

USB CDC/ECM Class Driver for Windows CE

Host-side Device Driver

Reference Manual

Version 1.10

September 21, 2010

Thesycon® Systemsoftware & Consulting GmbH
Werner-von-Siemens-Str. 2 · D-98693 Ilmenau · GERMANY

Tel: +49 3677 / 8462-0

Fax: +49 3677 / 8462-18

<http://www.thesycon.de>

Contents

Table of contents	4
1 Introduction	7
2 Overview	8
2.1 Architecture	8
2.2 Features	9
2.3 Supported Platforms	9
2.3.1 Windows CE Versions	9
2.3.2 Windows CE SDKs and CPU Architectures	10
2.3.3 Windows CE 6	10
2.4 Support for USB Host Controllers	10
3 USB Device Requirements	11
3.1 USB Descriptors	11
3.1.1 USB CDC/ECM Class Compliant Descriptors	11
3.1.2 Interface Association Descriptor (IAD)	11
3.1.3 Union Interface Functional Descriptor	11
3.1.4 Ethernet Networking Functional Descriptor	11
3.1.5 Composite Devices	12
3.2 USB Data Pipes	12
3.2.1 Bulk IN Pipe Behavior	12
3.2.2 Bulk OUT Pipe Behavior	13
4 Driver Installation and Uninstallation	14
4.1 CAB File Installation	14
4.1.1 Installing USB CDC/ECM	14
4.1.2 Uninstalling USB CDC/ECM	14
4.2 Platform Integration	14
4.3 Manual Installation	14
4.4 Registry Samples	14
5 Driver Configuration	16
5.1 CAB File Sample Projects	16
5.2 USB Bus Driver Loading Mechanism	16

5.2.1	Registration in Case of Class Compliant Descriptors	16
5.3	Driver Configuration Parameters	17
5.3.1	RxBuffers	17
5.3.2	TxBuffers	17
5.3.3	DefaultMediaState	17
5.3.4	OneByteTermination	18
5.4	Network Configuration	18
5.4.1	NDIS Miniport Registry Key	18
5.4.2	NDIS Adapter Instance Registry Key	19
5.4.3	TCP/IP Settings for the NDIS Adapter	19
6	Source Code Package	20
6.1	Build Instructions	20
7	Debugging Support	21
7.1	Enable Debug Traces	21

References

- [1] Microsoft Developer Network (MSDN) Library,
<http://msdn.microsoft.com/library/>
- [2] Windows Driver Development Kit,
<http://msdn.microsoft.com/library/>
- [3] Windows Platform SDK,
<http://msdn.microsoft.com/library/>

1 Introduction

The CDC/ECM class driver for Windows CE and Windows Mobile is a driver for USB devices which are compliant to the Communication Device Class (CDC), sub class Ethernet Control Model (ECM) defined by the USB Implementers Forum. For each CDC/ECM device instance the driver creates a network adapter.

The driver supports devices which are fully compliant to the CDC/ECM specification and devices which implement a subset of this specification only. See section 3 on page 11 for details about the supported protocols. The driver can be used with both USB 1.1 and USB 2.0 devices which operate at full speed or high speed.

This document describes the architecture and the features of the CDC/ECM class driver. Furthermore, it includes instructions for installing and using the driver.

The reader of this document is assumed to be familiar with the specification of the Universal Serial Bus (USB) Version 1.1 and 2.0, the CDC/ECM specification and with common aspects of Win32-based application programming.

2 Overview

2.1 Architecture

Figure 1 shows the USB driver stack of the Windows CE operating system with the USB CDC/ECM class driver.

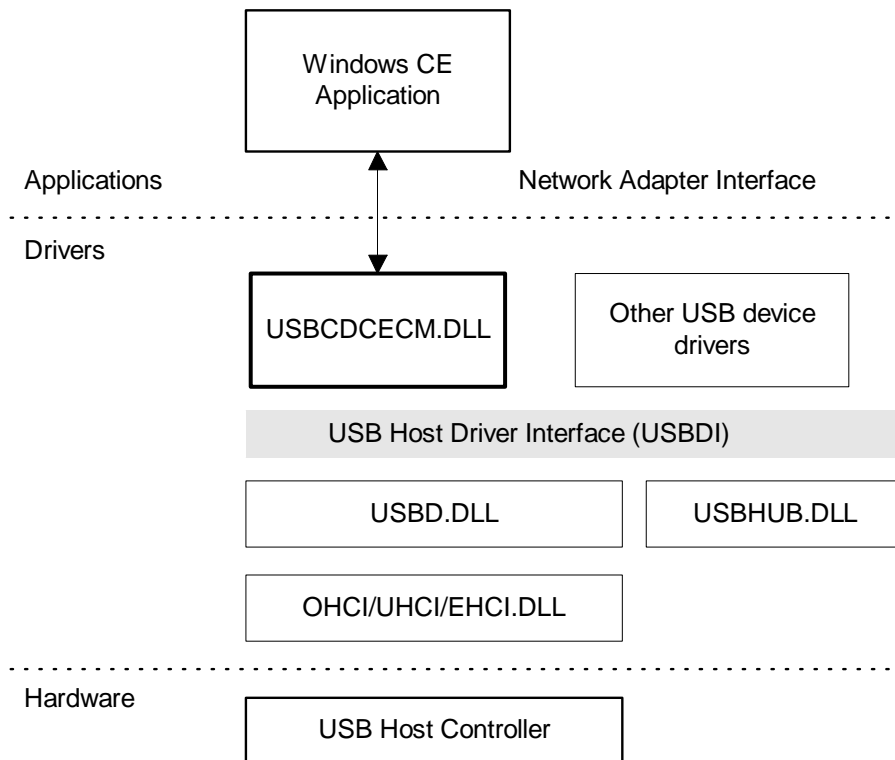


Figure 1: USB Driver Stack

The following modules are shown in Figure 1:

- USB Host Controller is the hardware component that controls the Universal Serial Bus. It also contains the USB Root Hub. There are two standard implementations of the host controller that support USB full speed: Open Host Controller (OHC) and Universal Host Controller (UHC). There is one standard implementation of the host controller that supports USB high speed: Enhanced Host Controller (EHC). Except these standard implementations third party USB Host Controller driver are available.
- UHCI.DLL, OHCI.DLL, OHCI2.DLL and EHCI.DLL are host controller drivers provided by Microsoft.
- USB.DLL is the USB bus driver that controls and manages all devices connected to the USB. It is provided by Microsoft as part of the operating system.
- USBHUB.DLL is the USB hub driver provided by Microsoft. It is responsible for managing and controlling USB Hubs.

- USBCDCECM.DLL is the USB CDC/ECM class driver. It creates standard Windows network adapter.

The software interface that is provided by the operating system for use by USB device drivers is called USB Host Driver Interface (USBHI). It is a function pointer interface exported by the USBH.

2.2 Features

The features provided by the USB CDC/ECM class driver are summarized below.

- Provides a network adapter for the USB device.
- Supports devices that are compliant with the CDC/ECM class specification.
- Supports proprietary variants of the CDC/ECM protocol to work with devices with a simplified or incomplete implementation of that protocol.
- Multiple USB devices can be connected to the driver concurrently. One network adapter will be created per device.
- Devices with multiple CDC/ECM interfaces (composite devices) are fully supported. One network adapter will be created per CDC/ECM instance.
- Composite devices which combine one or more CDC/ECM interfaces with other functionality (e.g. Mass Storage class) are fully supported.
- Supports Plug&Play. Network adapters will be created and destroyed dynamically.
- Supports power management. The device can be suspended if unused.
- Supports USB 1.1 and USB 2.0 at full and high speed.
- Complies with the Windows CE driver model.

2.3 Supported Platforms

2.3.1 Windows CE Versions

The CDC/ECM driver supports the following operating system versions:

- Windows CE 5.0
- Windows CE 6.0
- Windows Mobile 5
- Windows Mobile 6

2.3.2 Windows CE SDKs and CPU Architectures

The driver is built against an SDK available from Microsoft. Each SDK supports a specific set of CPU architectures as shown below. For each particular combination of SDK and CPU architecture a pre-compiled driver executable (DLL) is shipped by Thesycon. The name of the subdirectory in which the DLL is contained identifies the corresponding combination of SDK and CPU architecture.

Supported SDKs:

- Windows Mobile 5.0 Pocket PC SDK
CPU architectures: ARMV4I
- Windows CE 5.0 Standard SDK (STANDARDSDK_500)
CPU architectures: ARMV4I, MIPSII, MIPSII_FP, MIPSIV, MIPSIV_FP, SH4, x86
- Windows Mobile 6 Professional SDK
CPU architectures: ARMV4I

2.3.3 Windows CE 6

For Windows CE 6 no standard SDK is available from Microsoft. For this reason Thesycon cannot ship a pre-compiled driver executable for every target architecture. To use the driver on Windows CE 6 several approaches are possible as described below.

If the target system runs Windows CE 6 on ARMV4I (Intel XScale) then the "Windows Mobile 6 Professional SDK" variant can be used.

If the target system runs Windows CE 6 on another CPU then the "STANDARDSDK_500" variant that corresponds to the CPU type can be used. However, there is no guarantee that this will work on the given system, but it's worth a try.

If none of the above approaches work then the driver needs to be build against a Windows CE 6 SDK provided by the vendor of the target hardware platform. Some hardware vendors or OEMs provide such platform-specific SDKs. Of course, a source code license is required to build the driver (see also chapter 6). Alternatively, contact Thesycon to request support for a platform-specific SDK.

2.4 Support for USB Host Controllers

With Windows CE Microsoft provides drivers for standard USB host controllers such as Universal Host Controller (UHC), Open Host Controller (OHC) and Enhanced Host Controller (EHC). The USB CDC/ECM class driver is based on those driver stacks. Some host controllers provided by third parties (e.g. USB host CF plug-in cards) are shipped with a specific host driver stack. In this case the USB CDC/ECM class driver will work only if this host driver stack is compatible with the USB host programming interfaces defined by Microsoft. However, Thesycon guarantees the driver's functionality only if it is used in conjunction with Microsoft USB host drivers.

3 USB Device Requirements

3.1 USB Descriptors

The USB CDC/ECM class specification defines the USB descriptor layout to be implemented by a device. In short, it defines a USB interface descriptor for the control interface and a USB interface descriptor for the data interface. However, because Windows XP has problems with handling this descriptor layout correctly many devices use a different layout which is not compliant to the specification. To take account of this the USB CDC/ECM class driver supports various constellations of USB descriptors as described in the subsequent sections.

3.1.1 USB CDC/ECM Class Compliant Descriptors

Each CDC/ECM instance is comprised of two USB interface descriptors: control interface and data interface descriptor. The descriptors need to contain the following values:

Control interface:

```
bInterfaceClass = 2  
bInterfaceSubClass = 6
```

Data interface:

```
bInterfaceClass = 10 (0xA)  
bInterfaceSubClass = 0
```

The driver does not check the value of the `bInterfaceProtocol` field in any descriptor.

The control interface contains one interrupt IN endpoint. The data interface contains one bulk IN endpoint and one bulk OUT endpoint. The driver accepts an interface only if its endpoint layout is correct.

3.1.2 Interface Association Descriptor (IAD)

An Interface Association Descriptor (IAD) can be used to group the control and data interface of a given CDC/ECM instance. Use of this descriptor is optional. The USB CDC/ECM class driver does not evaluate an IAD in any way.

3.1.3 Union Interface Functional Descriptor

An Union Interface Functional Descriptor is used to group interfaces of a CDC instance. The descriptor is defined by the USB CDC specification. The USB CDC/ECM class driver requires that descriptor.

3.1.4 Ethernet Networking Functional Descriptor

An Ethernet Networking Functional Descriptor provides information such as MAC address for a CDC/ECM instance. The descriptor is defined by the CDC/ECM specification. The USB CDC/ECM class driver requires that descriptor.

3.1.5 Composite Devices

A composite device consists of one or more CDC/ECM instances and optionally of other logical functions such as USB mass storage. Composite devices are fully supported by the USB CDC/ECM class driver. The driver creates network adapter instance for each CDC/ECM instance the device exposes.

On Windows CE the order of the USB interfaces is important for loading the driver. For example the USB interface 0 is a printer interface and no driver is found for that interface, the drivers for all other interfaces are not loaded. Possible solutions are to install a driver for the interface or rearrange the USB interface numbers. If no Windows CE driver is available for a interface and the interface is not needed, a dummy driver could be installed on the USB interface.

3.2 USB Data Pipes

The Communication Device Class (CDC) specification defines a data interface which uses a bulk IN and a bulk OUT pipe for bidirectional data transfer. However, the specification does not define the exact behavior of the device and the host with respect to USB packet sizes. In particular, the handling of short packets (and zero packets) is not defined although this is an important part of the communication protocol used between driver and device. This section discusses these protocol details and explains how the driver needs to be configured to work correctly with a given device.

Note that the examples given in the discussion below assume a full speed USB device with `MaxPacketSize = 64` bytes for each bulk endpoint. The discussion does also apply to a high speed device which uses `MaxPacketSize = 512` bytes.

3.2.1 Bulk IN Pipe Behavior

When the CDC/ECM class driver is loaded, a bunch of read buffers is submitted to the IN pipe. As soon as read buffers are available in the Microsoft bus driver the host controller starts to issue IN tokens to the endpoint. If the device answers with a data packet then the data is placed into the first read buffer. Subsequent packets are placed contiguously into this buffer. Each received packet possibly completes the buffer and returns it to the CDC/ECM class driver. A buffer is returned if one of two conditions hold true:

- Either a short packet is received, or
- the buffer is filled completely.

A short packet is defined as a packet with a length less than the endpoint's `MaxPacketSize` parameter as reported in the endpoint descriptor. A special case of a short packet is a zero packet which does not contain any data.

In the first case the read buffer is returned after the data of the short packet has been placed into the buffer. Thus, in this buffer the driver gets the data received up to (and including) the short packet. This can be much less than the buffer's size.

The latter case (buffer completely filled) occurs if the device sends a sequence of packets of `MaxPacketSize` bytes each. Because there is no short packet the buffer will be filled up completely and then returned to the driver.

If none of the above conditions occur then the read buffer will be kept by the bus driver and will wait for more data. The data bytes already contained in this buffer cannot yet be processed by the driver and by an application. This may lead to unexpected delays in the data transfer.

3.2.2 Bulk OUT Pipe Behavior

When the CDC/ECM class driver is loaded and network packets are submitted through the virtual network adapter, the driver submits write buffers to the OUT pipe. The bus driver will split each buffer into USB packets and transfer these packets to the device. The maximum length of each packet is determined by the `MaxPacketSize` parameter reported in the endpoint descriptor.

If a write buffer contains a number of data bytes that is not an integral multiple of `MaxPacketSize` then the last packet sent will contain the remaining bytes. So the last packet will be a short packet. If a write buffer contains a number of data bytes that is an integral multiple of `MaxPacketSize` then the CDC/ECM class driver will send an extra zero packet after the transmission of the buffer.

In other words, in OUT direction the driver ensures that each transfer is terminated by a short packet or zero packet. The device needs to be able to handle the zero packets properly.

The driver provides a configuration parameter `OneByteTermination` which can be used to send an extra byte to avoid zero packet generation for the OUT endpoint. See section 5.3.4 for details.

4 Driver Installation and Uninstallation

This section discusses topics relating to the installation and un-installation of the USB CDC/ECM device driver.

There are different ways to install the USB CDC/ECM driver. The following methods are possible:

- **CAB File Installation** - Should be used if your platform supports cab files and you cannot change the platform image.
- **Platform Integration** - Should be used if you build your own platform image.
- **Manual Installation** - Fallback if no other method works.

4.1 CAB File Installation

4.1.1 Installing USB CDC/ECM

Copy the .cab file for your platform to the Windows CE target system. Execute the .cab file on the Windows CE system to install the driver. Now the USB CDC/ECM device can be connected to the Windows CE platform.

4.1.2 Uninstalling USB CDC/ECM

The USB CDC/ECM driver can be uninstalled via Settings - System - Remove Programs. Select your driver package and click Remove.

4.2 Platform Integration

The integration of a driver into a platform image is described in the Microsoft Platform Builder documentation in the chapter "How to Add a Device Driver to the Catalog". The USB CDC/ECM driver binary usbcdcecm.dll file must be added to the platform image. Normally usbcdcecm.dll should be in the \Windows directory. Additional registry settings are necessary to load the USB CDC/ECM class driver when the USB device is connected. These registry entries are described in chapter 5.

4.3 Manual Installation

Copy the usbcdcecm.dll file in the \Windows directory and edit the registry to load the driver. The section 5.2 describes the settings that are necessary to load the USB CDC/ECM class driver if the USB device is connected. Further settings are described in chapter 5.

4.4 Registry Samples

For a USB device with class compliant descriptors:

```
[HKEY_LOCAL_MACHINE\Drivers\USB\LoadClients]
  [Default]
    [Default]
      [10_0]
        [USBCDCECM_ClassDriver]
          REG_SZ "Dll"="usbcdcecm.dll"
      [2_6]
        [USBCDC_ClassDriver]
          REG_SZ "Dll"="usbcdcecm.dll"
```

For a USB device with one vendor specific device (VID:0x152A PID:0x100):

```
[HKEY_LOCAL_MACHINE\Drivers\USB\LoadClients]
  [5418_256]
    [Default]
      [Default]
        [USBCDCECM_ClassDriver]
          REG_SZ "Dll"="usbcdcecm.dll"
```

5 Driver Configuration

In this section various registry settings required to load and configure the driver are discussed. It is recommended to create these settings by means of a .cab file which is installed on the target machine.

5.1 CAB File Sample Projects

To install the driver, a .cab should be created. This is done with Visual Studio 2005 by creating a new project of type "Smart Device CAB Project". The registry settings to be made by a .cab file are described in the following sections.

Together with the driver Thesycon provides several sample projects which can be used as a starting point.

- `usbcdcecm_ce_inst_cls.vddproj`
This sample shows how to register the driver for USB CDC/ECM class-compliant USB interfaces. See section 5.2.1.
- `usbcdcecm_ce_inst_vidpid.vddproj`
This sample shows how to register the driver for a device using its specific vendor ID (VID) and product ID (PID) values.

5.2 USB Bus Driver Loading Mechanism

In this section a summary of the driver matching and loading mechanism as defined by Microsoft is given. For a more detailed description refer to the Windows CE documentation. To find the topic in the documentation, checkout <http://msdn.microsoft.com/library> and search for "USB Host Controller Driver Registry Settings".

To load a USB driver for a device or interface, appropriate registry settings need to be created under the following key:

```
HKEY_LOCAL_MACHINE\Drivers\USB\LoadClients
```

The USB bus driver uses a matching algorithm which is based on checks at three levels. For each level a sub key exists in the registry. At the first level the vendor ID (VID), product ID (PID) and optionally the revision code from the USB device descriptor is checked. At the second level the class code, subclass code and optionally the protocol code from the USB device descriptor is checked. At the third level the class code, subclass code and optionally the protocol code from the USB interface descriptor is checked. If a level is not used then the sub key name is set to `Default` to match with any values.

5.2.1 Registration in Case of Class Compliant Descriptors

The following example shows how the USB CDC/ECM class driver needs to be registered in case the device exposes CDC/ECM class compliant USB descriptors. For more information about USB descriptor layout variants, see also section 3.1.

```
[HKEY_LOCAL_MACHINE\Drivers\USB\LoadClients]
  [Default]
    [Default]
      [10_0]
        [USBCDCECM_ClassDriver]
          REG_SZ "Dll"="usbcdcecm.dll"
      [2_6]
        [USBCDCECM_ClassDriver]
          REG_SZ "Dll"="usbcdcecm.dll"
```

In this case the driver matches with any VID/PID (first level) and with any class/subclass code (second level) present in the USB device descriptor. At the USB interface level (third level) the driver matches with class/subclass code = 10/0 which corresponds to the CDC data interface and with class/subclass code = 2/6 which corresponds to the CDC/ECM control interface. See also section 3.1.1.

The Dll value specifies the driver executable to be loaded if an interface in the device's USB configuration descriptor matches with the specified class/subclass codes.

5.3 Driver Configuration Parameters

The USB CDC/ECM class driver supports a set of configuration parameters to modify the setup of the data path (USB buffer sizing) and to fine-tune the behavior of the driver. The configuration parameters described below are to be specified in the Params key of a given NDIS instance.

All parameters are optional. If a configuration parameter does not exist in the registry the driver uses an internal default value.

5.3.1 RxBuffers

Default: 8 buffers

Specifies the number of USB transfer buffers used for the bulk IN pipe.

5.3.2 TxBuffers

Default: 8 buffers

Specifies the number of USB transfer buffers used for the bulk OUT pipe.

5.3.3 DefaultMediaState

Default: 0 (disconnected)

If this parameter is 1 the driver reports the media state as connected. The PC can send immediately requests to the device. If this parameter is set to 0 the driver waits on a notification from the device that the media state is changed to connected.

5.3.4 OneByteTermination

Default: 0 (off)

If this parameter is 1 the driver terminates a packet that can be divided by the FIFO size without rest with an additional byte. If this parameter is zero the driver terminates such a packet with a zero length packet. Set this parameter to 1 if your hardware cannot handle zero length packets.

Note: Linux use the same algorithm.

5.4 Network Configuration

To configure the NDIS adapter exported by the USB CDC/ECM class driver three registry locations are important.

- [HKEY_LOCAL_MACHINE\Comm\]
This is a global registry key for the NDIS miniport driver. The registry key exists only once per driver. <MiniportDriverName> is by default set to the Dll name without file extension, normally usbcdcecm.
- [HKEY_LOCAL_MACHINE\Comm\\Parms]
This is a registry key created per NDIS adapter instance. Normally a device have one NDIS adapter instance. In this case a NDIS adapter instance is created for each connected USB CDC/ECM device. <AdapterInstanceName> is by default set to the Dll name without file extension plus an index beginning with 0 for the first instance, e.g. usbcdcecm0, usbcdcecm1, ...
- [HKEY_LOCAL_MACHINE\Comm\\Parms\TcpIp]
This is a subkey of the former NDIS adapter instance key. In this key TCP/IP settings are made for the adapter.

The first two registry settings are written automatically by the USB CDC/ECM class driver. The third registry key for TCP/IP settings is optional. If it is not created Windows CE tries to get an IP address by a DHCP request. If no DHCP server is reachable, Windows CE uses Auto-IP to assign an IP address for the NDIS adapter. The key must be defined by the user for a static IP. The following part shows some examples for the three registry keys.

5.4.1 NDIS Miniport Registry Key

The following NDIS miniport registry registration is done by the USB CDC/ECM class driver automatically.

```
[HKEY_LOCAL_MACHINE\Comm\usbcdcecm]
  "DisplayName"="USB CDC/ECM NDIS"
  "Group"="NDIS"
  "ImagePath"="usbcdcecm.dll"
```

5.4.2 NDIS Adapter Instance Registry Key

The following NDIS adapter instance registry registration is done by the USB CDC/ECM class driver automatically.

```
[HKEY_LOCAL_MACHINE\Comm\usbcdcecm0]
  "BusNumber"=dword:0
  "BusType"=dword:0
```

5.4.3 TCP/IP Settings for the NDIS Adapter

The TCP/IP configuration for the first NDIS adapter is made under the following registry location:

```
; TCP/IP settings
[HKEY_LOCAL_MACHINE\Comm\usbcdcecm0\Parms\TcpIp]
  ; This should be MULTI_SZ
  ; "DefaultGateway"="10.1.1.1"
  ; Domain Name Service
  ; "DNS"="10.1.1.1"
  ; This should be SZ... If null it means use LAN, else WAN and Interface.
  "LLInterface"=""
  ; Use zero for broadcast address? (or 255.255.255.255)
  "UseZeroBroadcast"=dword:0
  ; Thus should be MULTI_SZ, the IP address list
  "IpAddress"="10.1.1.3"
  ; This should be MULTI_SZ, the subnet masks for the above IP addresses
  "Subnetmask"="255.255.255.0"
  "EnableDHCP"=dword:0
```

The `usbcdcecminstall_ce` is a sample project that shows how to set the registry keys to register the driver and optionally set a static IP address.

6 Source Code Package

The source code is not part of the normal distribution. The source code can be licensed separately from Thesycon.

6.1 Build Instructions

- Execute the source package executable to unpack the source code files.
- Visual Studio 2005 is needed to build the driver and the CAB file installer project.
- Install the SDK for your Windows CE Platform. Microsoft provides SDK's for some Platforms. For Windows CE 6 no Standard SDK is available from Microsoft. The following SDK's have been tested with the USB CDC/ECM driver:
 - Windows CE 5.0 Standard SDK
 - Windows Mobile 5.0 SDK
 - Windows Mobile 6 SDK
- The source code package contains a solution for Visual Studio 2005. This solution can be used to edit the source code and to build the driver.

7 Debugging Support

7.1 Enable Debug Traces

The driver package contains a folder debug with the debug version of the driver. The debug version of the driver generates text messages into the file `usbcdcecm_XXXXXXXXX.log` in the root directory of the Windows CE device. These messages can help to analyze problems.

The `usbcdcecm_XXXXXXXXX.log` file is created when the driver is loaded. If the driver is not loaded, the reason could be a wrong driver binary for the Windows CE platform or the driver registry registration is incorrect.

