Thesycon® Systemsoftware & Consulting GmbH

# Mobile Phone Windows Driver Software

Technical Documentation

Version: 1.22
Date: 13 February 2013

Authors: Günter Hildebrandt, G.Hildebrandt@thesycon.de

Thesycon Systemsoftware & Consulting GmbH
Werner-von-Siemens-Str. 2
98693 Ilmenau
Germany
+49 3677 8462 0
http://www.thesycon.de

# Contents

# 1 References

[1] USB MI IF, USB Multi-Interface Driver Interface, An interface to switch the device configuration, Reference Manual, Thesycon

[2] USB CDC/ECM Class Driver, Reference Manual, Thesycon

[3] PPP Framer in PC CDC ACM Driver, Interface Enhancements and Device Driver Design Documentation, Thesycon.

[4] USB Selective Suspend Interface, Interface Enhancements and Device Driver Design Documentation, Version: 0.2, 6 October 2008

# 2 Introduction

This document is confidential und must not be passed to any third party without written permission of Thesycon.

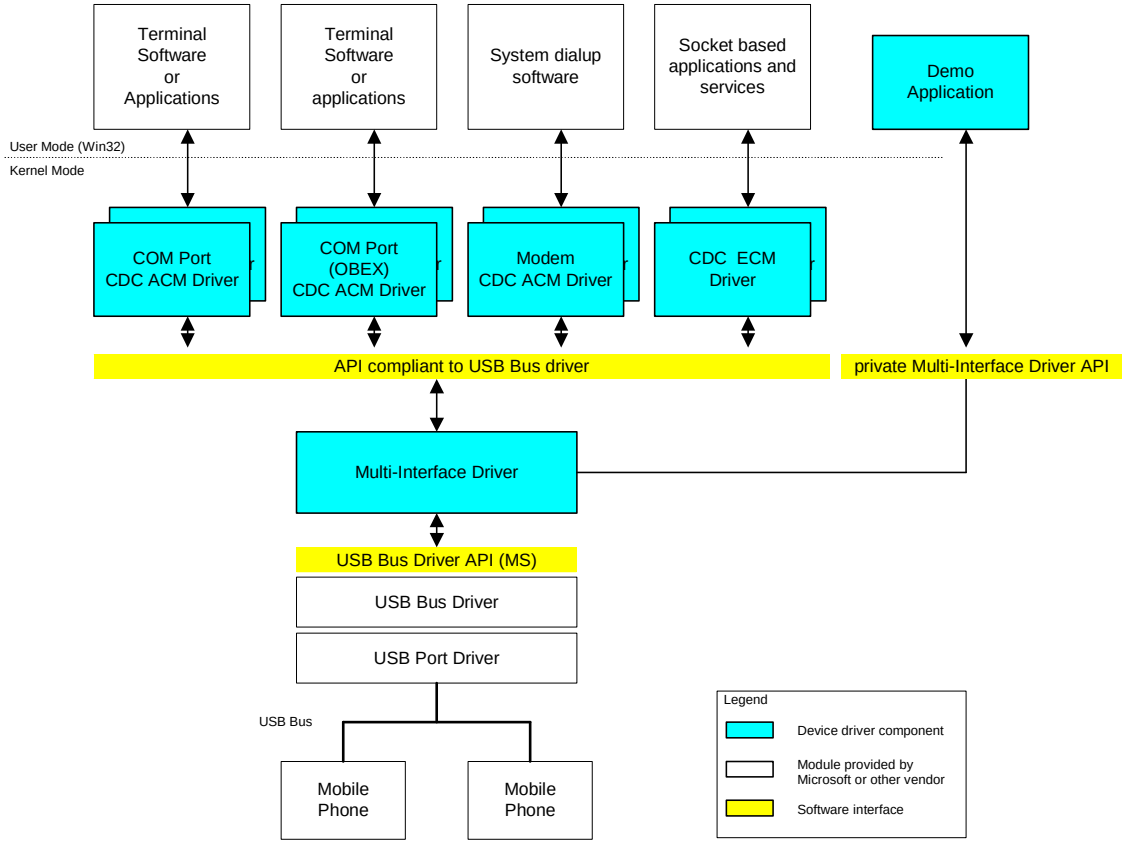
This document contains the design architecture of the driver stack, special features of the drivers and some limitations. The driver stack consists of a Multi-Interface driver, a Communication Device Class (CDC) Abstract Control Model (ACM) compliant driver and an ECM network driver. The CDC ACM driver can be installed either as a modem driver or a COM port driver. A special build version of this driver is compliant to the WMC OBEX model.

The features of the driver stack are modular. That means most of the features can be enabled during compile time. Such features are marked as “optional” in this document.

The reader of this document should be familiar with the USB 2.0 Specification and with some class specification like CDC , Mass Storage and WMC.

# 3 Architecture

The following figure shows the architecture of the driver stack. The lower end of the driver stack uses the API provided by the Microsoft USB bus driver. This bus driver API is available for the three host controller types Universal Host Controller, Open Host Controller and Enhanced Host Controller. On the upper edge the driver stack provides standard interfaces that can be used by applications or system services.




## 3.1 The Multi-Interface Driver

Windows contains a Multi-Interface driver for USB devices. But this driver has a number of limitations. For that reason a replacement driver was developed. The new features of this driver are:

- Correct handling of multiple USB interfaces
- Handling of Multi-Configuration devices (optional)
- Support for switching between USB configurations (optional)
- Exporting a private API defined in [1]. (optional)
- Handling Selective Suspend of the device (under development, optional)
- Fixing a bug of Microsoft's bus driver stack to handle control requests in a correct way.

### 3.1.1 Installation of the Multi-Interface Driver

The Multi-Interface driver is installed directly on top of the bus driver. Depending of the architecture of the phone there may be a Microsoft driver that is installed on top of the device node automatically and without user interaction. This can be the Multi-Interface driver provided by Microsoft or if the case of a CD-less installation the Mass Storage Driver. In both cases a special installation must be performed to replace the driver with the Multi-Interface driver.

### 3.1.2 CD Less Driver Installation

The phone can export a USB Mass Storage interface. The kernel mode drivers for such a interface are preinstalled and WHQL certified for the complete Mass Storage Class. This enables the installation of the kernel mode driver automatically. The file system on the Mass Storage device can be designed in such a way, that an auto-run option is activated and the installation of the PC software is started if the phone is connected the first time to a computer. Note: The auto run feature of Windows can be disabled by the user.

This architecture uses the possibility to export a set of USB configurations. If the CD less software installation is used, the first USB configuration contains one USB interface that is compliant to the USB Mass Storage class. The Microsoft's driver stack activates always the first USB configuration.

The software installation copies the installer to a temporary location on the hard disk and replaces the Mass Storage driver by the Multi-Interface driver. The Multi-Interface driver activates a different USB configuration that contains the functional interfaces of the phone. If the Multi-Interface driver has been installed successfully, it is loaded automatically if the phone is connected or turned on the next time. If the phone is connected to a PC, where the software is not installed, the installation process starts from the beginning.

The driver installation process requires administrator privileges.

The CD less installation is optional and must be supported by the phone.

### 3.1.3 Private Interface of the Multi-Interface Driver

The private interface of the Multi-Interface driver is described in detail in [1]. It provides the following features:

- Get the kernel driver information
- Get the currently activates USB configuration
- Set an USB configuration

There is a small command line demo application that can be used to switch the USB configuration of the device.

This interface is optional and required by the CD-less installation.

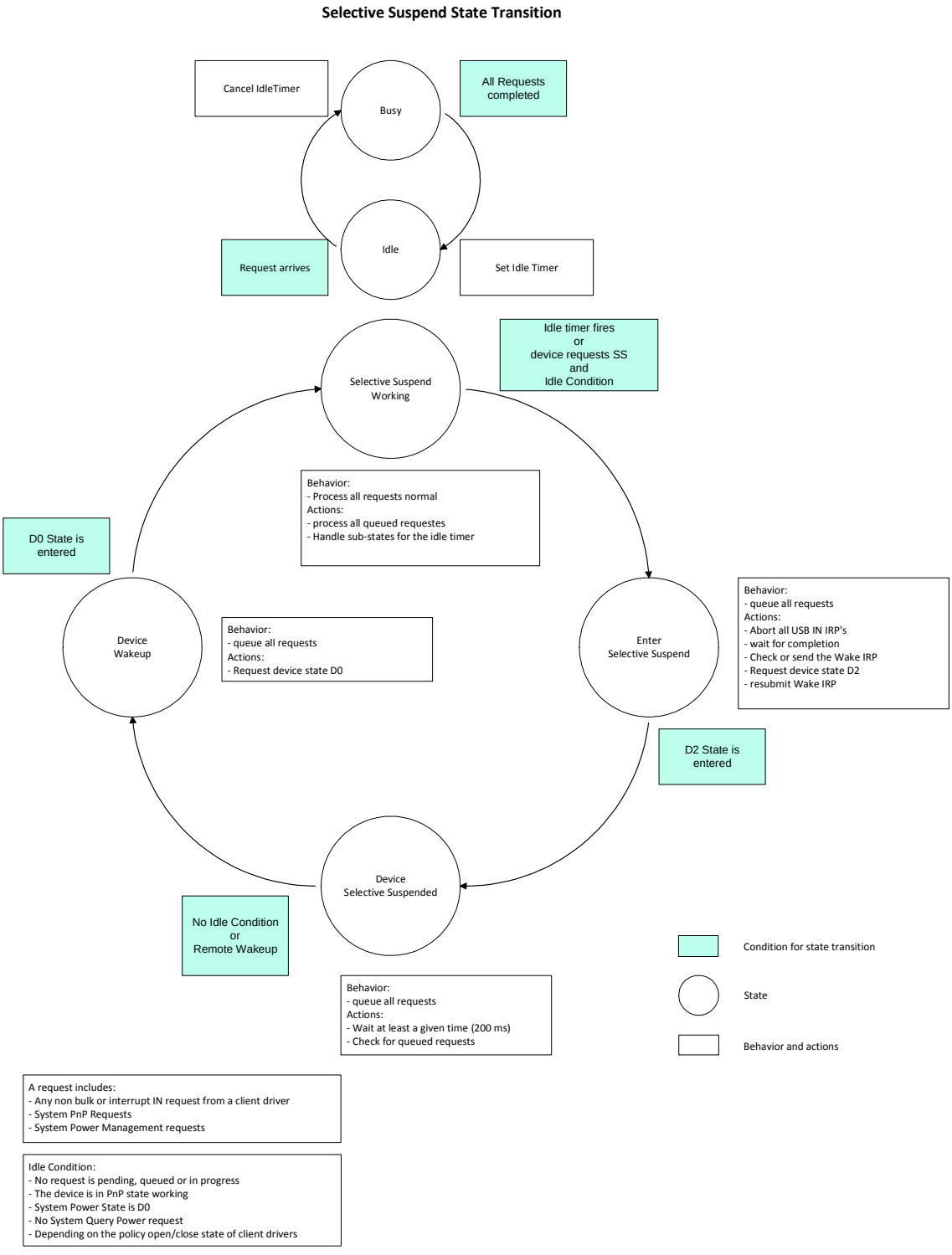
### 3.1.4 Selective Suspend Support

The selective suspend feature is described in [4]. It enables the driver to switch the device in a selective suspend state if it is not used. On the USB bus the Suspend state is entered. This feature enables the device to reduce the power consumption. The device must not change any state that is visible from the PC application if it enters suspend and returns to working state.

The main part of the Selective Suspend feature is implemented in the Multi-Interface driver. If the driver detects that the device and all client drivers are in an idle state the state transition to selective suspend is started. The driver supports two ways to trigger the state transition:

- With a idle timer in the MI driver after a idle period
- A message from the device on a USB interrupt pipe

The following picture shows the state diagram used for selective suspend:



The state transition to selective suspend can be blocked if either the CDCACM is driver is opened or if the Network adapter reports a connected Network Cable.

The protocol that is used to communicate with the device is described in [4].

As soon as the transition to selective suspend has been started all requests from client drivers and system notifications are stored in queues. If a request arrives in a queue the transition to a working state is started. Because all client driver requests and system notifications are handled in working state, the selective suspend is transparent to the client drivers.

To enter selective suspend the MI driver has to abort all pending requests. If all child drivers are idle the only pending requests are URB's with read requests on bulk or interrupt IN pipes. After all data traffic has been stopped the MI driver aborts all these requests. There is no risk of any data lose. The MI driver completes the requests with status success and a length 0. All client drivers must handle such an empty request without error and resubmit the IRP. During resubmission the IRP's are stored in a queue. A read request on a bulk or interrupt pipe is not handled as an activity that marks the device as busy.

The device can wake the MI driver with Remote Wakeup if it wants to send data to the PC.

The state transition to suspend and back to working state takes some time. The time results from the handling of internal requests in the driver and the system as well as data exchange between the driver and the device. It is difficult to make assumptions about the time schedule. This is a problem for the dimension of the buffer size inside the phone to keep data stored until the PC becomes ready to read them.

#### 3.1.4.1 Limitations of Selective Suspend

During the tests we have encountered some problems that are caused by the system software. The following paragraphs contain a short explanation of the observed problems.

Because of the limitation of selective suspend in Windows XP and Vista Selective suspend is supported in Windows 7 and 8, only. On other systems the selective suspend state is never entered.

##### 3.1.4.1.1 IOCTL_INTERNAL_USB_SUBMIT_IDLE_NOTIFICATION

On Windows XP is required to use an IRP with the function code IOCTL_INTERNAL_USB_SUBMIT_IDLE_NOTIFICATION to trigger the selective suspend state. The IRP registers a call back function. The call back function is called with a delay between 500 and 1000 ms. This delay is a problem if the device should be switched to selective suspend a short time after a trigger event has been occurred.

##### 3.1.4.1.2 Short USB Suspend period

The suspend period on the USB is shorter that the time period that can be measured between the call to the completion routine of the device power down IRP and the request for the wake up IRP. This effect was observed with the usage of the IRP IOCTL_INTERNAL_USB_SUBMIT_IDLE_NOTIFICATION. To make sure that the device sees the suspend state on USB the driver must keep the selective suspend state for a given time period even if in the mean time a new request has been arrived. This time period delays the wakeup process and the handling of the request. It is difficult to define a reasonable time period for this time interval.

#### 3.1.4.1.3 Problem after System Power Down

If the selective suspend is triggered some seconds after system power down (Sleep or StandBy state) it does not work. This seems to be a bug in the Microsoft bus driver. The selective suspend is blocked for some seconds after system power down.

#### 3.1.4.1.4 Call Back is not called

On Windows XP SP2 and SP3 the call back function registered with IOCTL_INTERNAL_USB_SUBMIT_IDLE_NOTIFICATION is not called if an external USB 2.0 hub is connected between the PC and the device. It seems to be a bug in the Microsoft bus driver stack. There is no known workaround for this problem. For that reason the selective suspend state is never entered, if a external hub is used.

## 3.2 The CDC ACM Driver

This driver can be used in one of the following ways:

- As a COM port driver in the device class Ports (COM & LPT)
- As a Modem in the device class Modems
- As an OBEX protocol driver compliant to the WMC specification.

The binary driver for the usage in the Ports class and in the Model class is the same. The relation to the device class is defined in the INF file. The OBEX driver is an build option. The OBEX protocol does not used the signalling of serial status lines. For that reason this interface uses only two bulk pipes.

Each of the interfaces is optional but it makes sense to have at least one interface to access the phone.

### 3.2.1 Differences in the COM Emulation

The Window COM port API is very close to the hardware of a UART. That means the value of the UART status register can be requested by the user. Some other features are very close to the UART behaviour. For that reason the driver behaves in some points different to the Microsoft driver, that is part of the system and serves the UART based COM ports.

- Flow control is always handled with the USB flow control.
- The status lines are transmitted and indicated to the device and the COM port. But they do not have a meaning for flow control in the driver.
- A char submitted with TransmitCommChar is transmitted as a normal char.
- The ReadIntervalTimeout is handled on the time difference of USB blocks.
- The Software does not support Xon/Xoff handshake. It can be set by software, but the driver will never send a Xon or Xoff char. The flow control is done by USB (NAK/ACK) tokens.
- The driver does never set or clear the signals RTS or DSR. The application can control these signals.
- The driver does not support 5, 6, or 7 data bits.

- The content of the internal FIFO's is not preserved, if a new FIFO size is set.

### 3.2.2 Modified Plug And Play Behaviour

A COM that is part of the operating system typically never disappears. A lot of application like Hyperterminal that are using COM ports are not Plug and Play compliant. That means they does not detect the removal of a COM port and close it. The Microsoft driver as well as the CDC ACM COM port driver indicates this Plug and Play events to the user mode.

On Windows 2000 we have observed, that the Windows FAX client is also not PnP compliant. If it is configured to receive FAX from a virtual COM port and the COM port is removed, it stops receiving FAX until the PC is restarted.

For these reasons we have enhanced the user interface in the following way. If the device is removed and the COM port is used, the driver keeps the handle of the application valid. It sends the Plug and Play event "Device is removed" to the user mode. A PnP compliant application can close the interface. If the application closes the handle the driver is unload. If the device with the same serial number is connected to the PC the driver connects the application handle to the phone.

Requests that are sent in the time where the phone is not connected, are stored in the circular buffer of the driver. These requests are sent if the device is connected. Timeouts for send and read requests are still active.

This feature enhances the compatibility of the driver interface to not Plug and Play compliant applications. It has no negative influence to WHQL test or to Plug and Play compliant applications.

### 3.2.3 Lower Layer Flow Control

The USB protocol wastes a lot of band width if the flow of the data in OUT direction is blocked by the device for a long time period. To avoid this an additional flow control is added to the CDC ACM driver. The driver stops sending data if the phone signals a full buffer in OUT direction. If the buffer in the phone becomes empty the phone enables the data flow again.

This feature is triggered by the phone and is optional.

### 3.2.4 Optimized PPP protocol

The PPP protocol contains an escaping of characters. This is a CPU consuming task in the phone. To save CPU power in the phone the CDC ACM driver can perform the escaping of the characters instead of the phone. This feature uses a special protocol defined in [3]. The phone triggers the process of PPP escaping in the driver.

This feature is optional.

### 3.2.5 Parameters

The CDC ACM driver has some parameters that can be configured with the INF file or temporary in the registry.

#### 3.2.5.1 RequestTimeout

This value defines the request timeout for any class or vendor request on EP0. When the device does not answer to the request in the given timeout the request is cancelled and traded as failed.

#### 3.2.5.2 TimerInterval

Contains the period that is used to check the timeout setting for read and write operations on the COM port. This value is the granularity for the timeout settings on the COM port.

#### 3.2.5.3 useCtsFlowControl

If this parameter is grater than 0 the driver does not send data when the device signals CTS inactive.

#### 3.2.5.4 ReadBufferSize

Is the buffer size used for read operation on the bus.

#### 3.2.5.5 WriteBufferSize

Is the buffer size used for write operation on the USB bus.

#### 3.2.5.6 ReadCircBufferSize

Is the initialize size of the circular buffer inside the driver. The application can set a different value with SetupComm().

#### 3.2.5.7 WriteCircBufferSize

Is the initialize size of the circular buffer inside the driver. The application can set a different value with SetupComm().

#### 3.2.5.8 VID

Is an encrypted value that contains the VID that is valid for the driver operation.

#### 3.2.5.9 PID

Is an encrypted value that contains the PID that is valid for the driver operation. The checking of the VID/PID is optional.

## 3.3 CDC ECM Driver

The CDC ECM (Ethernet Control Model) driver is a connection less network card driver that implements the NDIS protocol defined by Microsoft. The implemented Media type is Ethernet. The driver is installed in the Network Adapters class. The driver is optional. A detailed description is given in the reference manual [2].

# 4 Driver Software Installer

The driver software installer is based on the NSIS installation platform. It is an installer with a graphical user interface. It supports the language English. Other languages can be added on request if the customer can provide the translated texts.

The driver installer performs the following actions:

- Check of the supported operating system
- Check of the administrator privileges
- Deleting all old installations: device nodes and pre-installed INF files
- Pre-Installing of the required drivers
- Support for WHQL certified drivers
- Creation of log files
- Creation of an uninstall entry in the Add/Remove Programs dialog
- Support for CD less installation (optional)
- Silent installation without user interface

The installation of the kernel drivers require a special installation program. It is not reliable to let the user install the drivers with the system provided options.

The installer is optional.

# 5 Supported Operating Systems

The driver stack supports the following operating systems with the most current service packs:

- Windows XP 32 bits
- Windows XP 64 bits
- Windows Vista 32 bits
- Windows Vista 64 bits
- Windows 7 32 bits
- Windows 7 64 bits
- Windows 8 32 bits
- Windows 8 64 bits

The support for each operating system is optional.

# 6 WHQL Certification

The driver stack has passed the WHQL test several times in the “unclassified” category successfully. How ever the DTM test bench is frequently updated by Microsoft. For that reason the test must be repeated for each driver release.

The delivered driver does not have a WHQL certificate by default. The WHQL tests must be performed either by Thesycon or by the customer it self. If Thesycon should perform the tests this must be defined in a contract.

The WHQL test can be performed for the following operating systems:

- Windows Vista 32 bits
- Windows Vista 64 bits
- Windows 7 32 bits
- Windows 7 64 bits

# 7 Driver Debugging

## 7.1 Event Log Entries

If the driver detects a major problem during the startup process or when the COM port is opened it creates an entry in the event log of the system. The event log can be opened with the context menu on 'My Computer' -> Manage -> Event Viewer -> System. The entries are created in the category error.

## 7.2 Enable Debug Traces

The setup_dbg installer contains debug version of the drivers. Execute the installer to install the debug version of the drivers. The debug version of the driver can generate text messages on a kernel debugger or a similar application to view the kernel output. These messages can help to analyze problems.

The output of the driver can be controlled by a registry key: \HKEY_LOCAL_MACHINE\System\CurrentControlSet\services\ YOUR_SERVICE_NAME. YOUR_SERVICE_NAME is the name of the service name defined in the .INF file. It is typically the name of your driver. Edit the DWORD value key TraceMask. The messages are grouped in special topics. Each topic can be enabled with a bit in the debug mask. To enable the messages on bit 5 the DebugMask must be set to 0x00000020. The DebugMask contains the or'ed value of all active message bits. See the text files "trace_xxx.txt" in the doc folder to see the meaning of each bit.

Use DebugView to collect the traces. Debug View is a free tool that can be downloaded from Microsoft.

On Windows Vista an later you have to enable the kernel debug traces by adding a registry key. Create the key:

"HKLM\CurrentControlSet\Control\Session Manager\Debug Print Filter".

Under this key create a DWORD value with the name "DEFAULT" and set the value to 0x0f.

After you have done this reboot the PC. This is required to activate the system debug filter.

Run DebugView with Administrator privileges. It installs a kernel driver during start up and it cannot collect kernel traces if the driver is not loaded. Enable the kernel traces in the menu.