransportProtocol proto);
```

Retrieves the RTSP stream addresses of all device profiles and saves them in ONVIF_PROFILE.stream_uri.

Parameter	Description
proto	Transport protocol of the stream: TransportProtocol_UDP, TransportProtocol_TCP, TransportProtocol_RTSP or TransportProtocol_HTTP

Return value: TRUE on success, FALSE on failure.

- Call GetProfiles first so that all profiles of the device are available.
- This is the ONVIF Media 1 service interface.

```
HT_API BOOL GetVideoSourceConfigurations(ONVIF_DEVICE * p_dev);
```

Retrieves all video source configurations and saves them in p_dev->v_src_cfg. Existing configurations are released first.

Return value: TRUE on success, FALSE on failure.

- ONVIF Media 1 service interface.

```
HT_API BOOL GetAudioSourceConfigurations(ONVIF_DEVICE * p_dev);
```

Retrieves all audio source configurations and saves them in p_dev->a_src_cfg. Existing configurations are released first.

Return value: TRUE on success, FALSE on failure.

- ONVIF Media 1 service interface.

```
HT_API BOOL GetVideoEncoderConfigurations(ONVIF_DEVICE * p_dev);
```

Retrieves all video encoder configurations and saves them in p_dev->v_enc_cfg. Existing configurations are released first.

Return value: TRUE on success, FALSE on failure.

- ONVIF Media 1 service interface.

```
HT_API BOOL GetAudioEncoderConfigurations(ONVIF_DEVICE * p_dev);
```

Retrieves all audio encoder configurations and saves them in p_dev->a_enc_cfg. Existing configurations are released first.

Return value: TRUE on success, FALSE on failure.

- ONVIF Media 1 service interface.

```
HT_API BOOL GetVideoSources(ONVIF_DEVICE * p_dev);
```

Retrieves all video sources and saves them in p_dev->v_src. Existing sources are released first.

Return value: TRUE on success, FALSE on failure.

- ONVIF Media 1 service interface.

```
HT_API BOOL GetAudioSources(ONVIF_DEVICE * p_dev);
```

Retrieves all audio sources and saves them in p_dev->a_src. Existing sources are released first.

Return value: TRUE on success, FALSE on failure.

- ONVIF Media 1 service interface.

4.3 PTZ

```
HT_API BOOL GetNodes(ONVIF_DEVICE * p_dev);
```

Retrieves all PTZ nodes and saves them in p_dev->ptz_node. Existing nodes are released first.

Return value: TRUE on success, FALSE on failure.

- The device must support PTZ; p_dev->Capabilities.ptz.support must be 1.

```
HT_API BOOL GetConfigurations(ONVIF_DEVICE * p_dev);
```

Retrieves all PTZ configurations and saves them in p_dev->ptz_cfg. Existing configurations are released first.

Return value: TRUE on success, FALSE on failure.

- The device must support PTZ; p_dev->Capabilities.ptz.support must be 1.

4.4 Imaging

```
HT_API BOOL GetImagingSettings(ONVIF_DEVICE * p_dev);
```

Gets the image settings for all video sources on the device. The settings are saved in VideoSourceList.VideoSource.ImagingSettings.

Return value: TRUE on success, FALSE on failure.

- Call GetVideoSources first.
- The device must support the imaging service; p_dev->Capabilities.image.support must be 1.

4.5 Event subscription and pulling

```
HT_API BOOL Subscribe(ONVIF_DEVICE * p_dev, int index);
```

Subscribes to event notifications (event push mode). The internal timer automatically sends a Renew request before the TerminationTime expires.

Parameter	Description
index	The index is appended to the subscription reference address

Return value: TRUE on success, FALSE on failure.

- Call onvif_event_init first, and register the event notify callback with onvif_set_event_notify_cb.
- Register the subscribe-disconnect callback with onvif_set_subscribe_disconnect_cb to be notified when the subscription is disconnected.

```
HT_API BOOL Unsubscribe(ONVIF_DEVICE * p_dev);
```

Cancels the event notification subscription.

Return value: TRUE on success, FALSE on failure.

```
HT_API BOOL CreatePullPointSubscription(ONVIF_DEVICE * p_dev);
```

Creates an event pulling point (event pull mode).

Return value: TRUE on success, FALSE on failure.

```
HT_API BOOL PullMessages(ONVIF_DEVICE * p_dev, int timeout, int message_limit,
tev_PullMessages_RES * p_res);
```

Pulls event messages from a previously created pull point.

Parameter	Description
timeout	The timeout for obtaining messages, in seconds
message_limit	Maximum number of event messages to return
p_res	[out] The returned event message list (tev_PullMessages_RES)

Return value: TRUE on success, FALSE on failure.

- Call CreatePullPointSubscription first.
- Messages must be pulled before the timeout expires, otherwise the device may delete the pull point.

4.6 Snapshot

```
HT_API BOOL GetSnapshot(ONVIF_DEVICE * p_dev, const char * profile_token, unsigned
char ** p_buf, int * buflen);
```

Gets a JPEG snapshot of the device.

Parameter	Description
profile_token	Profile token of the profile used to capture the snapshot
p_buf	[out] Pointer to the snapshot buffer (must be freed by the caller)
buflen	[out] Length of the snapshot buffer

Return value: TRUE on success, FALSE on failure.

- On success the caller must call FreeBuff to release the snapshot buffer.

```
HT_API void FreeBuff(void * p_buf);
```

Frees a buffer previously returned by the library (e.g. the snapshot buffer from GetSnapshot).

Parameter	Description
p_buf	Pointer to the buffer to free

4.7 System maintenance

```
HT_API BOOL FirmwareUpgrade(ONVIF_DEVICE * p_dev, const char * filename);
```

Upgrades the device firmware by uploading the given firmware file.

Parameter	Description
filename	Path of the firmware upgrade file

Return value: TRUE on success, FALSE on failure.

```
HT_API BOOL SystemBackup(ONVIF_DEVICE * p_dev, const char * filename);
```

Downloads the device system configuration as a backup file.

Parameter	Description
`filename`	Path where the system backup is saved

Return value: TRUE on success, FALSE on failure.

```
HT_API BOOL SystemRestore(ONVIF_DEVICE * p_dev, const char * filename);
```

Restores the device system configuration from a backup file.

Parameter	Description
`filename`	Path of the system backup file to restore

Return value: TRUE on success, FALSE on failure.

4.8 Media service 2 (ONVIF Media 2)

```
HT_API BOOL tr2_GetProfiles(ONVIF_DEVICE * p_dev);
```

Retrieves all device media profiles using the Media 2 service and saves them in p_dev->media_profiles. Existing profiles are released first.

Return value: TRUE on success, FALSE on failure.

- ONVIF Media 2 service interface.

```
HT_API BOOL tr2_GetStreamUris(ONVIF_DEVICE * p_dev, const char * proto);
```

Retrieves the RTSP stream addresses of all device profiles and saves them in MediaProfile.stream_uri.

Parameter	Description
`proto`	Stream protocol string, one of: RtspUnicast (RTP as UDP unicast), RtspMulticast (RTP as UDP multicast), RTSP (RTP over TCP), RtspOverHttp (RTSP control and RTP tunnelled over HTTP/HTTPS)

Return value: TRUE on success, FALSE on failure.

- Call tr2_GetProfiles first.
- ONVIF Media 2 service interface.

```
HT_API BOOL tr2_GetVideoSourceConfigurations(ONVIF_DEVICE * p_dev);
```

Retrieves all video source configurations (Media 2) and saves them in p_dev->v_src_cfg.

Return value: TRUE on success, FALSE on failure.

- ONVIF Media 2 service interface.

```
HT_API BOOL tr2_GetAudioSourceConfigurations(ONVIF_DEVICE * p_dev);
```

Retrieves all audio source configurations (Media 2) and saves them in p_dev->a_src_cfg.

Return value: TRUE on success, FALSE on failure.

- ONVIF Media 2 service interface.

```
HT_API BOOL tr2_GetVideoEncoderConfigurations(ONVIF_DEVICE * p_dev);
```

Retrieves all video encoder configurations (Media 2) and saves them in p_dev->v_enc_cfg2.

Return value: TRUE on success, FALSE on failure.

- ONVIF Media 2 service interface.

```
HT_API BOOL tr2_GetAudioEncoderConfigurations(ONVIF_DEVICE * p_dev);
```

Retrieves all audio encoder configurations (Media 2) and saves them in p_dev->a_enc_cfg2.

Return value: TRUE on success, FALSE on failure.

- ONVIF Media 2 service interface.

5. Event Handling Interfaces (onvif_event.h)

These interfaces provide the event notification infrastructure. The library starts HTTP and/or HTTPS servers that receive event Notify messages pushed by devices, and an internal timer that periodically sends Renew requests for active subscriptions.

5.1 Callback types and structures

```
typedef void (* onvif_event_notify_cb)(Notify_REQ * p_req, void * pdata);
typedef void (* onvif_subscribe_disconnect_cb)(ONVIF_DEVICE * p_dev, void * pdata);

typedef struct {
    BOOL    used_flag;                // used flag
    ONVIF_DEVICE * p_dev;             // pointer to ONVIF_DEVICE
} ONVIF_TIMER_DEV;

typedef struct {
    uint32  init_flag   : 1;           // init flag
    uint32  http_enable : 1;           // http enable flag
    uint32  https_enable: 1;           // https enable flag
    uint32  timer_run   : 1;           // timer running flag
    uint32  reserved    : 28;
    onvif_event_notify_cb notify_cb;    // event notify callback
    void  * notify_cb_data;            // event notify callback data
    void  * notify_cb_mutex;           // event notify callback mutex
    onvif_subscribe_disconnect_cb disconnect_cb;
    void  * disconnect_cb_data;        // disconnect callback data
    void  * disconnect_cb_mutex;       // disconnect callback mutex
    char        http_host[128];        // http host
    char        https_host[128];       // https host
    HTTPSRV     http_srv;              // http server for receiving event notify
    HTTPSRV     https_srv;             // https server for receiving event notify
    HTIMER      event_timer_id;        // event timer id
    PPSN_CTX  * event_dev_fl;          // event device free list
    PPSN_CTX  * event_dev_ul;          // event device used list
    int         event_dev_max;         // max event devices
} ONVIF_EVENT_CLS;
```

onvif_event_notify_cb is invoked when a device pushes an event Notify message.
onvif_subscribe_disconnect_cb is invoked when an event subscription is disconnected.

5.2 Functions

```
HT_API BOOL onvif_event_init(BOOL http_enable, const char * http_srv_addr, uint16
http_srv_port,
                             BOOL https_enable, const char * https_srv_addr, uint16
https_srv_port,
                             const char * cert_file, const char * key_file, int
ipv6, int max_clients);
```

Initializes the event handler. It starts the HTTP and/or HTTPS servers used to receive event Notify messages pushed by devices.

Parameter	Description
http_enable	Enable the HTTP server flag
http_srv_addr	HTTP server listen address; NULL means listening on all interfaces
http_srv_port	HTTP server listen port
https_enable	Enable the HTTPS server flag
https_srv_addr	HTTPS server listen address; NULL means listening on all interfaces
https_srv_port	HTTPS server listen port
cert_file	Certificate file for HTTPS
key_file	Private key file for HTTPS
ipv6	Whether IPv6 is enabled for HTTP and HTTPS (non-zero enables IPv6)
max_clients	Maximum number of supported clients

Return value: TRUE on success, FALSE on failure.

- Must be called before Subscribe.

```
HT_API void onvif_event_deinit();
```

Deinitializes the event handler, stops the HTTP/HTTPS servers and the renew timer.

```
HT_API void onvif_set_event_notify_cb(onvif_event_notify_cb cb, void * pdata);
```

Sets the event notify callback invoked when an event Notify message is received.

Parameter	Description
cb	Callback function; set to NULL to disable the callback
pdata	User data passed back to the callback

```
HT_API void onvif_set_subscribe_disconnect_cb(onvif_subscribe_disconnect_cb cb,
void * pdata);
```

Sets the subscribe-disconnect callback invoked when an event subscription is disconnected.

Parameter	Description
cb	Callback function; set to NULL to disable the callback
pdata	User data passed back to the callback

```
HT_API void onvif_event_notify(HTTPCLN * p_user, HTTPMSG * rx_msg, XMLN * p_xml);
```

Event notify handler. It is called internally by http_soap_process when an event Notify message arrives; applications normally do not need to call it directly.

Parameter	Description
p_user	HTTP client connection
rx_msg	Received HTTP message
p_xml	Parsed XML message

```
HT_API BOOL onvif_event_timer_add(ONVIF_DEVICE * p_dev);
```

Adds a device to the event timer list so the internal timer sends Renew requests for its subscription.

Parameter	Description
p_dev	The ONVIF device to add

Return value: TRUE on success, FALSE on failure.

- Used internally after Subscribe; normally not required by applications.

```
HT_API BOOL onvif_event_timer_del(ONVIF_DEVICE * p_dev);
```

Removes a device from the event timer list.

Parameter	Description
p_dev	The ONVIF device to remove

Return value: TRUE on success, FALSE on failure.

- Used internally after Unsubscribe; normally not required by applications.

6. Logging Interfaces (bm/sys_log.h)

6.1 Log levels and modes

```
#define HT_LOG_TRC      0
#define HT_LOG_DBG      1
#define HT_LOG_INFO     2
#define HT_LOG_WARN     3
#define HT_LOG_ERR      4
#define HT_LOG_FATAL    5

#define HT_LOG_MODE_LOOP    0
#define HT_LOG_MODE_DAY     1
```

Level	Meaning
HT_LOG_TRC (0)	Trace - most verbose debugging output
HT_LOG_DBG (1)	Debug
HT_LOG_INFO (2)	Information

HT_LOG_WARN (3)	Warning
HT_LOG_ERR (4)	Error
HT_LOG_FATAL (5)	Fatal error

Messages with a level lower than the configured log level are discarded. The default log level is HT_LOG_ERR. Two file rotation modes are supported: loop mode (HT_LOG_MODE_LOOP) writes multiple rotating log files, and day mode (HT_LOG_MODE_DAY) creates one file per day and stops writing once the daily file reaches its maximum size.

6.2 Log context

```
typedef struct {
    FILE  * fp;         // Log file pointer
    void  * mutex;      // read-write lock
    int     level;      // Log level
    uint32  cur_size;   // The current write length of the log
    uint32  max_size;   // Maximum file size in KB
    int     cur_idx;    // Current log file index
    int     max_idx;    // Maximum file indexes
    int     rewind;     // Loop write log flag
    uint32  pre_time;   // previous time
    char    name[256];  // Log file name
    char    path[256];  // log path
} HT_LOG_CTX;
```

6.3 Functions

```
HT_API int log_init(const char * log_fname);
```

Initializes the logging facility and opens the log file for writing. If rewind (loop mode) is enabled, the file name is rotated with an index; otherwise the given name is used directly.

Parameter	Description
`log_fname`	Base log file name (a path can be included)

Return value: 0 on success, -1 on failure.

- Call log_set_* functions before log_init to configure level, size, index and path.

```
HT_API int log_time_init(const char * fname_prev);
```

Initializes logging in day mode: a file named '<fname_prev>-YYYYMMDD_HHMMSS.log' is created and a new file is created automatically every day.

Parameter	Description
`fname_prev`	Prefix of the log file name

Return value: 0 on success, -1 on failure.

```
HT_API int log_reinit(const char * log_fname);
```

Reopens the log file (used by the internal rotation logic in loop mode). The previous file is closed and a new one is opened.

Parameter	Description
log_fname	Base log file name

Return value: 0 on success, -1 on failure.

```
HT_API int log_time_reinit(const char * fname_prev);
```

Reopens the log file in day mode (used by the internal rotation logic).

Parameter	Description
fname_prev	Prefix of the log file name

Return value: 0 on success, -1 on failure.

```
HT_API void log_close();
```

Closes the log file and destroys the internal log mutex.

```
HT_API void log_set_level(int level);
```

Sets the log level. Messages below this level are not written.

Parameter	Description
level	One of HT_LOG_TRC .. HT_LOG_FATAL

```
HT_API int log_get_level();
```

Gets the current log level.

Return value: The current log level.

```
HT_API void log_set_rewind(int rewind);
```

Sets the log mode. rewind = HT_LOG_MODE_LOOP(0) uses loop file rotation; any other value (typically HT_LOG_MODE_DAY(1)) uses day mode.

Parameter	Description
rewind	Loop write log flag (0 = day mode, non-zero = loop mode)

```
HT_API int log_get_rewind();
```

Gets the current log mode flag.

Return value: The current rewind/loop flag.

```
HT_API void log_set_max_size(int max_size);
```

Sets the maximum log file size in kilobytes. Values <= 0 are clamped to 1024 KB. In loop mode a new file is started when the limit is reached; in day mode writing stops when the limit is reached.

Parameter	Description
max_size	Maximum file size in KB

```
HT_API int log_get_max_size();
```

Gets the maximum log file size in kilobytes.

Return value: The maximum file size in KB.

```
HT_API void log_set_max_idx(int max_idx);
```

Sets the maximum number of log files in loop mode. Values <= 1 are clamped to 3.

Parameter	Description
`max_idx`	Maximum number of log file indexes

```
HT_API int log_get_max_idx();
```

Gets the maximum number of log files.

Return value: The maximum number of log file indexes.

```
HT_API void log_set_path(const char * path);
```

Sets the directory where log files are stored. If path is NULL or empty, files are written to the current directory.

Parameter	Description
`path`	Log directory path

```
HT_API int log_print(int level, const char * fmt, ...);
```

Writes a formatted log message with the given level (not available on iOS builds). The output line is prefixed with a timestamp and the level string. The internal lock is taken for the whole line.

Parameter	Description
`level`	Log level of the message
`fmt, ...`	printf-style format string and arguments

Return value: Number of characters written, or 0 if the level is filtered out.

- The line is written atomically (mutex-protected), so log_print can be used from multiple threads.

```
HT_API int log_ios_print(int level, const char * fmt, ...);
```

iOS variant of log_print. On iOS builds the log_print macro is mapped to log_ios_print.

Parameter	Description
`level`	Log level of the message
`fmt, ...`	printf-style format string and arguments

Return value: Number of characters written, or 0 if the level is filtered out.

```
HT_API int log_lock_start(const char * fmt, ...);
```

Writes a formatted message while holding the log lock. It is the first function of the log_lock_start / log_lock_print / log_lock_end group used to write a multi-part message atomically.

Parameter	Description
`fmt, ...`	printf-style format string and arguments

Return value: Number of characters written.

```
HT_API int log_lock_print(const char * fmt, ...);
```

Writes a formatted message while the lock acquired by log_lock_start is held.

Parameter	Description
`fmt, ...`	printf-style format string and arguments

Return value: Number of characters written.

```
HT_API int log_lock_end(const char * fmt, ...);
```

Writes the last part of a multi-part log message and releases the log lock acquired by log_lock_start.

Parameter	Description
`fmt, ...`	printf-style format string and arguments

Return value: Number of characters written.

7. Buffer Management Interfaces

These four additional interfaces manage the internal memory buffer pools of the library. They should be called once during application initialization and once during cleanup, in the order shown below.

7.1 Network buffer pool (bm/sys_buf.h)

```
HT_API BOOL sys_buf_init(int nums);
```

Initializes the network buffer pool. Internally it initializes the network buffer pool with nums buffers (NET_BUFF_SIZE each) and the derived-header buffer pool with 8*nums buffers.

Parameter	Description
`nums`	Number of network buffers to allocate

Return value: TRUE on success, FALSE on failure.

- Call once at startup, before any ONVIF/HTTP request is made.
- If initialization fails (memory allocation error), the library logs the failure with log_print at HT_LOG_ERR level.

```
HT_API void sys_buf_deinit();
```

Deinitializes the network buffer pool, releasing all buffers allocated by sys_buf_init.

7.2 HTTP message buffer pool (http/http_parse.h)

```
HT_API BOOL http_msg_buf_init(int num);
```

Initializes the HTTP message buffer pool. It pre-allocates num HTTPMSG objects in a free list used to parse received HTTP messages.

Parameter	Description
num	Number of HTTPMSG buffers to pre-allocate

Return value: TRUE on success, FALSE on failure.

- Call once at startup, after sys_buf_init.
- The function is idempotent: if the pool is already initialized it returns TRUE without reallocating.

```
HT_API void http_msg_buf_deinit();
```

Deinitializes the HTTP message buffer pool, releasing all pre-allocated HTTPMSG buffers.

8. Typical Usage Flow

A typical application flow is summarized below:

```
// 1. initialize buffers and logging
log_init("onvifclient.log");
log_set_level(HT_LOG_INFO);
sys_buf_init(64);
http_msg_buf_init(64);

// 2. event handler (for event push subscriptions)
onvif_event_init(TRUE, NULL, 8899, FALSE, NULL, 0, NULL, NULL, 0, 16);
onvif_set_event_notify_cb(my_notify_cb, NULL);
onvif_set_subscribe_disconnect_cb(my_disconnect_cb, NULL);

// 3. discovery (optional)
set_probe_cb(my_probe_cb, NULL);
start_probe(NULL, 30);

// 4. create and initialize a device
ONVIF_DEVICE dev;
memset(&dev, 0, sizeof(dev));
onvif_initDevice(&dev, "192.168.1.64", 80, FALSE);
onvif_SetAuthInfo(&dev, "admin", "12345");
GetCapabilities(&dev);
GetDeviceInformation(&dev);
GetProfiles(&dev);
GetStreamUris(&dev, TransportProtocol_TCP);   // dev.profiles->stream_uri
Subscribe(&dev, 1);                            // event push subscription

// 5. cleanup
Unsubscribe(&dev);
stop_probe();
```

```
onvif_event_deinit();
http_msg_buf_deinit();
sys_buf_deinit();
log_close();
```

9. Header File Reference

Interface	Header file
set_monitor_cb / set_probe_cb / set_probe_interval / start_probe / stop_probe / send_probe_req	onvif/onvif_probe.h
GetCapabilities .. tr2_GetAudioEncoderConfigurations	onvif/onvif_api.h
onvif_event_init .. onvif_event_timer_del	onvif/onvif_event.h
log_init .. log_ios_print	bm/sys_log.h
sys_buf_init / sys_buf_deinit	bm/sys_buf.h
http_msg_buf_init / http_msg_buf_deinit	http/http_parse.h

References

ONVIF Application Programmers Guide: https://www.onvif.org/wp-content/uploads/2016/12/ONVIF_WG-APG-Application_Programmers_Guide-1.pdf