

USB CDC/ACM Class Driver for Windows CE

Host-side Device Driver

Reference Manual

Version 1.40

December 11, 2009

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>

Copyright (c) 2007-2009 by Thesycon Systemsoftware & Consulting GmbH

All Rights Reserved

Disclaimer

Information in this document is subject to change without notice. No part of this manual may be reproduced, stored in a retrieval system, or transmitted in any form or by any means electronic or mechanical, including photocopying and recording for any purpose other than the purchaser's personal use, without prior written permission from Thesycon Systemsoftware & Consulting GmbH. The software described in this document is furnished under the software license agreement distributed with the product. The software may be used or copied only in accordance with the terms of the license.

Trademarks

The following trade names are referenced throughout this manual:

Microsoft, Windows, Win32, Windows NT, Windows XP, Windows Vista, Visual C++ and Windows CE are either trademarks or registered trademarks of Microsoft Corporation.

Other brand and product names are trademarks or registered trademarks of their respective holders.

Contents

Table of Contents	6
References	7
1 Introduction	9
2 Overview	10
2.1 Architecture	10
2.2 Features	11
2.3 Supported Platforms	11
2.3.1 Windows CE Versions	11
2.3.2 Windows CE SDKs and CPU Architectures	12
2.3.3 Windows CE 6	12
2.4 Support for USB Host Controllers	12
3 USB Device Requirements	14
3.1 USB Descriptors	14
3.1.1 USB CDC/ACM Class Compliant Descriptors	14
3.1.2 Non-class Compliant Descriptors	14
3.1.3 Interface Association Descriptor (IAD)	14
3.1.4 Composite Devices	15
3.2 USB Class Requests	15
3.3 USB Data Pipes	15
3.3.1 Bulk IN Pipe Behavior	15
3.3.2 Bulk IN Pipe Configuration	16
3.3.3 Bulk OUT Pipe Behavior	17
4 Driver Installation and Uninstallation	18
4.1 CAB File Installation	18
4.1.1 Installing USB CDC/ACM	18
4.1.2 Uninstalling USB CDC/ACM	18
4.2 Platform Integration	18
4.3 Manual Installation	18
4.4 usb_ce_install	18
4.5 Registry Samples	19

5	Driver Configuration	21
5.1	CAB File Sample Projects	21
5.2	USB Bus Driver Loading Mechanism	21
5.2.1	Registration in Case of Class Compliant Descriptors	22
5.2.2	Registration in Case of Non-class Compliant Descriptors	22
5.3	Driver-defined Registry Locations	22
5.3.1	Config Key	22
5.3.2	Template Key	24
5.4	COM Port Number Assignment	24
5.5	Modem Configuration	24
5.6	Driver Configuration Parameters	25
5.6.1	BufferCountIn	25
5.6.2	BufferCountOut	25
5.6.3	BufferSizeFactorIn	26
5.6.4	BufferSizeFactorOut	26
5.6.5	FifoSizeIn	26
5.6.6	FifoSizeOut	26
5.6.7	SendShortPacketsOut	26
5.6.8	SupportSetControlLineState	26
5.6.9	SupportSetLineCoding	27
5.6.10	SupportGetLineCoding	27
5.6.11	SupportSendBreak	27
5.6.12	PowerManagementLevel	27
5.7	Device Specific Parameters	28
5.7.1	IdleTimerInterval	28
5.7.2	SetRemoteWakeUp	28
5.7.3	RequestD3viaPM	28
5.8	Power Management	28
5.9	COM Port Enumeration	29
6	Source Code Package	30
6.1	Build Instructions	30
7	Debugging Support	31
7.1	Enable Debug Traces	31

References

- [1] Universal Serial Bus Specification 1.1,
<http://www.usb.org>
- [2] Universal Serial Bus Specification 2.0,
<http://www.usb.org>
- [3] USB Communications Device Class,
http://www.usb.org/developers/devclass_docs/
- [4] Microsoft Developer Network (MSDN) Library,
<http://msdn.microsoft.com/library/>
- [5] Windows Platform SDK,
<http://msdn.microsoft.com/library/>
- [6] Platform Builder for Windows CE 5.0 Help,
<http://msdn.microsoft.com/library/>

1 Introduction

The CDC/ACM 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 Abstract Control Model (ACM) defined by the USB Implementers Forum. For each CDC/ACM device instance the driver exposes a serial COM port and emulates the behavior of legacy (physical) COM ports. That means the driver supports the standard Win32 API calls typically used for serial communication, such as CreateFile, WriteFile, ReadFile, SetCommState, SetCommMask, WaitCommEvent, etc.

The driver supports devices which are fully compliant to the CDC/ACM specification and devices which implement a subset of this specification only. See section 3 on page 14 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/ACM 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/ACM 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/ACM class driver.

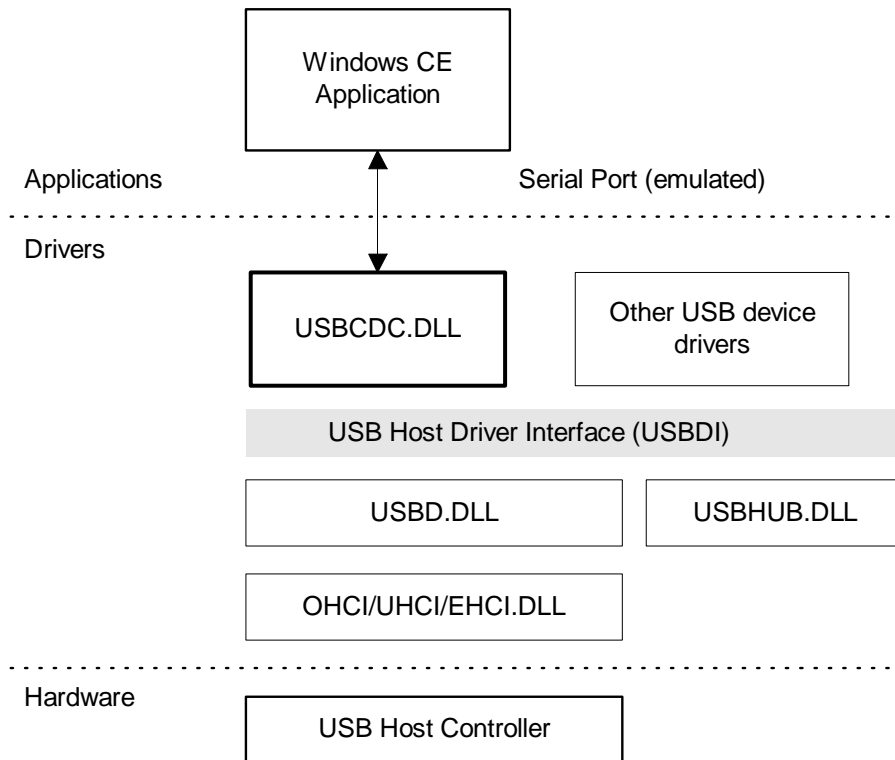


Figure 1: USB Driver Stack

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 D.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.
- USB HUB.DLL is the USB hub driver provided by Microsoft. It is responsible for managing and controlling USB Hubs.

- USB CDC.DLL is the USB CDC/ACM class driver. It emulates a serial COM port and supports the standard Win32 API class for COM ports.

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

2.2 Features

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

- Provides a serial COM port emulation for the USB device.
- Supports devices that are compliant with the CDC/ACM class specification.
- Supports proprietary variants of the CDC/ACM 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 COM port will be created per device.
- Devices with multiple CDC/ACM interfaces (composite devices) are fully supported. One COM port will be created per CDC/ACM interface.
- Composite devices which combine one or more CDC/ACM interfaces with other functionality (e.g. Mass Storage class) are fully supported.
- Supports Plug&Play. COM ports 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/ACM driver supports the following operating system versions:

- Windows CE 5.0
- Windows CE 6.0
- Windows Mobile 5
- Windows Mobile 6
- Windows Pocket PC 2003 on request
- Windows Smartphone 2003 on request

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:

- Pocket PC 2003 SDK (Pocket PC 2003) on request
CPU architectures: ARMV4
- Smartphone 2003 SDK (Smartphone 2003) on request
CPU architectures: ARMV4
- 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/ACM 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/ACM 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/ACM 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/ACM class driver supports various constellations of USB descriptors as described in the subsequent sections.

3.1.1 USB CDC/ACM Class Compliant Descriptors

Each CDC/ACM 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 = 2
```

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 Non-class Compliant Descriptors

The driver supports different variants of a non-complaint descriptor layout.

- All three endpoints (interrupt IN, bulk IN, bulk OUT) of a CDC/ACM instance are combined in one USB interface.
- A CDC/ACM instance consists of a bulk IN and bulk OUT endpoint only which are combined in one interface; there is no interrupt endpoint.

In each of these variants the values of the fields `bInterfaceClass`, `bInterfaceSubClass` and `bInterfaceProtocol` will be ignored by the driver.

3.1.3 Interface Association Descriptor (IAD)

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

3.1.4 Composite Devices

A composite device consists of one or more CDC/ACM instances and optionally of other logical functions such as USB mass storage. Composite devices are fully supported by the USB CDC/ACM class driver. The driver creates one serial COM port for each CDC/ACM 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 Class Requests

The USB CDC/ACM class driver uses the class requests specified in the USB CDC/ACM specification. The driver can be configured to use a reduced set of class requests to solve compatibility problems with some devices which do not implement all class requests or do not implement some requests correctly. Refer to section 5.6 for a description of the appropriate configuration options.

3.3 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.

3.3.1 Bulk IN Pipe Behavior

When the COM port is opened by an application the CDC/ACM class driver submits a bunch of read buffers to the IN pipe. The size of each of these buffers is an integral multiple of the corresponding endpoint's `MaxPacketSize` parameter (as indicated in the endpoint descriptor).

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/ACM 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.

3.3.2 Bulk IN Pipe Configuration

The read buffer size to be used by the CDC/ACM class driver can be adjusted by means of the configuration parameter `BufferSizeFactorIn` (5.6.3). See section 5.6 for a description of this parameter. By default, the read buffer size is set to a multiple of `MaxPacketSize` reported in the endpoint descriptor.

According to the discussion in section 3.3.1 above the buffer size needs to be configured properly to match the resulting behavior with the device implementation. There are two distinct cases which are discussed below.

Device implements short packet termination. The device terminates each data transfer with a short packet. If the length of the transfer was a multiple of `MaxPacketSize` then the device sends an extra zero packet. For example, if `MaxPacketSize` = 64 bytes and the device transferred a sequence of 128 bytes then it needs to send a subsequent zero packet. This ensures that the driver (and the application) will receive the 128 bytes sequence immediately.

This is the usual (and preferred) behavior. In this case, the read buffer size can be set to a multiple of `MaxPacketSize`. To achieve a good throughput, it is recommended to use a read buffer size of 16 times `MaxPacketSize` which is the default. See section 5.6 for more information.

Depending on the individual host controller type, with this configuration typically a throughput of 700.000... 800.000 bytes/sec can be achieved (assumed that `MaxPacketSize` = 64).

Device does not implement short packet termination. If the length of a data transfer was a multiple of `MaxPacketSize` then the device does not send an additional zero packet. This leads to unexpected delays in data reception if a large read buffer is used (see section 3.3.1 above for an explanation). For example, if `MaxPacketSize` = 64 bytes and the device transferred a sequence of 128 bytes then the read buffer contains these 128 bytes but will not be passed to the driver and the application for further processing.

To work around this, the read buffer size needs to be set to `MaxPacketSize`. To achieve this, the driver configuration parameter `BufferSizeFactorIn` (section 5.6.3) must be set to 1. The drawback is that throughput is very limited with this configuration. The impact depends on the individual host controller type but typically with this configuration there will be one IN transaction per millisecond only. So if `MaxPacketSize` = 64 bytes then this results in a transfer rate of 64.000 bytes/sec.

3.3.3 Bulk OUT Pipe Behavior

When the COM port is opened by an application and data is written to the port then the CDC/ACM class 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/ACM 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 `SendShortPacketsOut` which can be used to turn off the zero packet generation for the OUT endpoint. See section 5.6.7 for details. However, zero packet generation should be turned off only if the device is not able to handle those packets correctly.

4 Driver Installation and Uninstallation

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

There are different ways to install the USB CDC 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.
- **usb_ce_install** - Should be used for fast and simple device installation e.g., for testing.

4.1 CAB File Installation

4.1.1 Installing USB CDC/ACM

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/ACM device can be connected to the Windows CE platform.

4.1.2 Uninstalling USB CDC/ACM

The USB CDC/ACM 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/ACM driver binary usbcdd.dll file must be added to the platform image. Normally usbcdd.dll should be in the \Windows directory. Additional registry settings are necessary to load the USB CDC/ACM class driver when the USB device is connected. These registry entries are described in chapter 5.

4.3 Manual Installation

Copy the usbcdd.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/ACM class driver if the USB device is connected. Further settings are described in chapter 5.

4.4 usb_ce_install

This method is preferred for testing the driver. It is a fast and simple method to install the driver. First copy the usbcdd.dll file in the \Windows directory. Now create a file called usbcdd.cfg in the

same directory as the `usb_ce_install.exe` file on your target device. A sample for the `usbcdc.cfg` file is delivered in the driver package. The file contains information about your device e.g., VID and PID that needs to be modified to match your device. After that execute the `usb_ce_install.exe` to make the registry settings for loading the driver. The `usb_ce_install.exe` reads the `usbcdc.cfg` file and writes the settings that are necessary to load the driver in the registry.

The `usbcdc.cfg` file contains one or more sections. One section for a device is needed. To register the driver for different device add another section. A section starts with `[xxx]` there `xxx` can be defined by the customer. Each section contains required and optional parameters. The following parameters are valid.

- **Required: VID** Vendor ID of your USB device.
- **Required: PID** Product ID of your USB device.
- **Required: DLL** Name of the driver Dll copied to the \Windows directory.
- **Optional: GUID** Custom GUID for your device generated with GuidGen.
- **Optional: IFC** Interface index if the driver should be loaded on a USB interface.
- **Optional: IFC_Class** Interface class the driver should be loaded on.
- **Optional: IFC_SubClass** Interface subclass the driver should be loaded on.
- **Optional: IFC_Protocol** Interface protocol the driver should be loaded on.

4.5 Registry Samples

For a USB device with class compliant descriptors:

```
[HKEY_LOCAL_MACHINE\Drivers\USB\LoadClients]
  [Default]
    [Default]
      [10_0]
        [USBCDC_ClassDriver]
          REG_SZ "Dll"="usbcdc.dll"
      [2_2]
        [USBCDC_ClassDriver]
          REG_SZ "Dll"="usbcdc.dll"

[HKLM\Drivers\USB\ClientDrivers\USBCDC_Template_Default]
  REG_SZ "Dll" = "usbcdc.dll"
  REG_SZ "Prefix" = "COM"
  REG_SZ "IClass" = "{2C3D239D-D718-42a6-A446-290BE1886FD0}"
```

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

```
[HKEY_LOCAL_MACHINE\Drivers\USB\LoadClients]
  [5418_256]
    [Default]
      [255_0]
```

```
[USBCDC_ClassDriver]
    REG_SZ "Dll"="usbcdc.dll"

[HKEY_LOCAL_MACHINE\Drivers\USB\ClientDrivers\USBCDC_Template_Default]
REG_SZ "Dll"="usbcdc.dll"
REG_SZ "Prefix"="COM"
    REG_SZ "IClass" = "{2C3D239D-D718-42a6-A446-290BE1886FD0}"

[HKEY_LOCAL_MACHINE\Drivers\USB\ClientDrivers\USBCDC_Config]
    [VID_152A&PID_0100&IFC_00]
    REG_SZ "Template"="USBCDC_Template_Default"
```

5 Driver Configuration

In this section various registry settings required to load and configure the driver are discussed. It is recommend 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.

- `usbcdc_ce_inst_cls.vddproj`
This sample shows how to register the driver for USB CDC/ACM class-compliant USB interfaces. See section 5.2.1.
- `usbcdc_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. See section 5.2.2.
- `usbcdc_ce_inst_modem.vddproj`
This sample shows how to define modem settings to create a modem instance on top of the COM port. See section 5.5.

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/ACM class driver needs to be registered in case the device exposes CDC/ACM 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]
        [USBCDC_ClassDriver]
          REG_SZ "Dll"="usbcdc.dll"
      [2_2]
        [USBCDC_ClassDriver]
          REG_SZ "Dll"="usbcdc.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 = 2/2 which corresponds to the CDC 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.

5.2.2 Registration in Case of Non-class Compliant Descriptors

The following example shows how the USB CDC/ACM class driver needs to be registered with the bus driver in case the device does not expose CDC/ACM class compliant USB descriptors. For a discussion of descriptor layout variants see also section 3.1.2.

The example uses VID=0x152A (5418 decimal) and PID=0x0100 (256 decimal). Note that all numbers must be specified in decimal notation.

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

5.3 Driver-defined Registry Locations

5.3.1 Config Key

The USB CDC/ACM class driver stores interface-specific configuration information at the following registry location:

```
HKEY_LOCAL_MACHINE\Drivers\USB\USBCDC_Config
```

This registry location is called the config key. When the driver recognizes a USB interface then it builds a string which contains the VID, the PID and the interface number. Then the driver looks up a sub key of that name under the config key. The general format of those sub key names is:

```
HKLM\Drivers\USB\ClientDrivers\USBCDC_Config\VID_XXXX&PID_XXXX&IFC_XX
```

The XXXX sequences are placeholders and must be replaced by the appropriate values in hexadecimal representation. For example, if VID=0x152A and PID=0x0100 then the config key entry for interface number 0 will be:

```
HKLM\Drivers\USB\ClientDrivers\USBCDC_Config\VID_152A&PID_0100&IFC_00
```

In case of CDC-compliant descriptors (separate control and data interface) the interface number of the control interface needs to be specified under the config key.

In the interface-specific config sub key the driver looks up a REG_SZ value named `Template`. The `Template` value specifies the name of a template key. The contents of a template key is explained in more detail in the next section. An arbitrary name can be chosen for the template key. For example, to establish a link from interface 0 to the template key for COM port setup, the following registry entry needs to be created:

```
HKLM\Drivers\USB\ClientDrivers\USBCDC_Config\VID_152A&PID_0100&IFC_00
  REG_SZ "Template" = "USBCDC_Template_COM"
```

In fact, this causes the driver to use the configuration settings stored under

```
HKLM\Drivers\USB\ClientDrivers\USBCDC_Template_COM
```

for the COM port instance it creates for interface 0 of the USB device.

Note that more than one interface config key can refer to the same template key. Such an example is shown below.

```
HKLM\Drivers\USB\ClientDrivers\USBCDC_Config\VID_152A&PID_0100&IFC_00
  REG_SZ "Template" = "USBCDC_Template_COM"
```

```
HKLM\Drivers\USB\ClientDrivers\USBCDC_Config\VID_152A&PID_0100&IFC_02
  REG_SZ "Template" = "USBCDC_Template_COM"
```

In this case the same settings will be used for both COM port instances created by the driver, the COM port for interface 0 and the COM port for interface 2.

If no interface-specific config sub key exists and thus no template key is specified for a given interface then for that interface the driver uses a default template key:

```
HKLM\Drivers\USB\ClientDrivers\USBCDC_Template_Default
```

5.3.2 Template Key

The template key specifies settings to be applied to COM port or modem instances created by the driver. Several COM port or modem instances can be created from the same template key. An example of a template key for creating COM port instances is given below.

```
HKLM\Drivers\USB\ClientDrivers\USBCDC_Template_Default
  REG_SZ "Dll" = "usbcdc.dll"
  REG_SZ "Prefix" = "COM"
  REG_SZ "IClass" = "{2C3D239D-D718-42a6-A446-290BE1886FD0}"
```

The values `Dll` and `Prefix` are required and must be present. The value `IClass` is an optional customizable GUID, it is used for GUID based COM port enumeration (see section

`Dll` specifies the name of the driver to be loaded for the COM port and must be set to the name of the USB CDC/ACM class driver DLL.

`Prefix` must be set to "COM" for a serial port driver.

`IClass` should be set to a custom GUID used by the application to detect the COM ports.

Optionally, further settings can be specified in the template key as listed below.

- A static COM port number can be assigned; refer to section 5.4 for details.
- A modem instance can be created on top of the COM port; refer to section 5.5 for details.
- Driver configuration parameters can be adjusted; refer to section 5.6 for details.

5.4 COM Port Number Assignment

By default, when a new COM port instance is created the Windows CE system automatically assigns a COM port number by selecting the next available or unused number. The COM port number can also be assigned explicitly by specifying the `REG_DWORD` value `Index` in the template key (see also 5.3.2).

For example, to assign COM3 to a port the following setting needs to be created:

```
HKLM\Drivers\USB\ClientDrivers\USBCDC_Template_COM
  REG_DWORD "Index" = 3
```

Note that in case of an explicit assignment the template key must not be referenced by more than one interface-specific config sub key (see also 5.3.1). A separate template key and a dedicated `Index` value must be used for each COM port instance.

5.5 Modem Configuration

It is possible to create a modem instance on top of a COM port. To achieve this, the template key has to contain a sub key with settings for the UNIMODEM driver shipped with Windows CE. If the `Unimodem` sub key is present then the UNIMODEM driver will be loaded on the COM port.

The Unimodem sub key has several sub keys, namely Init, Config and Settings, which contain various further settings. For a complete description of UNIMODEM configuration refer to the Windows CE documentation. To find the topic in the documentation, checkout <http://msdn.microsoft.com/library> and search for "Unimodem Registry Settings".

An example of a modem configuration is given below.

```
[HKLM\Drivers\USB\ClientDrivers\USBCDC_Template_Modem\Unimodem]
  REG_BINARY "DevConfig"=20,00, 00,00 78,00,00,00 10,01,00,00,
              00,4B,00,00 00,00, 08, 00, 00,00,00,00
  REG_DWORD "DeviceType"=1
  REG_SZ "FriendlyName"="USBCDC Modem Device"
  REG_SZ "TSP"="unimodem.dll"

[Config]
  REG_SZ "BaudRate"="230400"

[Init]
  REG_SZ "1"="AT<cr>"
  REG_SZ "2"="AT&FE0Q0V1&C1&D2S0=0<cr>"

[Settings]
  REG_SZ "Answer"="ATA<cr>"
  REG_SZ "CallSetupFailTimeout"="ATS7=<#><cr>"
  REG_SZ "DialPrefix"="D"
  REG_SZ "DialSuffix"=";"
  REG_SZ "Monitor"="ATS0=0<cr>"
  REG_SZ "Prefix"="AT"
  REG_SZ "Pulse"="P"
  REG_SZ "Reset"="ATZ<cr>"
  REG_SZ "Terminator"="<cr>"
  REG_SZ "Tone"="T"
```

5.6 Driver Configuration Parameters

The USB CDC/ACM 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 template key of a given COM port a modem instance. For more information about that key see also section 5.3.2.

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

5.6.1 BufferCountIn

Default: 8 buffers

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

5.6.2 BufferCountOut

Default: 8 buffers

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

5.6.3 BufferSizeFactorIn

Default: 16

The MaxPacketSize value reported in the descriptor of the bulk IN endpoint is multiplied by this factor to calculate the size of a USB transfer buffer. For example, if MaxPacketSize = 64 bytes and BufferSizeFactorIn = 16 then each data buffer will be 1024 bytes in size.

Note: If the device does not terminate MaxPacketSize-aligned transfers with a zero packet then BufferSizeFactorIn must be set to 1. See section 3.3 for a general discussion of this topic.

5.6.4 BufferSizeFactorOut

Default: 16

The MaxPacketSize value reported in the descriptor of the bulk OUT endpoint is multiplied by this factor to calculate the size of a USB transfer buffer. For example, if MaxPacketSize = 64 bytes and BufferSizeFactorIn = 16 then each data buffer will be 1024 bytes in size.

5.6.5 FifoSizeIn

Default: 4096 bytes

Specifies the size, in bytes, of the internal RX FIFO.

5.6.6 FifoSizeOut

Default: 4096 bytes

Specifies the size, in bytes, of the internal TX FIFO.

5.6.7 SendShortPacketsOut

Default: 1 (yes)

If this parameter is set to 1 then the driver sends a zero packet to the OUT endpoint if the TX FIFO gets empty and the length of the last transfer was a multiple of MaxPacketSize. If SendShortPacketsOut is set to 0 no zero packet will be send.

This parameter should be changed only if the device is not able to handle zero packets. See section 3.3.3 for more details about this configuration option.

5.6.8 SupportSetControlLineState

Default: 1 (yes)

If this parameter is set to 1 then the driver will issue the SET_CONTROL_LINE_STATE class request defined by the USB CDC/ACM specification. If this parameter is set to 0 then the driver

will never issue this class request. This parameter should be changed only if the device is not able to handle the class request. See also section [3.2](#).

5.6.9 SupportSetLineCoding

Default: 1 (yes)

If this parameter is set to 1 then the driver will issue the `SET_LINE_CODING` class request defined by the USB CDC/ACM specification. If this parameter is set to 0 then the driver will never issue this class request. This parameter should be changed only if the device is not able to handle the class request. See also section [3.2](#).

5.6.10 SupportGetLineCoding

Default: 1 (yes)

If this parameter is set to 1 then the driver will issue the `GET_LINE_CODING` class request defined by the USB CDC/ACM specification. If this parameter is set to 0 then the driver will never issue this class request. This parameter should be changed only if the device is not able to handle the class request. See also section [3.2](#).

5.6.11 SupportSendBreak

Default: 1 (yes)

If this parameter is set to 1 then the driver will issue the `SEND_BREAK` class request defined by the USB CDC/ACM specification. If this parameter is set to 0 then the driver will never issue this class request. This parameter should be changed only if the device is not able to handle the class request. See also section [3.2](#).

5.6.12 PowerManagementLevel

Default: 2 (system power management)

This parameter sets the power management level.

- 0 means no power management.
- 1 means internal power management. If the last driver is closed the device is set into suspend state
- 2 means system power management. The system power manager controls the state of the driver. The user could control the behaviour over the power manager API.
- 3 means advanced power management. Enables a internal idle timer that sets the device in suspend. This option is only available if the device has a serial number.

See also section [5.8](#).

5.7 Device Specific Parameters

Device specific parameters are located under:

```
HKLM\Drivers\USB\ClientDrivers\USBCDC_Config\VID_XXXX&PID_XXXX
```

The XXXX sequences are placeholders and must be replaced by the appropriate values in hexadecimal representation.

The following parameters are device specific. They are only valid if the driver is in advanced power management mode. All parameters are optional. If a configuration parameter does not exist in the registry the driver uses an internal default value.

5.7.1 IdleTimerInterval

Default: 3000 ms

This parameter sets the time after that the driver sets the USB device to suspend if no activity happens. A value of 0 means disable the idle timer.

5.7.2 SetRemoteWakeUp

Default: 0 (No)

If set to 1 a SetFeature/ClearFeature remote wakeup request is send to the device. A value of 0 does not send the request.

5.7.3 RequestD3viaPM

Default: 0 (No)

If set to 0, the PM (Power Manager) is not called if the the driver goes into D3 state. Otherwise the PowerNotify function of the PM is called.

5.8 Power Management

The USBCDC driver supports different levels of power management. The used power management level could be configured in the registry, see section 5.6.12. The following levels are possible:

- No power management (Level 0)
This means the driver didn't suspend the device and the driver is not registered with the system power manager.
- Internal power management (Level 1)
This means the the driver sets the device in suspend state if the last COM port handle is closed and resumes the device if the first handle is open. The driver didn't registered with the system power manager.

- System power management (Level 2)

This means the driver is registered with the system power manager. The driver suspends the device if the system power manager requests a power state of D3 or D4. He resumes the device in the power states D0, D1 and D2. The application developer can decide the power state of the device. If a device has multiple virtual COM ports the power state must be set for every COM port to suspend the device.

- Advanced power management (Level 3)

This enables a internal idle timer that sets the device in suspend. The option is only available if the device has a serial number.

5.9 COM Port Enumeration

Normally application developers did not know what COM port number their device have. They let the user decide the COM port number or tries to open every COM port and check if it is the right device.

The preferred method to enumerate COM ports of USB CDC driver is the GUID-based enumeration. That guarantees that only COM ports of your devices are opened. At first the value IClass needs to be added to the driver registry key. The key contains a custom GUID for your devices.

The GUID-based enumeration is shown in the SimpleComTest source-code example in the file DeviceEnumerator.cpp.

```
[HKEY_LOCAL_MACHINE\Drivers\USB\LoadClients]
  [Default]
    [Default]
      [10_0]
        [USBCDC_ClassDriver]
          REG_SZ "Dll"="usbcdc.dll"
      [2_2]
        [USBCDC_ClassDriver]
          REG_SZ "Dll"="usbcdc.dll"

[HKLM\Drivers\USB\ClientDrivers\USBCDC_Template_Default]
  REG_SZ "Dll" = "usbcdc.dll"
  REG_SZ "Prefix" = "COM"
  REG_SZ "IClass" = "{2C3D239D-D718-42a6-A446-290BE1886FD0}"
```

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/ACM 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 usbcdc.log in the root directory of the Windows CE device. These messages can help to analyze problems.

The usbcdc.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.