acontis technologies GmbH

SOFTWARE

EC-Master

Feature Pack EoE-Endpoint

Version 3.2

Edition: September 23, 2025

Contents

1 Introduction

The EoE module within the Master acts as an intermediary between the EtherCAT master stack and the operating system's TCP/IP/ETH protocol stack. It operates as an Ethernet switch, which connects the slaves, capable of handling the EOE protocol, with the operating system.

- The EoE-Endpoint can be used with different operating systems.
- It works as a standard Ethernet driver / Network Interface Card (NIC) driver
- It requires minimal adaption to the operating system.
- It is simple to implement
  - Single function to register all EoE service function hooks
  - follows the open/read/write/close lifecycle of many other drivers
  - flexible data buffer handling
  - polling or interrupt driven operation

2 Programmer's Guide

2.1 emEoeRegisterEndpoint

static EC_T_DWORD **ecatEoeRegisterEndpoint** (
const EC_T_CHAR *szEoEDrvIdent,
EC_T_LINK_DRV_DESC *pLinkDrvDesc
)

EC_T_DWORD **emEoeRegisterEndpoint** (
EC_T_DWORD dwInstanceID,
const EC_T_CHAR *szEoEDrvIdent,
EC_T_LINK_DRV_DESC *pLinkDrvDesc
)

Register the EoE Endpoint (driver)

Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **szEoEDrvIdent** – [in] String identifying the EoE endpoint, e.g. "Eoe0".
- **pLinkDrvDesc** – [out] Pointer to a structure with information about exported driver/adapter functions. The function pointers are filled during this function call.

Returns

EC_E_NOERROR or error code

struct **EC_T_LINK_DRV_DESC**

Public Members

EC_T_DWORD **dwValidationPattern**
Link Layer driver validation pattern

EC_T_DWORD **dwInterfaceVersion**
Link Layer driver interface version

EC_T_LOG_PARMS **LogParms**
Log parameters structure

PF_EcLinkOpen **pfEcLinkOpen**
Function pointer to open link

PF_EcLinkClose **pfEcLinkClose**
Function pointer to close link

PF_EcLinkSendFrame **pfEcLinkSendFrame**
Function pointer to send frame

PF_EcLinkSendAndFreeFrame **pfEcLinkSendAndFreeFrame**
Function pointer to send and free frame

PF_EcLinkRecvFrame **pfEcLinkRecvFrame**
Function pointer to receive frame

PF_EcLinkAllocSendFrame **pfEcLinkAllocSendFrame**
Function pointer to allocate a frame buffer used to send

PF_EcLinkFreeSendFrame **pfEcLinkFreeSendFrame**
Function pointer to release a frame buffer previously allocated with AllocSendframe()

PF_EcLinkFreeRecvFrame **pfEcLinkFreeRecvFrame**
Function pointer to release a frame buffer given to the EtherCAT master through the receive callback function

PF_EcLinkGetEthernetAddress **pfEcLinkGetEthernetAddress**
Function pointer to get Link Layer MAC address

PF_EcLinkGetStatus **pfEcLinkGetStatus**
Function pointer to get current link status

PF_EcLinkGetSpeed **pfEcLinkGetSpeed**
Function pointer to get current link speed

PF_EcLinkGetMode **pfEcLinkGetMode**
Function pointer to get current link mode

PF_EcLinkIoctl **pfEcLinkIoctl**
Function pointer to a multi-purpose Link Layer IOCTL

EC_T_VOID ***pvLinkInstance**
Pointer to the Link Layer Object/Instance

struct **EC_T_LOG_PARMS**

Public Members

EC_T_DWORD **dwLogLevel**
[in] Log level. See EC_LOG_LEVEL_…

EC_PF_LOGMSGHK **pfLogMsg**
[in] Log callback function called on every message

EC_T_LOG_CONTEXT ***pLogContext**
[in] Log context to be passed to log callback

2.2 emEoeUnregisterEndpoint

static EC_T_DWORD **ecatEoeUnregisterEndpoint** (*EC_T_LINK_DRV_DESC* *pLinkDrvDesc)

EC_T_DWORD **emEoeUnregisterEndpoint** (
EC_T_DWORD dwInstanceID,
EC_T_LINK_DRV_DESC *pLinkDrvDesc
)

Unregisters a previously registered endpoint.

Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **pLinkDrvDesc** – [in] Pointer to a structure with information about registered endpoint.

Returns
EC_E_NOERROR or error code

2.3 pfEcLinkOpen

typedef EC_T_DWORD (***PF_EcLinkOpen**)(EC_T_VOID *pvLinkParms, EC_T_RECEIVEFRAMECALLBACK pfReceiveFrameCallback, EC_T_LINK_NOTIFY pfLinkNotifyCallback, EC_T_VOID *pvContext, EC_T_VOID **ppvInstance)

Open Link Layer connection.

Parameters

- **pvLinkParms** – [in] Pointer to link parameters
- **pfReceiveFrameCallback** – [in] Pointer to Rx callback function
- **pfLinkNotifyCallback** – [in] Pointer to notification callback function
- **pContext** – [in] Context pointer. This pointer is used as parameter when the callback function is called
- **ppvInstance** – [out] Instance handle

Returns
EC_E_NOERROR or error code

The first input parameter `pvLinkParms` is of type *EC_T_LINK_OPENPARMS_EOE*

struct **EC_T_LINK_OPENPARMS_EOE**

Variables to identify the EOE link layer driver/instance which shall be opened with the EoELinkOpen() call.

Note: Parameters not used for identification of the link layer (or master stack instance) must be cleared to 0.

Public Members

EC_T_DWORD **dwEmInstanceID**
Instance ID, identical to the instance ID of the EtherCAT master intended to open.

EC_T_PVOID **pvUplinkInstance**
Pointer to the `CEcEoEUplink` instance/object (if available).

EC_T_BYTE **abyEthAddress**[6]
Ethernet address of the driver/adapter

EC_T_BYTE **abyIpAdress**[4]
IP address of the driver/adapter

EC_T_CHAR **szEoEDrvIdent**[EC_DRIVER_IDENT_NAMESIZE]
Name of the driver/adapter

2.4 pfEcLinkClose

typedef EC_T_DWORD (***PF_EcLinkClose**)(EC_T_VOID *pvInstance)
Close Link Layer connection.

Parameters
pvInstance – [in] Instance handle

Returns
EC_E_NOERROR or error code

2.5 pfEcLinkSendFrame

typedef EC_T_DWORD (***PF_EcLinkSendFrame**)(EC_T_VOID *pvInstance, *EC_T_LINK_FRAMEDESC* *pLinkFrameDesc)
Send data frame.

Parameters

- **pvInstance** – [in] Instance handle
- **pLinkFrameDesc** – [in] Pointer to the link frame descriptor

Returns
EC_E_NOERROR or error code

struct **EC_T_LINK_FRAMEDESC**
Frame Descriptor.

Public Members

EC_T_VOID *`pvContext`
Link Layer context

EC_T_BYTE *`pbyFrame`
Frame data buffer

EC_T_DWORD `dwSize`
Frame data buffer size

EC_T_BOOL `bBuffersFollow`
If EC_TRUE try to queue next frame in link layer, if EC_FALSE fill up DMA descriptors to force immediate send

EC_T_DWORD *`pdwTimeStampLo`
Timestamp buffer

EC_T_DWORD *`pdwTimeStampPostLo`
Timestamp buffer

EC_T_PVOID `pvTimeStampCtxt`
Context for pfnTimeStamp

EC_T_LINK_GETTIMESTAMP `pfnTimeStamp`
Get timestamp function (optional)

EC_T_DWORD *`pdwLastTSResult`
Get timestamp result buffer

EC_T_WORD `wTimestampOffset`
Timestamp location in frame data buffer as byte offset

EC_T_WORD `wTimestampSize`
Timestamp size in bytes

EC_T_UINT64 `qwTimestamp`
Send or receive timestamp in ns

EC_T_DWORD `dwRepeatCnt`
Repeat count, if 0 or 1 send once

EC_T_DWORD `dwFlags`
Link frame flags, see ECAT_LINK_FRAMEFLAG_…

2.6 pfEcLinkSendAndFreeFrame

typedef EC_T_DWORD (***PF_EcLinkSendAndFreeFrame**)(EC_T_VOID *pvInstance, *EC_T_LINK_FRAMEDESC* *pLinkFrameDesc)

Send data frame and free the frame buffer. This function must be supported if EcLinkAllocSendFrame() is supported.

Parameters

- **pvInstance** – [in] Instance handle
- **pLinkFrameDesc** – [in] Pointer to the link frame descriptor

Returns

EC_E_NOERROR or error code

2.7 pfEcLinkRecvFrame

typedef EC_T_DWORD (***PF_EcLinkRecvFrame**)(EC_T_VOID *pvInstance, *EC_T_LINK_FRAMEDESC* *pLinkFrameDesc)

Poll for received frame.

Parameters

- **pvInstance** – [in] Instance handle
- **pLinkFrameDesc** – [in] Pointer to the link frame descriptor

Returns

EC_E_NOERROR or error code

2.8 pfEcLinkAllocSendFrame

typedef EC_T_DWORD (***PF_EcLinkAllocSendFrame**)(EC_T_VOID *pvInstance, *EC_T_LINK_FRAMEDESC* *pLinkFrameDesc, EC_T_DWORD dwSize)

Allocate a frame buffer used for send. If the link layer doesn’t support frame allocation, this function must return EC_E_NOTSUPPORTED.

Parameters

- **pvInstance** – [in] Instance handle
- **pLinkFrameDesc** – [in/out] Pointer to the link frame descriptor
- **dwSize** – [in] Size of the frame to allocate

Returns

EC_E_NOERROR or error code

2.9 pfEcLinkFreeSendFrame

typedef EC_T_VOID (***PF_EcLinkFreeSendFrame**)(EC_T_VOID *pvInstance, *EC_T_LINK_FRAMEDESC* *pLinkFrameDesc)

Release a frame buffer previously allocated with EcLinkAllocSendFrame()

Parameters

- **pvInstance** – [in] Instance handle
- **pLinkFrameDesc** – [in] Pointer to the link frame descriptor

Returns

EC_E_NOERROR or error code

2.10 pfEcLinkFreeRecvFrame

typedef EC_T_VOID (***PF_EcLinkFreeRecvFrame**)(EC_T_VOID *pvInstance, *EC_T_LINK_FRAMEDESC* *pLinkFrameDesc)

Release a frame buffer given to the EtherCAT master through the receive callback function.

Parameters

- **pvInstance** – [in] Instance handle
- **pLinkFrameDesc** – [in] Pointer to the link frame descriptor

Returns

EC_E_NOERROR or error code

2.11 pfEcLinkGetEthernetAddress

typedef EC_T_DWORD (***PF_EcLinkGetEthernetAddress**)(EC_T_VOID *pvInstance, EC_T_BYTE *pbyEthernetAddress)

Get Link Layer MAC address.

Parameters

- **pvInstance** – [in] Instance handle
- **pbyEthernetAddress** – [out] Ethernet MAC address

Returns

EC_E_NOERROR or error code

2.12 pfEcLinkGetStatus

typedef EC_T_LINKSTATUS (***PF_EcLinkGetStatus**)(EC_T_VOID *pvInstance)

Get current link status.

Parameters

pvInstance – [in] Instance handle

Returns

Current link status.

2.13 pfEcLinkGetSpeed

typedef EC_T_DWORD (***PF_EcLinkGetSpeed**)(EC_T_VOID *pvInstance, EC_T_DWORD *pdwSpeed)

Get current link speed.

Parameters

- **pvInstance** – [in] Instance handle
- **pdwSpeed** – [out] Current link speed

Returns

EC_E_NOERROR or error code

2.14 pfEcLinkGetMode

typedef EC_T_LINKMODE (***PF_EcLinkGetMode**)(EC_T_VOID *pvInstance)

Get current link mode.

Parameters

pvInstance – [in] Instance handle

Returns

Current link mode.

2.15 pfEcLinkIoctl

typedef EC_T_DWORD (***PF_EcLinkIoctl**)(EC_T_VOID *pvInstance, EC_T_DWORD dwCode, EC_T_LINK_IOCTLPARMS *pParms)

Multi Purpose LinkLayer IOCTL.

Parameters

- **pvInstance** – [in] Instance handle
- **dwCode** – [in] Control code
- **pParms** – [in/out] Pointer to the IOCTL parameters

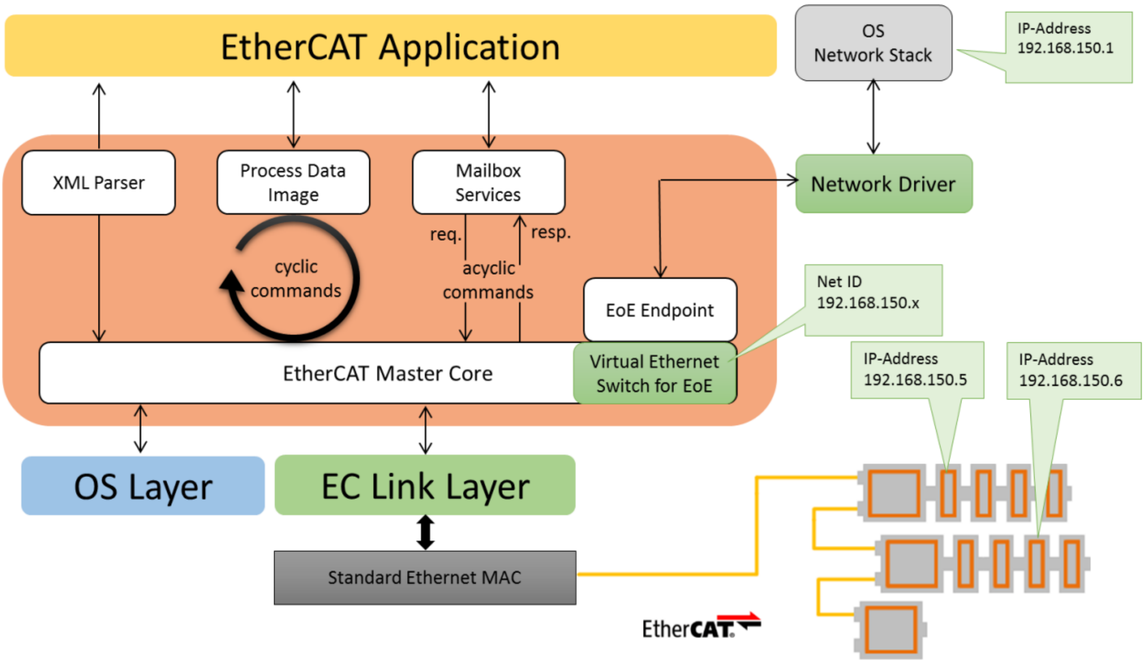
Returns

EC_E_NOERROR or error code

3 TAP devices

TAP devices are virtual network devices. Being network devices supported entirely in software, they differ from ordinary network devices which are backed up by hardware network adapters. Packets sent by an operating system via a TAP device are delivered to a user-space program which attaches itself to the device. A user-space program may also pass packets into a TAP device. In this case the TAP device "injects" these packets to the operating-system network stack thus emulating their reception from an external source.

3.1 Overview



3.2 Integration in the EC-Master application

1. Add `taped.cpp` to your project
2. Include `taped.h` into your EC-Master application
3. **Start the EoE endpoint driver: Add the following code once for example in myAppSetup():**

```
TapEdParams_t tapedParams;
TapEdInitParams(&tapedParams);
strcpy(tapedParams.IfaceName, "tap0");
tapedParams.CreateTxEvent = true;
tapedParams.MasterInstanceID = 0; // 0 first, 1 2nd,
TapEdHandle_t S_hTapDrv = TapEdInstall(&tapedParams);
//else S_hTapDrv  must be initialized with zero
```

4. **In any cyclic loop trigger the send thread with:**

```
TapEdTriggerTxEvent(S_hTapDrv);
```

For optimal performance, it is recommended, but not required, to make this call in the EtherCAT job task (e.g. in tEcJobTask()) after cyclic frames are received from the EtherCAT bus.

5. **Stop EoE endpoint driver with:**

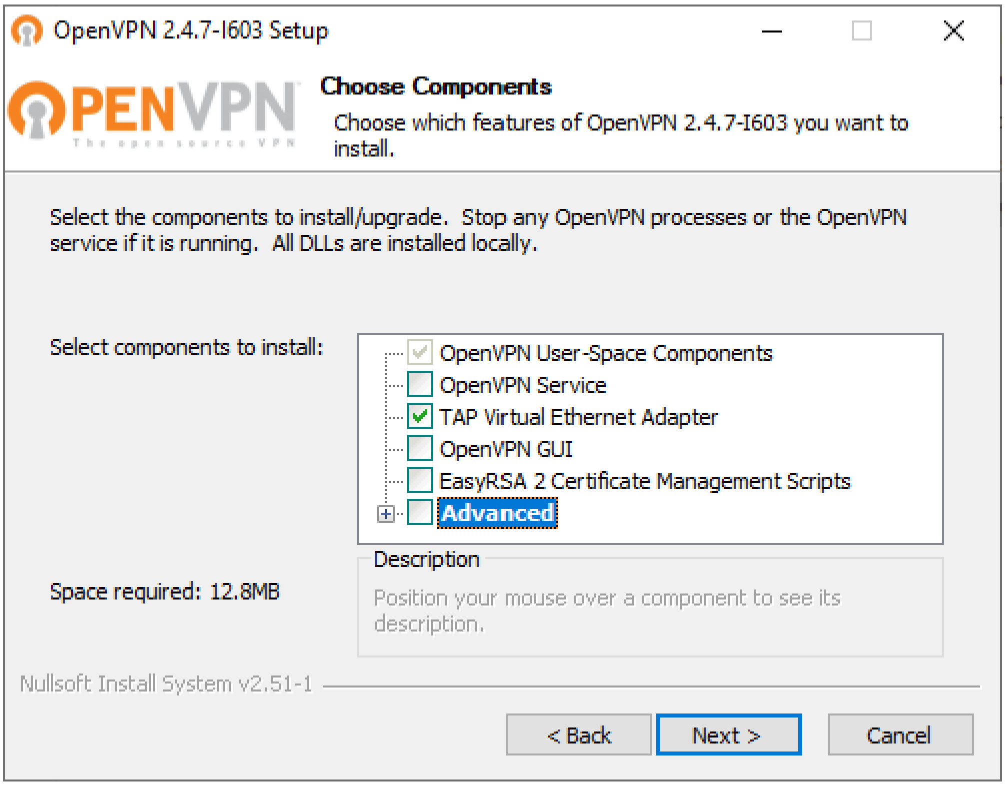
```
TapEdUninstall(S_hTapDrv);
S_hTapDrv = EC_NULL;
```

3.3 Windows

1. Download and install the TAP-Windows driver from OpenVPN (https://openvpn.net/community-downloads/).

Note: The OpenVPN Service or traffic encryption is not related to the EC-EoE-EndPoint. Only the *TAP Virtual Ethernet Adpater* component is required (https://github.com/OpenVPN/tap-windows6/releases).

2. Configure the interface



OpenVPN 2.4.7-I603 Setup

Choose Components

Choose which features of OpenVPN 2.4.7-I603 you want to install.

Select the components to install/upgrade. Stop any OpenVPN processes or the OpenVPN service if it is running. All DLLs are installed locally.

Select components to install:

☑ OpenVPN User-Space Components
□ OpenVPN Service
☑ TAP Virtual Ethernet Adapter
□ OpenVPN GUI
□ EasyRSA 2 Certificate Management Scripts
□ Advanced

Space required: 12.8MB

Description

Position your mouse over a component to see its description.

The EoE devices and the TAP adapter must be **in the same IPv4 subnet** and it must be **independent of the other network addresses** used on the EC-EoE-EndPoint system.

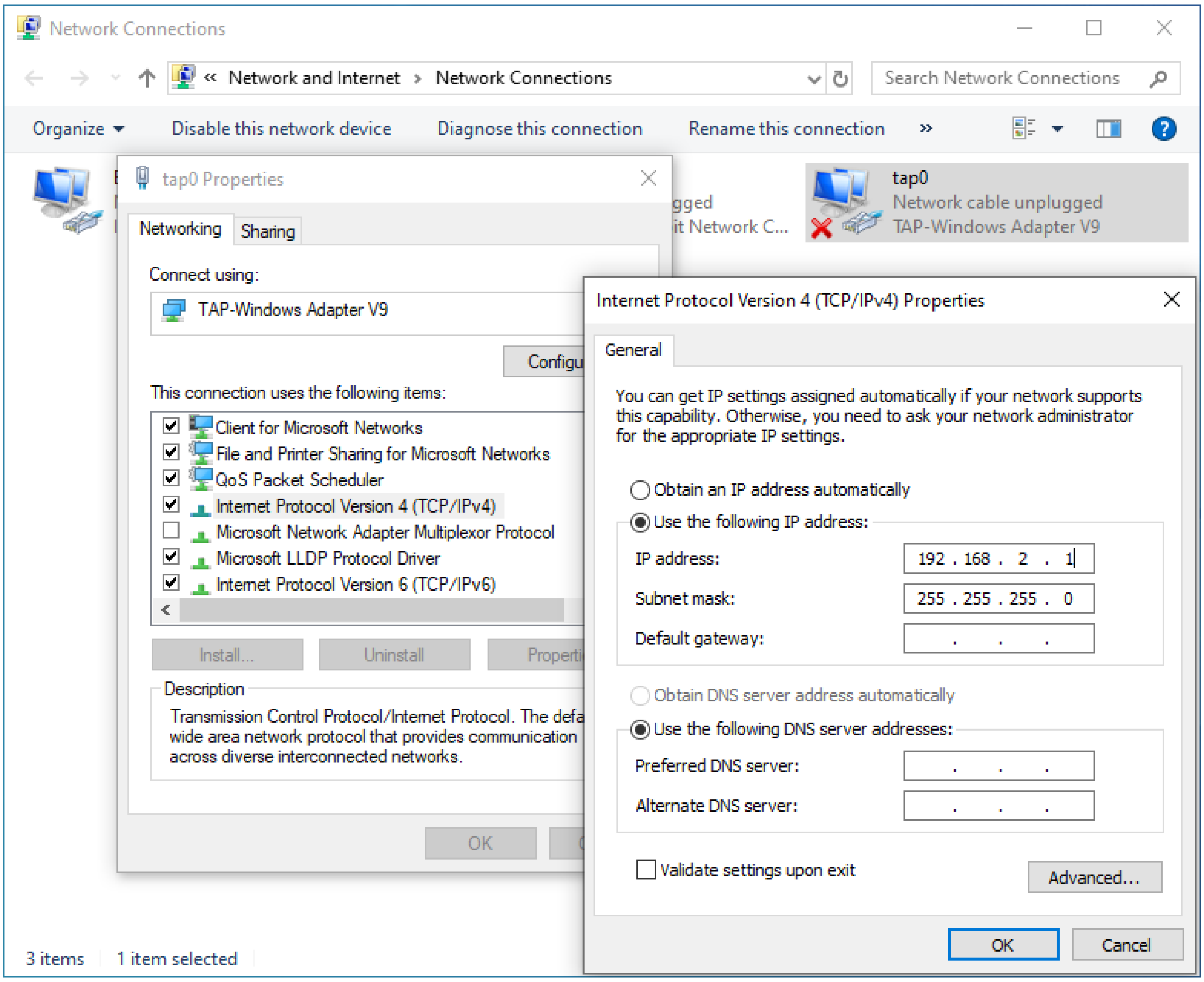
Example of independent subnets for LAN and EoE:

```
LAN: 192.168.0.1, netmask 255.255.255.0 (192.168.0.x)
EoE: 192.168.2.1, netmask 255.255.255.0 (192.168.2.x)
```

The command *ipconfig /all* shows the ip addresses and netmasks of the networks. The EoE network must be independent from the other network addresses.

The IPv4 addresses of the EoE devices are manually assigned in the EtherCAT network configuration using EC-Engineer or another EtherCAT configuration tool and are finally part of the ENI file loaded at EC-Master.

The IPv4 address for EoE must be assigned at the TAP adapter:



tap0 Properties

Networking | Sharing

Connect using:

TAP-Windows Adapter V9

This connection uses the following items:

- ☑ Client for Microsoft Networks
- ☑ File and Printer Sharing for Microsoft Networks
- ☑ QoS Packet Scheduler
- ☑ Internet Protocol Version 4 (TCP/IPv4)
- □ Microsoft Network Adapter Multiplexor Protocol
- ☑ Microsoft LLDP Protocol Driver
- ☑ Internet Protocol Version 6 (TCP/IPv6)

Description

Transmission Control Protocol/Internet Protocol. The defa wide area network protocol that provides communication across diverse interconnected networks.

Internet Protocol Version 4 (TCP/IPv4) Properties

General

You can get IP settings assigned automatically if your network supports this capability. Otherwise, you need to ask your network administrator for the appropriate IP settings.

○ Obtain an IP address automatically

● Use the following IP address:

IP address: 192 . 168 . 2 . 1

Subnet mask: 255 . 255 . 255 . 0

Default gateway: . . .

○ Obtain DNS server address automatically

● Use the following DNS server addresses:

Preferred DNS server: . . .

Alternate DNS server: . . .

□ Validate settings upon exit

Note: Disabling and re-enabling the “NonAdmin Access” on the TAP driver can sometimes fix communication issues.

---

3. Start EC-Master application

   Check if there are two additional threads `tTapTx` and `tTapRx`.

4. The EoE endpoint driver should work now. Try to ping your EoE device.

3.4 Linux

1. **Create the TAP interface. On your shell type:**

   ```
   tunctl -t tap0
   ```

   If tunctl is not installed, e.g. for Ubuntu you can type:

   ```
   apt-get install uml-utilities
   ```

2. **Configure the interface. E.g. on your shell type:**

   ```
   ip link set tap0 up
   ip addr add 10.0.0.2/24 dev tap0
   ```

3. **Start the EC-Master application**
   Check if there are two additional threads `tTapTx` and `tTapRx`. E.g. with:

   ```
   ps -eL -o pid,tid,rtprio,pri,class,cmd,comm,psr,%cpu
   ```

4. The EoE endpoint driver should work now. Try to ping your EoE device.

3.5 Windows-CE

1. Modify your BSP
   - Put the NDIS/tap folder inside your CE-BSP. E.g. in `[WINCE700]/platform/CeWin/SRC/DRIVERS/tap`
   - Add tap directory to your “dirs” file: DIRS=tap
   - Modify your `.bib` file. E.g. `platform.bib: tap-ce.dll $(_FLATRELEASEDIR)/tap-ce.dll` NK SHK
   - Modify your `.reg` file and setup an IP address. Please see content of `NDIS-TAP.config`
2. Integrate `taped.c` and `taped.h` as decribed above

   Note: Please use TAP0: as interface name

   ```
   wcscpy (tapedParams.IfaceName, L"TAP0:");
   ```

3. Start EC-Master application

   Check if there are two additional threads `tTapTx` and `tTapRx`.
4. The EoE endpoint driver should work now. Try to ping your EoE device.