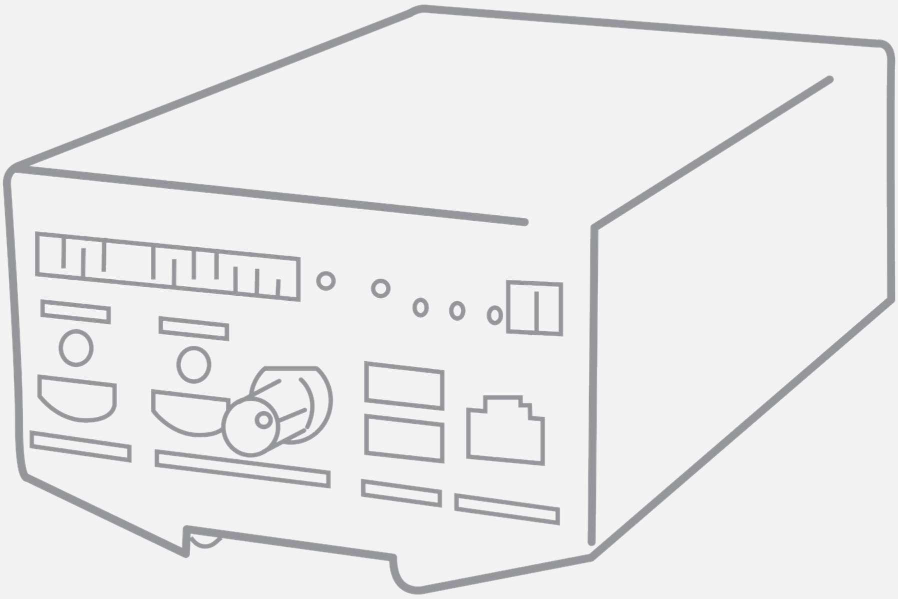
euresys
Empowering Computer Vision

REFERENCE MANUAL

Picolo.net HD1



This documentation is provided with Picolo.net HD1 2.7.3 (doc build 3038).
www.euresys.com

Contents

PART I

THE ONVIF SERVICES

1. The ONVIF Device Service

See also: "The Proprietary Device Service" on page 12 | Using the Device Service

The ONVIF device service is the entry point to all other services provided by your device.

You can easily:

□ List the capabilities provided by your device.

□ Configure the network settings.

□ Manage the system: get device info, start backup, set and get the date and the time...

□ Manage the device security configuration: access policy, user credentials, certificates...

Mandatory network capabilities

□ IPv4 with static IP configuration

□ IPv4 with dynamic IP configuration (DHCP)

Mandatory discovery capabilities

□ Target service role (WS-Discovery) on port 80

□ Discoverable and non-discoverable modes

□ Management of the messages "Hello", "Status changes", "Probe and Resolve" and "Bye"

□ Scopes

Mandatory system capabilities

□ List of supported ONVIF versions: 1.0 and 1.02

□ System support information

Mandatory security capabilities

□ Access security policy: "Administrator", "Operator", "User" and "Anonymous"

□ Default access policy

2. The ONVIF Media Service

See also: "The Proprietary Media Service" on page 13 | Using the Media Service

The ONVIF media service provides functions to configure the streaming properties of the media streams.

You can easily:

- Configure the ONVIF Media Profiles.
- Configure the video sources and the video encoders.
- Configure the audio sources and the audio encoders.
- Configure the metadata streams.
- Request the stream URI.
- Configure the overlays.

Mandatory codec capabilities

- JPEG video encoding - QVGA resolution
- G.711 µ-law audio encoding
- JPEG video encoding - Other than QVGA resolution
- H.264 video encoding

Mandatory streaming capabilities

- RTP / RTCP
- RTP over UDP - Unicast
- RTP interleaved in RTSP over HTTP
- RTP interleaved in RTSP over HTTPS
- RTP payloads for the formats supported by the device
- RTP metadata payload
- RTSP Port 554 as default session description using SDP
- RTSP Metadata Stream Description

3. The ONVIF Recording Service

See also: "The Proprietary Recording Service" on page 14 | Using the Recording Service

The ONVIF recording service provides function to interact with the network storage devices and cameras with embedded storage.

You can easily:

- □ Configure the local media profile to record (on the attached USB media).
- □ Start and stop the recording job.

The WSDL filename is "recording.wsdl".

As of version 2.6, Picolo.net HD1 provides a limited implementation of the ONVIF recording services with only one pre-existing recording job and one pre-existing recording using a local media profile. Picolo.net HD1 does not generate events so far.

4. The ONVIF Event Service

The ONVIF event service provides functions to manage the events.

You can easily:

- Find out the notifications that a device supports and the information these notifications contain.
- Poll the device to check for events using the Real-time Pull-Point Notification Interface.
- Be notified by the device of selected events occurrence.

Mandatory capabilities

- Basic notification interface as specified in WS-BaseNotification and WS-Topics specifications
- Real-time Pull-Point Notification Interface
- Notification Streaming Interface

5. The ONVIF PTZ Service

See also: "The Proprietary PTZ Service" on page 15 | Using the PTZ Service

The ONVIF media service provides functions to control the NVT pan, tilt and zoom.

The WSDL filename is "ptz.wsdl".

Mandatory capabilities

- Standard commands:
  - □ Get PTZ node properties
  - □ Get and set PTZ configurations
  - □ Get PTZ configurations options
  - □ Continuous pan/tilt/zoom movements
  - □ Stop movement
  - □ Get status
- Standard Pelco commands:
  - □ Zoom Wide
  - □ Zoom Tele
  - □ Down
  - □ Up
  - □ Left
  - □ Right
- Extended Pelco commands:
  - □ Set Preset
  - □ Clear Preset
  - □ Go To Preset
  - □ Set Zoom Speed
  - □ Recording PTZ presets

6. The ONVIF Device IO Service

See also: "The Proprietary Device IO Service" on page 16 | Using the Device IO Service

Sending Custom Commands over the Serial Port

The ONVIF device IO service provides functions to retrieve and configure the inputs and outputs of a device.

You can easily:

- Retrieve and configure the video inputs.
- Retrieve and configure the audio inputs.
- Retrieve and configure the audio outputs.
- Retrieve and configure the relay outputs.

The WSDL filename is "deviceio.wsdl".

Mandatory capabilities

For an ONVIF 2.0 NVT device:

- Number of video sources

For relay outputs:

- DeviceIORelayOutputs
- DeviceIORelayOutputsMonostableClosed
- DeviceIORelayOutputsMonostableOpen
- DeviceIORelayOutputsMonostable
- DeviceIORelayOutputsBistableClosed
- DeviceIORelayOutputsBistableOpen
- DeviceIORelayOutputsBistable

For serial ports:

- Set/GetSerialSerialPortConfiguration
- SendReceiveSerialCommand with Timeout values ranging between 1 ms and 5 seconds.

Picolo.net HD1 automatically uses base64 encoding (binary payload) for replies captured from the serial line when the request used base64 encoding.

PART II

THE PROPRIETARY SERVICES

1. The Proprietary Device Service

See also: "The ONVIF Device Service" on page 5 | Using the Device Service

With the proprietary device service, you can:

- Get the internal temperature of the device.
- Retrieve and configure additional security parameters.

The WSDL filename is `hd4DeviceProprietary.wsdl`.

The XML schema is `hd4DeviceProprietary.xsd`.

The namespace for this service is `http://www.euresys.com/picolo/device`.

This namespace is used in the `<device:Namespace>` element of `<device:Service>` items found in response to ONVIF `GetServices()` calls.

2. The Proprietary Media Service

See also: "The ONVIF Media Service" on page 6 | Using the Media Service

With the proprietary media service, you can:

- □ Perform the auto setup of ONVIF Media Profiles.
- □ Manage the audio outputs.

The WSDL filename is `hd4MediaProprietary.wsdl`.

The XML schema is `hd4MediaProprietary.xsd`.

The namespace for this service is `http://www.euresys.com/picolo/media`.

This namespace is used in the `<device:Namespace>` element of `<device:Service>` items found in response to ONVIF `GetServices()` calls.

3. The Proprietary Recording Service

See also: "The ONVIF Recording Service" on page 7 | Using the Recording Service

With the proprietary recording service, you can:

- Query the status and set the AES storage.
- List the stored contents.
- Use the `ExtendedRecordingConfiguration` function.

The namespace for this service is `http://www.euresys.com/picolo/recording`.

This namespace is used in the `<device:Namespace>` element of `<device:Service>` items found in response to ONVIF `GetServices()` calls.

4. The Proprietary PTZ Service

See also: "The ONVIF PTZ Service" on page 9 | Using the PTZ Service

With the proprietary PTZ service, you can:

- Set and get the serial port configuration of the COM IO device.
- Set and get the address configuration of each PTZ node.
- Get the address configurations of all the PTZ nodes.

The WSDL filename is `hd4PTZProprietary.wsdl`.

The XML schema is `hd4PTZProprietary.xsd`.

The namespace for this service is `http://www.euresys.com/picolo/ptz`.

This namespace is used in the `<device:Namespace>` element of `<device:Service>` items found in response to ONVIF `GetServices()` calls.

5. The Proprietary Device IO Service

See also: "The ONVIF Device IO Service" on page 10 | Using the Device IO Service

Sending Custom Commands over the Serial Port

The proprietary device IO service extends the ONVIF device IO service to manage the alarm input. It allows to:

- Set and get the configuration of the alarm inputs
- Get the state of the alarm inputs

The WSDL filename is `hd4IOProprietary.wsdl`.

The XML schema is `hd4IOProprietary.xsd`.

The namespace for this service is `http://www.euresys.com/picolo/deviceio`.

This namespace is used in the `<device:Namespace>` element of `<device:Service>` items found in response to ONVIF `GetServices()` calls.

PART III

THE CUSTOM SERVICES

1. The ERSP - Euresys Remote Serial Protocol

ERSP (**Euresys Remote Serial Protocol**) is a wrapper protocol above TCP/IP that allows the transparent delivery of binary commands to serial devices over RS-232 and RS-485 connectors and the retrieval of the serial devices replies.

- One ERSP server instance binds one TCP port with one serial connector. It allows exactly one client session before terminating.
- The current protocol version is 1.
- A sample client for this protocol is available in python 2.x language in the `remoteSerial` folder of the sample programs.

Header structure

The commands and the replies use 4-byte headers structured as follow:

0	1	2	3
command code	options	size (MSB)	size (LSB)

A command or reply can carry payload in addition to that header, in which case the size of the payload is stored in the two `size` fields.

Command Codes

To ease protocol supervision, command codes are ASCII letters.

ASCII command code	meaning	uses `options`	has data
A	ACKnowledge reply	N	sometimes
A	AUTHorized	Y	N
E	ERRor reply	Y	sometimes
G	`GET` request	N	sometimes
H	`HELLO` message	Y	Y
S	`SEND` request	N	Y
W	`WAIT` request	Y	N
Z	`ZAP` request	Y	N

Authentication

- A client sends a {`HELLO, opt=<client version number>, data=<authentication code>`} message.
- A malformed hello message or invalid authentication codes terminate the connection but the server keeps running.
- The server replies with {`AUTH, opt=<server version number>, size=0`}.

The current protocol version is 1, which is encoded as a `V`.

Data transfers

- {`SEND,size=N, data=<N bytes of raw data>`}
  - □ Request delivery of up to 65535 bytes of data over the serial line
  - □ Replied with {`ACK,size=0`} or {`ERR, opt=error pointer`}
- {`GET, size=0`}
  - □ Request discarding of all received bytes so far
  - □ Replied with {`ACK, size=0`}
- {`GET, size=N`}
  - □ Request read of up to 65535 bytes of data over the serial line
  - □ Can return bytes buffered before the command was issued
  - □ Replies {`ACK, size=N, data=<N bytes of raw data>`} on success
  - □ Replies {`ERR, opt=3, size=M, data=<M bytes of raw data`} if retrieval timed out after `M < N` bytes

Synchronization commands

- {`WAIT, opt=synchronization byte`}
  - □ Request discarding of all incoming bytes unless the synchronization byte is received
  - □ The synchronization byte is still returned in the next `GET` request
  - □ Replies {`ACK, size=0`} if the synchronization byte was found
  - □ Replies {`ERR, opt=1`} if no synchronization byte was found when the reading timeout expired
  - □ Replies {`ERR, opt=4`} if other I/O errors occurred while waiting for the synchronization byte
- {`ZAP, opt=<byte to discard>`}
  - □ Request discarding of 0 or more bytes of a specific value
  - □ Replies {`ACK, size=0`} if a byte of a value different than the byte to discard was found

- □ Replies {`ERR, opt=1`} if no byte different than the byte to discard was found when the reading timeout expired
- □ Replies {`ERR, opt=4`} if other I/O errors occurred while waiting for the synchronization byte

Error replies

- The `options` byte of the `ERR` message header indicates the offending byte in the corresponding request.

  For example:

  - □ `opt=0` indicates an invalid command.
  - □ `opt=1` indicates an invalid option for a command.
  - □ `opt=2` indicates an excessive amount of data.
  - □ `opt=3` indicates an insufficient amount of data.

- The error messages can carry an additional error string for human debuggers. In such a case the size of the string is stored in `size`.
- Error types:
  - □ A timeout while reading is the only non-fatal error in protocol version 1.
  - □ All other errors will close the connection as soon as the error message is sent.