SYS TEC
ELECTRONIC

USB-CANmodul

**GW-001, GW-002,
3004006, 32040xx, 34040xx**

Systems Manual

Edition February 2010

system house for distributed automation

	EUROPE	NORTH AMERICA
Address:	SYS TEC electronic GmbH August-Bebel-Str. 29 D-07973 Greiz GERMANY	PHYTEC America LLC 203 Parfitt Way SW, Suite G100 Bainbridge Island, WA 98110 USA
Ordering Information:	+49 (3661) 6279-0 info@systec-electronic.com	1 (800) 278-9913 info@phytec.com
Technical Support:	+49 (3661) 6279-0 support@systec-electronic.com	1 (800) 278-9913 support@phytec.com
Fax:	+49 (3661) 6279-99	1 (206) 780-9135
Web Site:	http://www.systec-electronic.com	http://www.phytec.com

22th Edition February 2010

Table of Contents

Index of Figures and Tables

Preface

This USB-CANmodul Systems Manual describes the board's design and function. Precise specifications for the on-board microcontrollers can be found in the enclosed microcontroller Data Sheet/User's Manual.

In this manual, and in the attached schematics, low active signals are denoted by a "/" in front of the signal name (i.e.: /RD). A "0" indicates a logic-zero or low-level signal, while a "1" represents a logic-one or high-level signal.

Declaration of Electro Magnetic Conformity for the SYS TEC USB-CANmodul

The USB-CANmodul is a tested and ready-to-use interface device and must only be used as such.

Note:

The USB-CANmodul should not be operated without additional protection circuitry and further testing if cables to the product's connectors are longer than 3 m. It is required to use shielded CAN cables.

The USB-CANmodul fulfills the norms of the European Union’s Directive for Electro Magnetic Conformity only in accordance to the descriptions and rules of usage indicated in this hardware manual (particularly in respect to the described steps for putting the device into operation).

Implementation of SYSTEC products into target devices, as well as user modifications and extensions of SYSTEC products, is subject to renewed establishment of conformity to, and certification of, Electro Magnetic Directives. Only after doing so the devices are allowed to be put into circulation.

Introduction

Unveiled in 1995, the **Universal Serial Bus (USB)** connectivity standard provides a simple and convenient way to connect various peripheral devices to a host-PC. It will replace a wide variety of serial and parallel connections. The USB standard allows up to 127 devices to be connected to the PC without using multiple connector types, without interrupt conflicts (IRQs), hardware address adjustments (jumpers) or channel changes (DMA). USB provides powerful true hot plug-and-play capability; i.e., dynamic attach and recognition for new devices. It allows the user to work with those devices immediately without restarting the operating system.

The USB-CANmodul takes advantage of this communication standard and provides an easy to use portal from a host-PC to a CAN network. Connecting the USB-CANmodul to the host-PC is simple. The included USB cable supports the two types of USB connectors, type A and type B. The type A plug connects to the host computer or an upstream hub. Type B plug connects downstream to the USB-CANmodul. The USB interface enables data transfer with a rate of up to 12 MBit/s. With a uniform connector for all device types, the system is absolutely user friendly.

Once the USB-CANmodul is connected to the host-PC, the operating system reads the configuration data and automatically loads the device driver. All CAN messages are transferred transparently through the USB Bus. CAN Baud Rates of up to 1 MBit/s are supported. The transmitted and received CAN messages are buffered by the USB-CANmodul. The device supports CAN messages according to CAN 2.0A and 2.0B specifications (11- and 29-Bit identifiers). Connection to the CAN bus meets the CiA Standard DS 102 (DB-9) and features optional optical isolation of the CAN signals.

Drivers for Linux, LabView (contributed), Windows 2000/XP and higher are provided for the USB-CANmodul. The USB configuration tool for Windows enables connectivity and management of more than one device on the USB bus. This USB network is configured using device numbers which are assigned by the user and are stored in an EEPROM. The functions for data exchange with the USB-CAN application are available through a DLL (Dynamic Linked Library). The enclosed demo program shows the easy handling of the DLL API functions.

This manual refers to the following USB-CANmodul versions:

Order number	Features
GW-001	Obsolete, but software support for compatibility reasons. - Galvanic isolation configurable via Jumper. - No Software support since Windows driver version V4.00.
GW-002	Obsolete, but software support for compatibility reasons. - More compact housing: 102x54x30 (LxBxH in mm), protection class IP40, supports DIN-rail mounting - USB-CANmodul with galvanic isolated available under separate order number. Thus, it is not longer needed to open the housing and to supply power via CAN-bus. - CAN-ground (CAN-GND) and CAN-shield (CAN-SHLD) not connected internally. - Various CAN-transceivers available optionally for low-speed CAN or single-wire CAN, special PCB pads and socket connector for assembly of special CAN-transceivers - External power supply of up to 30V possible, depending on the CAN-transceiver used - 8-bit user port (I/O with TTL level) provides for customer-specific extensions
3004006	Multiport CAN-to-USB with 16 CAN-channels - 19“ rack mounted device - Contains 8 logical devices with 2 channels each. - Fast 32-bit microcontroller - External power-supply with 230VAC/500mA (inlet connector for non-heating devices) - Galvanic isolation of the CAN-channels
3204000, 3204001	USB-CANmodul1 - more compact enclosure with dimensions of 78x45x18 (LxWxH in mm) - Single CAN interface - Fast 32-bit MCU, enhanced firmware - Power-supply via USB, current consumption max. 110mA - High-speed CAN transceiver 82C251 - Galvanic isolation available with order no. 3204001
3204000 R2, 3204001 R2	USB-CANmodul1 Revision 2 - same as above but includes better EMC behaviour - 120 ohm termination resistor can be set at PCB

Order number	Features
3204002, 3204003, 3204007, 3204008, 3204009, 3204011, 3204013, 3204017, 3204018, 3204019	USB-CANmodul2 - Two CAN-channels, independently utilizable - Fast 32-bit MCU, enhanced firmware - Power-supply via USB - High-speed CAN transceiver 82C251 or Low-speed CAN transceiver TJA1054 or Single-wire CAN transceiver AU5790 - Galvanic isolation available with order no. 3204003 - 8-bit user port (I/O with TTL level) provides for customer-specific extensions with order no. 3204007
3204002 R2, 3204003 R2, 3204007 R2, 3204008 R2, 3204009 R2, 3204011 R2, 3204013 R2, 3204017 R2, 3204018 R2, 3204019 R2	USB-CANmodul2 Revision 2 - same as above but includes better EMC behaviour
3404000	USB-CANmodul8 - 8 CAN-channels, independently utilizable - Table case - Contains 4 logical devices with 2 channels each - Fast 32-bit MCU, enhanced firmware - External power-supply with 230VAC/500mA (inlet connector for non-heating devices) - Galvanic isolation of the CAN-channels
3404001	USB-CANmodul16 - 16 CAN-channels, independently utilizable - Table case - Contains 8 logical devices with 2 channels each - Fast 32-bit MCU, enhanced firmware - External power-supply with 230VAC/500mA (inlet connector for non-heating devices) - Galvanic isolation of the CAN-channels

References to Hardware and Software changes

In this section you will find references to new functions in the hardware and software of the USB-CANmodul.

The selection of arbitrary baud rates within the PCANview tool is supported starting with version 2.15. The *UcanWriteCanPort()* and *UcanReadCanPort()* functions were integrated for control of the low speed transceivers.

Software version 2.16:

- Following a new installation, the USB-CANmodul will appear in the device manager under the entry "**USB-CAN-Hardware**" instead of under the entry "**USB Controller**".
- Reading the serial number and the CAN controller's operational mode via the function *UcanGetHardwarInfo()* is now possible.
- The function *UcanGetVersionEx()* has been added to support extended version inquiry.
- The function *UcanInitCanEx()* is available for expanding the configuration of the SJA1000, i.e. listing only - mode
- The new functions *UcanConfigUserPort()*, *UcanWriteUserPort()* and *UcanReadUserPort()* are provided for use of the 8-bit user port.

Software version 2.17:

- Only the installation for Windows2000/XP is supported.
- Installation and operation under Windows98/Me is possible, however SYSTEC will offer no guarantee that this will work.
- The messages "bus off", "error passive" and "warning limit" were not previously passed on to the application
- The CAN status error is no longer cleared automatically by *UcanGetStatus()* after it has been read, rather it is cleared only when *UcanResetCan()* is called

Software version 2.18:

- The USBCAN.SYS device driver now also supports power management. If the computer is activate after being in stand-by mode, the device driver will be loaded again.
- The time stamp precision for receive messages in the structure *tCanMsgStruct* has been improved.
- The CAN status messages USBCAN_CANERR_BUSLIGHT and USBCAN_CANERR_BUSHEAVY are deleted automatically as soon as the CAN controller error counter drops below the corresponding threshold.
- The UCAN Config tool has been replaced by the new USB-CANmodul Control tool in the control panel.

- The PCANView tool didn't correctly answer RTR frames of 29-bit CAN messages in previous versions. This bug has been fixed starting at version 2.0.4 build 043 of PCANView.
- New function *UcanGetFwVersion()* implemented allowing version query of the module software.
- New demo project for Microsoft Visual Studio C/C++ 6.0.
- Software support for Borland Delphi as library and demo project.

Software version 2.19:

- An error was eliminated from the USBCAN32.DLL. The Connect Control callback function was not called in version 2.18, as soon as a USB-CANmodul was plugged in or plugged off from the PC.
- new: Software support for LabView as library and demo project

Software version 2.20:

- An error was eliminated from the USBCAN32.DLL. After calling function UcanDeinitHwConnectControl(), it was not possible to register a new callback function by means of UcanInitHwConnectControl().
- If the function UcanResetCan() was called while the receive buffer still contained CAN messages, old messages could happen to be received twice.
- The file USBCAN32.DLD does not exist any longer. Instead, the checkbox “Debug” in the USB-CANmodul Control tool is to be used to produce debug information.

Software version 3.00:

- Under some circumstances, the hardware connect control callback functions was called repeatedly although only one USB-CANmodul was plugged in or plugged off from the PC.
- New API functions introduced to support multiple CAN interfaces (available with GW-006, 3004006).
- The software driver now supports simultaneously operation of up to 64 modules on one PC.

Software Version 3.01:

- Bugfix: Under some circumstances the function UcanDeinitCan() could cause an access violation.
- Bugfix: The USB-CANmodul GW-002 does not transmit CAN-messages correctly after bus-off, although function UcanResetCan() was called.
- Bugfix: When using several USB-CANmoduls on the very same PC over multiple application instances, it happened that an application returned error code 0x06 (illegally handle), although the initialization was successful.
- Bugfix: When a CAN2.0A message was received with Multiport CAN-to-USB, the CAN-ID of the next CAN2.0B message was incorrectly transferred to the PC.

- Transmit-Echo for CAN-messages introduced. GW-002 and Multiport CAN-to-USB now return transmitted CAN messages as receipt message to the PC. This messages are marked as echo messages (transmit echo).
- Multiport CAN-to-USB now also supports a baud rate of 10kBit/s.
- Multiport CAN-to-USB activates the Traffic LED only, if the respective CAN-channel was previously initialized.
- Function UcanGetModuleTime() implemented to USBCAN32.DLL.

Software Version 3.02

- Bugfix: First time installation of version 3.01 did not work properly under Windows 2000
- Bugfix: On Multiport CAN-to-USB 3004006 function *UcanReadCanMsgEx()* returned with error code USBCAN_ERR_CANNOTINIT if function was called with parameter *bChannel_p = USBCAN_CHANNEL_ANY* and only the 2nd CAN-channels was initialized.
- Change of software: If the function *UcanInitCanEx.()* was called with CAN mode kUcanModeNormal, then the function *UcanReadCanMsg..()* returned with error code USBCAN_WARN_NODATA, although the buffer still contains CAN messages. This was caused by the transmit messages, which were continuously processed within the DLL. This was caused by the transmission echoes, which were continuously processed within the DLL. The USBCAN32.DLL was changed to skip the transmission echoes when CAN mode *kUcanModeNormal* is used and to return the next received CAN message instead.

Software Version 3.03

- Change of software: The USBCAN32.DLL now has two more receive buffers per CAN-channel. Thus it is possible to read CAN messages from one CAN-channel with function *UcanReadCanMsgEx()*, without having these blocked by CAN messages of the other CAN-channel.
- Change of software: Function *UcanResetCanEx()* now has a parameter to specify what components are to be reset and what is left unchanged.

Software Version 3.04

- Bugfix referring to Multiport CAN-to-USB 3004006: some CAN messages were sent twice if the bit rate was too high
- Bugfix referring to Multiport CAN-to-USB 3004006: Some CAN messages queued for transmission were deleted if function UcanInitCanEx2() was called for the second CAN-channel.
- Bugfix: Now the firmware-internal message buffer is not deleted by function UcanInitCanEx2(), if one of the channels is already initialized.
- Software change referring to Multiport CAN-to-USB 3004006: Forcing a firmware update is now possible.
- Software change: Now a restart is required after deleting the USB-CANmodul drivers from Windows-OS.

Software Version 3.05

- The drivers now support the USB-CANmodul1 (order no. 3204000, 3204001) and USB-CANmodul2 (order no. 3204002, 3204003)
- Software change: Now the hardware type is returned with structure tUcanHardwareInfoEx.
- Software change: The size of the receive and transmit buffers in USBCAN32.DLL is now configurable. Therefore the two new members/parameters *m_wNrOfRxBufferEntries* and *m_wNrOfTxBufferEntries* exist in structure tUcanInitCanParam.
- Bugfix referring to Manual: The pin description in Table 6 was wrong. Pin 9 and Pin 10 were switched.

Software Version 3.06

- New feature: Support for automatically transmission of cyclic CAN messages added. A new demo for Microsoft Visual Studio 6.0 or higher (MFC) is available in folder "%INSTALLDIR%\DemoCyclicMsg".
- New feature: Function *UcanGetMsgPending()* added for reading the number of pending CAN messages (current buffer contents).
- New feature: Function *UcanGetCanErrorCounter()* added for reading the current error counters from the CAN controller. This function is only available for the sysWORXX derivates of the USB-CANmodul.
- New feature: Function *UcanWriteCanMsgEx()* returns warning USBCAN_WARN_TXLIMIT when it was called to send more than one CAN messages but not all CAN messages could be sent because the transmit buffer is full. The number of accepted CAN messages were indicated to the application by the parameter *pdwCount_p*.
- New feature: Function *UcanGetVersionEx()* also supports to read versions of other driver files (such as loader, device driver and control panel application).
- Software change: Misspelling the define USBCAN_ERR_DISCONECT was corrected to USBCAN_ERR_DISCONNECT within the header file USBCAN32.H.
- Software change: USB-CANmodul Control revised with better construction of the dialog box elements. Already used USB-CANmoduls are also displayed (but grayed). The debug log file on using USB-CANmodul Control can be switched on/off separately.
- Bugfix: Function UcanResetCanEx(...,..., USBCAN_RESET_ONLY_STATUS) could not be called in state HW_INIT.
- Bugfix: Time stamp timer was corrected within sysWORXX derivates of USB-CANmodul. Time stamps of CAN messages were some different to the old hardware derivates (such as GW-001 or GW-002).
- Software change: All API functions of USBCAN32.DLL which returns an error code got the return code type UCANRET. The type UCANRET is also defined to 'unsigned char' like the previous type BYTE. Therefore applications need not to be recompiled for this change.

Software Version 3.07

- Bugfix: Modules of the sysWORXX USB-CANmodul series hangs when transmission of cyclic CAN messages was started.
- Bugfix: Access violation has occurred when multiple Windows users tried to initialize USB-CANmodules on Windows XP.
- Bugfix: Function *UcanWriteCanMsgEx()* returned warning message USBCAN_WARN_TXLIMIT instead of USBCAN_SUCCESSFUL when only one CAN message should be sent and it was successfully transmitted.
- Bugfix: Function *UcanInitHardware()* or *UcanInitHardwareEx()* returned USBCAN_ERR_ILLHW when more than one applications tried to initialize modules on the same time. A Mutex is added now to fix the problem.

Software Version 3.08

- Bugfix: If *UcanResetCanEx()* was called to reset the CAN interface without deleting the message buffers in firmware, then CAN messages were nevertheless deleted.
- New feature: USB-CANnetwork driver available for all sysWORXX series USB-CANmodules. The use of this driver has to be enabled for each device. Use the USB-CANmodul Control to switch and/or show the current state of it.
- New feature: Hardware list within the USB-CANmodul Control is now sorted by serial numbers.
- New feature: USB-CANmodul Control was extended to check for new driver versions on the SYS TEC homepage. Therefore a new tab sheet "Update" was added.
- New feature: UcanForceUpdate tool gets two new command line arguments. "-snr:" specifies the serial number of the USB-CANmodul which has to be updated. "-net:" specifies whether the new USB-CANnetwork driver should be used for the specified device.

Software Version 3.09

- Bugfix: The red CAN-status-LED on the sysWORXX USB-CANmoduls was not reset correctly after an error state had been erased.
- Bugfix: When function *UcanInitCanEx2()* was called for two CAN channels with transmit echo it could happen that the second channel never received the transmit echo.
- Bugfix: Device driver Usbcan.sys noticed a receive buffer overrun although there never was an overrun.
- New feature: Support of device driver Usbcan.sys for Microsoft Vista. Note: USB-CANnetwork Driver is not available under Vista!

Software Version 3.10

- Bugfix: When changes in Debug Settings were made in the USB-CANmodul Control they were not stored to the Registry if the windows user was not the administrator.

- New feature: The firmware of all new sysWORXX modules also filters the Data bytes 0 and 1 of CAN2.0A messages and the RTR-frame of CAN2.0A and CAN2.0B messages - similar to GW-001 and GW-002 (compare 2.3.5).
- New feature: DLL API function *UcanSetTxTimeout()* for dual channel sysWORXX modules was added. It configures a transmission timeout (default is 0). If a CAN message cannot be sent within this timeout the firmware sets a special internal state. All further CAN messages for this CAN channel will be deleted automatically and the CAN driver state is set: USBCAN_CANERR_TXMSGLOST to inform the application about the deletion. This way, the transmission through another CAN channel is not blocked any more. If the CAN message is sent the special CAN driver state will be deleted automatically. Once the timeout is configured to zero this feature is switched off.
- Software change: In DemoGW006 and DemoCyclicMsg it is shown how to load Usbcan32.dll for the run-time of the application.

Software Version 3.11

- New feature: Driver for Windows CE 5.0 is available. It is based on CPU-type ARMV4I and was tested on Intel PXA255 and PXA270. Other CPU can be ported on request.
- New feature: Linux driver version 2.02 r3 released, including a shared library. Kernel driver supports firmware update of USB-CANmodul1.

Software Version 4.00

- Bugfix: Calling function *UcanResetCanEx()* with parameter dwFlags_p that was equal to zero resulted for USB-CANmodul1 not to be able to receive any CAN messages.
- Bugfix: Calling function *UcanResetCanEx()* for cleaning receive buffer resulted not to be able to delete all received CAN messages of USB-CANmodul1 or GW-002.
- Bugfix: When function *UcanWriteCanMsgEx()* returned an error code, the variable referenced by parameter pdwCount_p was set to zero.
- New feature: Support of automatically sending of cyclic CAN messages added in wrapper for LabView 8.5 and 8.6. Now multi-channel USB-CANmoduls are supported too.
- New feature: New API function *UcanSetDebugMode()* enables the creation of a debug log file out of the USBCAN-library.
- New feature: Support of Windows Vista (32 and 64 bit) added. Note: GW-001 not longer supported!
- Software change: performance improvements for sysWORXX USB-CANmoduls.
- Software change: Predefined baud rate values of 10kBit/sec, 20kBit/sec, 50kBit/sec and 100kBit/sec for sysWORXX modules changed. "Sync Jump Width" (SJW) of these baud rate values is set to 1 for better synchronization.
- Software change: Default path for creating LOG files from USBCAN32.DLL was set to "Documents".

Software Version 4.01

- Bugfix: for USB-CANmodul GW-001 or GW-002, the USB-CANmodul Control sometimes showed that they use the USB-CANnetwork Driver although it does not apply to those modules; only the Device Driver can be used.
- Bugfix: Library/API function UcanGetStatus() or UcanGetStatusEx() always returned "no error" in driver version V4.00 although a CAN bus error was detected.
- Bugfix: In Windows XP, a “guest” user account was only able to start one application that used USBCAN32.DLL.
- Bugfix: Disconnecting a USB-CANmodul while it was still being used in an application sometimes caused a blue screen on Windows XP with driver version V4.00.
- New feature: USB-CANmodul Control automatically checks if a new driver version is available from the SYS TEC homepage. This feature can be activated by the user upon installation or in tabsheet "Update" in the USB-CANmodul Control.
- New feature: a new context menu has been added in the USB-CANmodul Control in tabsheet "Hardware".

Software Version 4.02

- Bugfix: If API function UcanResetCanEx() was called with flag parameter USBCAN_RESET_NO_BUFFER_COMM, not all CAN messages were sent from a multi-channel USB-CANmodul.
- Bugfix: If API functions UcanWriteCanMsg() and UcanResetCanEx() were often called alternately from one application, it was likely that one of those functions would hang up.
- Software change: the Setup.exe under Windows 2000/XP/Vista only installs the new driver if beforehand all USB-CANmodules are disconnected from the computer and an older driver version is removed.

Software Version 4.03

- Bugfix: USB-CANmodul Control V4.02 always indicated that a driver update would be available from the SYS TEC homepage although this was not the case.
- Software change: For installing a new driver using update of USB-CANmodul Control the Setup.exe will not be called automatically but the download folder is opened in Windows Explorer. The user has to start Setup. exe manually.

Software Version 4.04

- Bugfix: the CAN Traffic LED did not work on all sysWORXX USB-CANmoduls with firmware version V4.03.

Software Version 4.05

- New feature: Support for new module versions.
- New feature: Driver support for Windows CE 6.0 on x86 CPU is available.

Software Version 4.06

- New feature: the Watchdog supervision has been implemented in the firmware of sysWORXX USB-CANmoduls.
- New feature: A new separate bootloader has been implemented in the firmware of sysWORXX USB-CANmoduls. It enables the downgrade to earlier driver versions. To use this feature the module has to be connected to a Windows 2000/XP/Vista PC with the driver version 4.06 before. The bootloader is always started after the USB-CANmodul has been connected. The bootloader notifies the computer about the firmware version that is programmed. If necessary, it programs new firmware into a different flash area and starts this firmware. This implies the advantage that in case of an incorrect firmware update the module must not be sent back because the bootloader is not overwritten.
- New feature: When starting the application, a dialog box is shown if debug feature USBCAN32.DLL is activated. The Debug Info Dialog of USBCAN32.DLL can be switched off with USB-CANmodul Control's Debug tabsheet using check box "Show Dbg Info".
- Software change: New constants for CAN bit rate defined in USBCAN32.DLL for sysWORXX USB-CANmoduls. These new constants sets the sample point above 85%. Using BTR0/BTR1 to set the CAN bit rate (dwBaudrate == 0) the new sample points are enabled. PCANView enables the new sample points too. New 32-bit baudrate values have been defined for the sysWORXX USB-CANmoduls. For those, the sample point is between 85 and 90%. If baudrate values BTR0/BTR1 are used for sysWORXX USB-CANmoduls, the new sample points will be activated - also if tool PCANView is used. For user applications, the new 23-bit baudrate values must be replaced to use the new sample points.

Software Version 4.07

- Bugfix: USB-CANmodul2 with order number 3204019 could not read back the correct /ERR pin state of the low speed CAN transceiver TJA1054.

Software Version 4.08

- Bugfix: For USB-CANmoduls that use the USB-CANnetwork driver, if the acceptance filter was set to receive only one CAN-ID, other CAN-IDs were received as well.
- Bugfix: USBCANCE.dll for Windows CE was not able to write a debug log file.

Software Version 4.09

- New feature: PCANView.exe can be called with command line parameters to select the device number, bit rate, channel and window position.
- New feature: PCANView is able to configure cyclical CAN messages that are automatically sent by the hardware of all sysWORXX modules.

Technical Data:

- CAN interface:
  - Meets the CiA DS 102 Standard and ISO 11898-2/3
  - Optically isolated voltage supply (available as option)
  - GW-001 only: 2 jumpers for configuration of the CAN voltage supply (optically isolated via CAN bus, or via USB bus)
  - Connection to the CAN bus via DB-9 plug
  - Supports CAN frame format according to specifications for CAN 2.0A and 2.0B (11- and 29- bit CAN identifier)
  - Standard version with Philips PCA82C251 CAN transceiver, other variants available on request, i.e. low-speed and single-wire transceivers
  - Further CAN transceivers: 82C252, TJA1054, TJA1041, AU5790
  - GW-002 only: Connector for adapting other CAN transceivers by user (e.g. B10011S)
  - optional power supply via CAN bus, depending on CAN transceiver (see ordering number)
  - intermediate buffer for 768 CAN messages (fix value) in each direction on the USB-CANmodul
  - intermediate buffer for 4096 messages in each direction on the PC (changeable since software version 3.05)
- USB interface:
  - USB connector type B in accordance to the USB standard
  - Power supply through the USB bus (max. 200mA in operating mode) for GW-001, GW-002, USB-CANmodul1 and USB-CANmodul2
  - Transmission type: Bulk, 12MBit/s
- Power (green) and status LED (red) for GW-001 and GW-002
- Power (yellow), status LED (red) and traffic LED (green) for all sysWORXX USB-CANmoduls
- 8 bit expansion port (only GW-002, 3204002 and 3204003)
- Operating temperature 0℃...+55℃ for GW-001 and G W-002
- Operating temperature 0℃...+85℃ for all sysWORXX modules
- Conforms to CE standard
- Optional mounting accessories for DIN rail and wall assembly

Scope of Delivery

- Assembled and tested device
- Systems Manual
- Software (tools, demos in source, driver software)

- USB cable (type A to type B; approximately 1.5 meters)

Software Support:

- Kernel-Mode driver for Windows 2000, XP (32 bit edition, since driver V4.00 64 bit edition too) and Vista (32 and 64 bit version since driver version 4.00):
    - USBCANLD.SYS, USBCANL2.SYS, USBCANL3.SYS, USBCANL4.SYS and USBCANL5.SYS for automatic firmware download to the USB-CANmodul
    - USBCAN.SYS supports the various functions of the USB-CANmodul
    - UCANNET.SYS (network driver) realizes the use of a USB-CANmodul by up to 6 applications
- User-Mode driver for Windows 2000, XP (32 bit edition, since driver V4.00 64 bit edition too) and Vista (32 and 64 bit version since driver version 4.00):
- USBCAN.DLL for easy use of the USB-CANmodul functions
- Up to 64 CAN-channels (corresponds to i.e. 64 USB-CANmodul1 or 32 USB-CANmodul2)
- Tools for Windows 2000, XP (32 bit edition, since driver V4.00 64 bit edition too) and Vista (32 and 64 bit version since driver version 4.00):
    - ***USB-CANmodul Control*** – administration and configuration of more than one USB-CANmodul by allocation of device numbers
    - ***PCANView(USBCAN)*** – CAN monitor program
- Demo programs in source (Microsoft C/C++ using MFC and Microsoft Visual Basic .NET)
- Contributor drivers for LabView
- Device driver and demo software for Linux (Kernel 2.6)
- Device driver and Demo for Windows CE

1 Getting Started

What you will learn in this Getting Started section:

- Installing the USB-CANmodul
- Software installation
- Connecting the USB-CANmodul to the host-PC
- Connecting the USB-CANmodul to a CAN network
- using PCANView (USBCAN)

1.1 Installation

1.1.1 Installation of the USB-CANmodul under Windows-OS

Ensure that the individual components are not damaged. The contents of the USB-CANmodul are:

- USB-CANmodul
- Installation CD-ROM with electronic version of this Systems Manual and all software and drivers
- USB cable

1.1.2 Software Installation

Note:

Installation of the software and operation of the USB-CANmodul on Windows 98/Me is not possible any longer!

Installation and operation of the USB-CANmodul requires a host-PC with a USB port that is running Microsoft Windows 2000, XP and Vista (since driver version 4.00). The USB-CANmodul will not work under Windows NT because there is no USB support in this operating system.

Note:

Make sure to install the software before the USB-CANmodul is connected to the PC. Make sure that you are logged in Windows-OS with admin rights. We provide driver updates for downloading under:

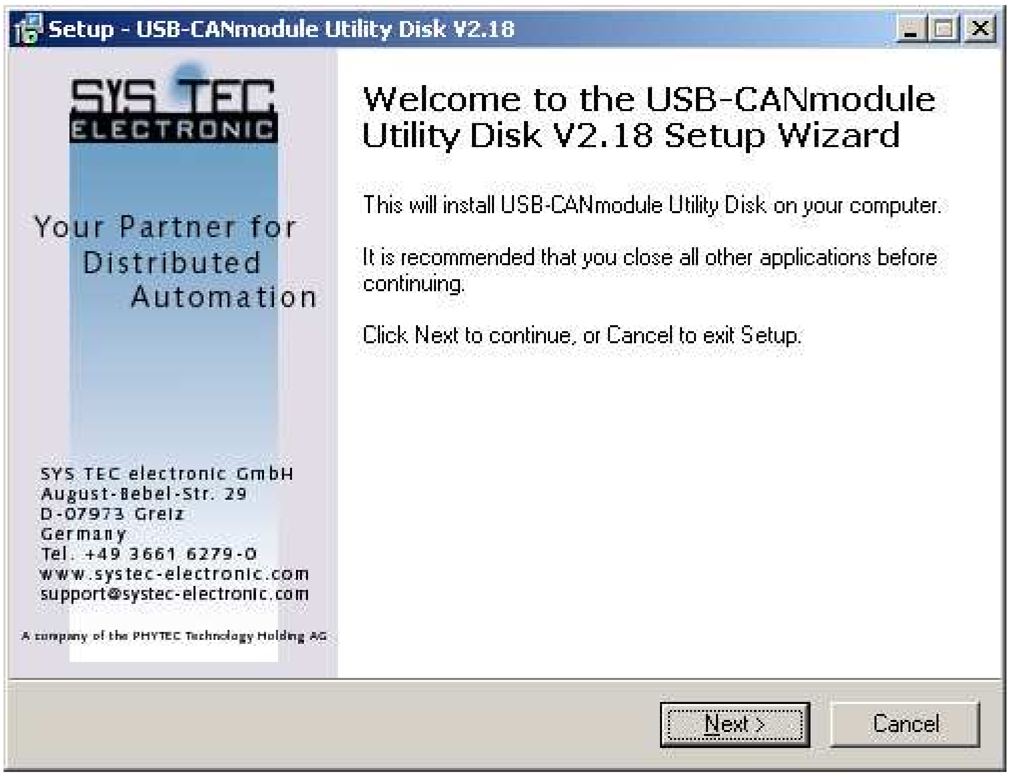
http://www.systec-electronic.com

- Start your computer.

- Insert the **USB-CANmodul Utility CD-ROM** in your CD-ROM drive.
- Open the Windows Explorer
- Go to path:

  "<CD-ROM>:\Products\USB-CANmodul_xxxxxx\Software\SO-387 ".

  Execute file SO-387.exe, which will start the setup tool.
- Click on ***"OK"*** to start the setup program. The following window will appear:



Setup - USB-CANmodule Utility Disk V2.18

Welcome to the USB-CANmodule Utility Disk V2.18 Setup Wizard

This will install USB-CANmodule Utility Disk on your computer.

It is recommended that you close all other applications before continuing.

Click Next to continue, or Cancel to exit Setup.

SYS TEC ELECTRONIC
Your Partner for Distributed Automation
SYS TEC electronic GmbH
August-Bebel-Str. 29
D-07973 Greiz
Germany
Tel. +49 3661 6279-0
www.systec-electronic.com
support@systec-electronic.com
A company of the PHYTEC Technology Holding AG
Next >
Cancel

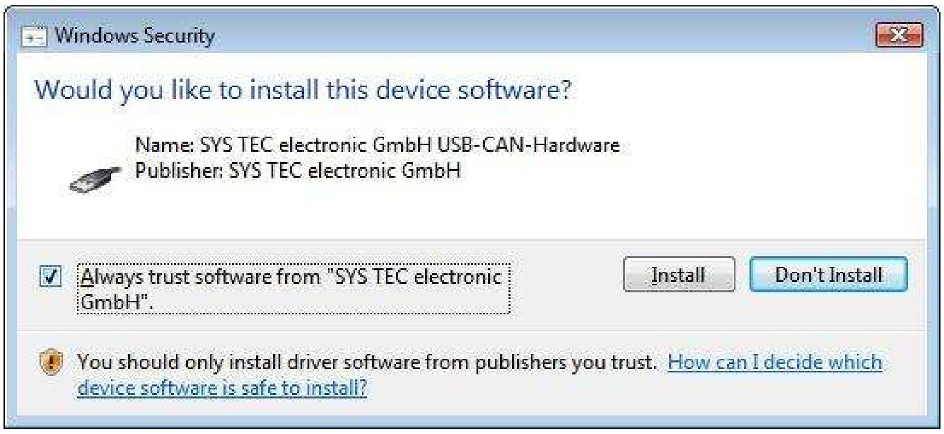
- Click *Next*. Accept the License Agreement in the next window and click *Next* again.
- In the next windows you select the destination location of the USB-CANmodul software and the type of installation you wish to perform *(Full Installation is recommended)*.
- Follow the setup instructions to install the USB-CANmodul software and click *Finish* at the end of the process.
- Connect the USB-CANmodul to your computer using the included USB cable.
- **Windows automatically detects** the USB-CANmodul. The appropriate driver files will be found automatically *(see Note below)*. The firmware will now be downloaded to the USB-CANmodul. The red status LED blinks with a frequency of 10 hertz to indicate this procedure.

- After successful download of the device firmware the red status LED will stay on.

Note:

The USB-CANmodul device driver does not have the Microsoft signature. Because of this an error message will appear when using Windows XP operation system that the driver didn't pass the loop test. Ignore this message and click on *Continue Installation*.

Since 64 Bit Edition of Windows Vista all Kernel Mode Drivers has to be shipped with an certificate which identifies the manufacturer of the driver. Installing the driver for the first time a windows appears as shown in Figure 1. Please tick the box for always trusting the software from company SYS TEC electronic GmbH.


Windows Security

Would you like to install this device software?

Name: SYS TEC electronic GmbH USB-CAN-Hardware
Publisher: SYS TEC electronic GmbH

[✓] Always trust software from "SYS TEC electronic GmbH".

You should only install driver software from publishers you trust. How can I decide which device software is safe to install?

Figure 1: Installation of the driver at Windows Vista

1.1.3 Updating an Existing Installation

Follow the steps below if you have an existing USB-CANmodul installation and just want to update the driver:

- Start your computer.
- Open the Windows Explorer
- Go to path:

  "<CD-ROM>:\Products\USB-CANmodul_xxxxxx\Software\SO-387 ".
- Execute file SO-387.exe, which will start the setup tool.Follow the setup instructions to install the USB-CANmodul software and click *Finish* at the end of the process.Connect the USB-CANmodul to your computer using the included USB cable.

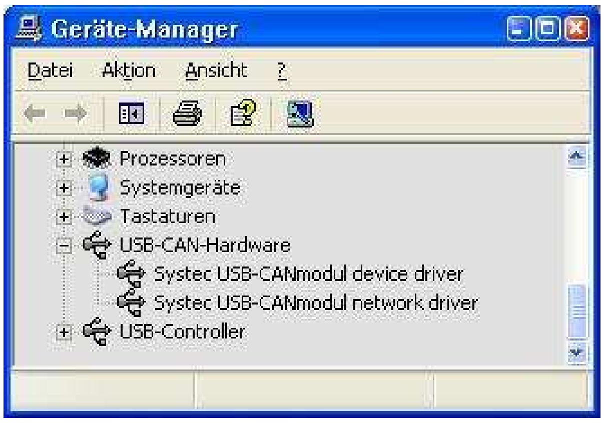
1.1.4 Verifying the Device Installation

Verification of correct device installation on your host-PC can be done by following the steps listed below:

- Highlight the icon "My Computer" on the desktop and click the right mouse key.
- A pop-up menu appears. Click on "Properties". The dialog box "System Properties" appears.
- Choose the tab "Device Manager" at the top. In Windows 2000,XP and Vista the device manager is located in the **"Hardware"** register card.
- Click on the pull-down menu "**Universal Serial Bus Controller**" resp. "**USB Controller**" resp. "**USB-CAN-Hardware**". If the device **"Systec USB-CANmodul device driver" or "Systec USB-CANmodul network driver"** is shown in the list, the new USB device has been detected properly. This is shown in the figure below.

Note:

Starting with version 2.16 of the installation program, the USB-CANmodul will appear in the device manager under the entry "**USB-CAN-Hardware**" and no longer under the entry "**USB Controller**" after setup is completed.



Prozessoren
Systemgeräte
Tastaturen
USB-CAN-Hardware
Systec USB-CANmodul device driver
Systec USB-CANmodul network driver
USB-Controller

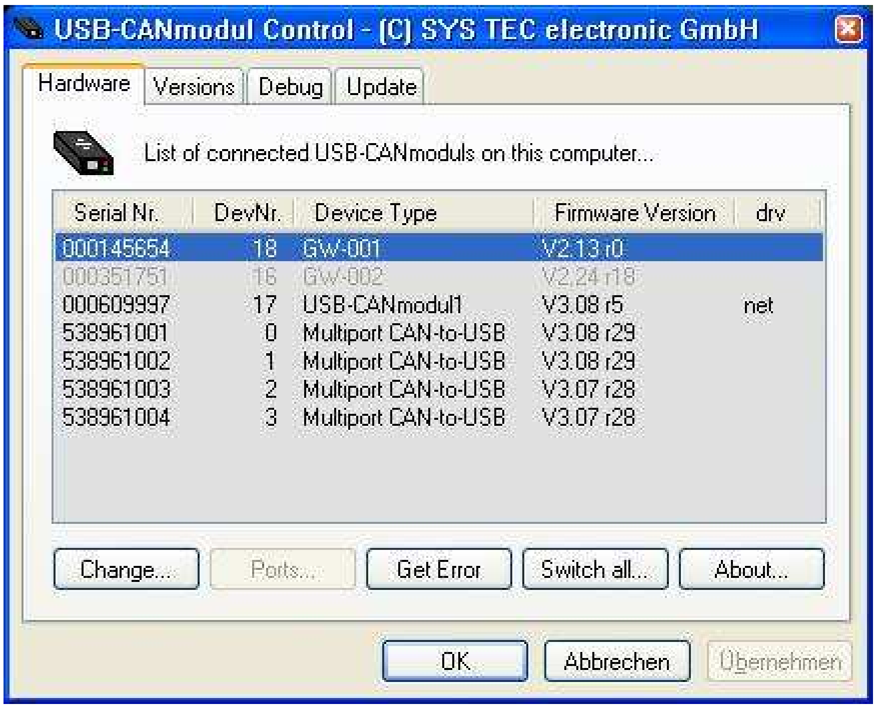
Figure 2: Device Manager with the USB-CANmodul

If the installation was not successful, check the installation steps as described above and try to re-install.

1.1.5 Device Number Allocation

With the help of device number allocation, it is possible to use more than one USB-CANmodul simultaneously on the host-PC. The device number identifies the individual USB-CANmodul.

- Click on **Start → Settings → Control Panel**. Using the Category View in Windows XP additionally click to **Other Control Panal Options** – in Windows Vista use **Additional Options**. In 64 Bit edition of Windows XP or Vista click to **View 32-bit Control Panel Items**.
- Click on the **USB-CANmodul Control** symbol. The following window will appear:



List of connected USB-CANmoduls on this computer...

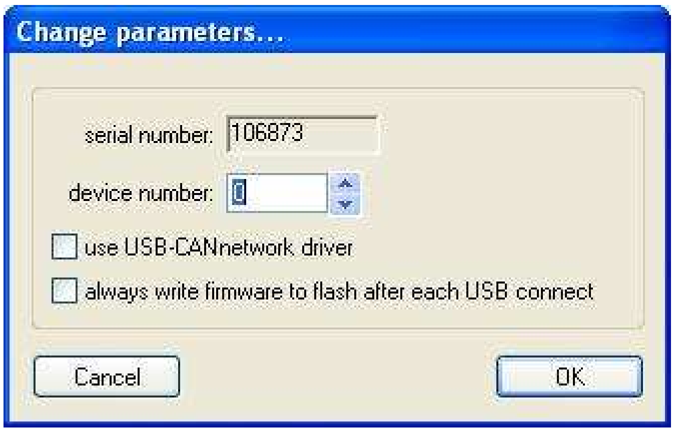
Serial Nr.	DevNr.	Device Type	Firmware Version	drv
000145654	18	GW-001	V2.13 r0	
000351751	16	GW-002	V2.24 r18	
000609997	17	USB-CANmodul1	V3.08 r5	net
538961001	0	Multiport CAN-to-USB	V3.08 r29	
538961002	1	Multiport CAN-to-USB	V3.08 r29	
538961003	2	Multiport CAN-to-USB	V3.07 r28	
538961004	3	Multiport CAN-to-USB	V3.07 r28	

Figure 3: USB-CANmodul Control Tool

- Select/highlight one of the modules shown in the hardware list and then click on the **Change...** button.

Note:

The device number of USB-CANmoduls grayed out in the list cannot be changed because they are used by other applications.


Change parameters...

serial number: 106873

device number: 0

☐ use USB-CANnetwork driver

☐ always write firmware to flash after each USB connect

Figure 4: Device Number Selection Dialog Box

- Enter a new device number in the input field or modify the device number using the Up or Down button. Click **OK** to exit this window.
- The new device number will only take affect and gets downloaded into the device after clicking the **Apply** or **OK** button.

1.1.6 Connection to a CAN Network

The USB-CANmodul provides a DB-9 plug for connection to the CAN network. The pin assignment on this connector is in accordance to the CiA (CAN in Automation) specification. Connect your CAN network to this connector with an appropriate CAN bus cable. The pinout is described in *Table 3* on page *34*.

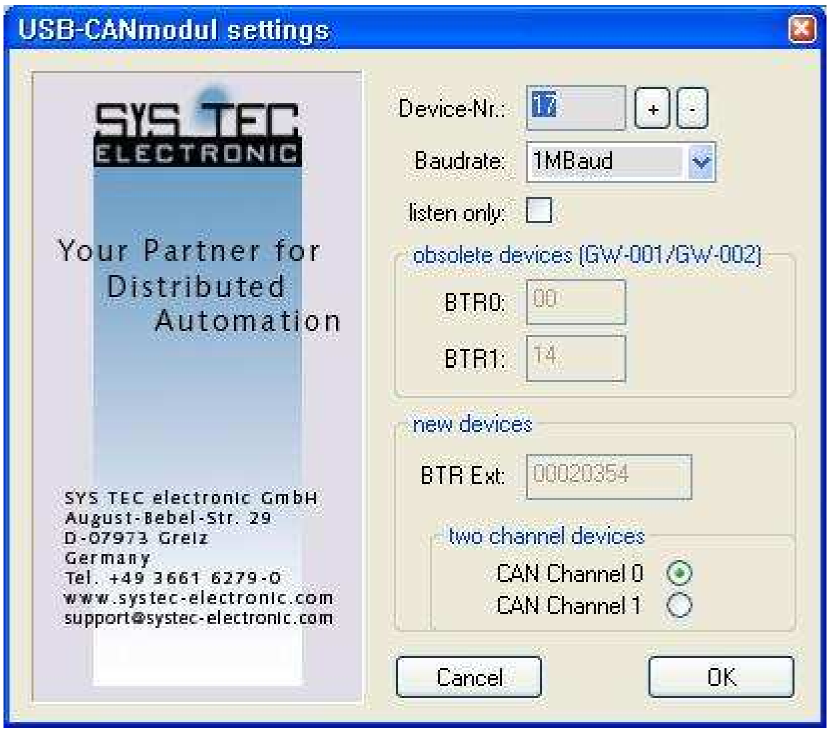
Note:

When using the standard version of the GW-002 with on-board high-speed CAN transceivers (82C251) a termination resistor of 120 Ohms at both ends of the CAN cable between CAN_L (pin 2) and CAN_H (pin 7) is required to ensure proper signal transmission. When using a special version of the device featuring a low-speed CAN transceiver (e.g. TJA1054 etc.) no terminating resistor must be used because it is already integrated in the device. It is necessary to use shielded cables if the CAN bus extension exceeds 3 meters.

1.1.7 Starting PCANView (USBCAN)

The included program PCANView (USBCAN) is a CAN bus monitor for Windows.

- Start the utility program using the Windows *Start* button and browse to *Programs → USB-CANmodul Utilities → PCANView (USBCAN)*. It is recommended that you drag the PCANView (USBCAN) icon onto the desktop of your PC. This enables easy start of this utility program by double-clicking on the icon.
- The *USB-CANmodul settings* window will appear:

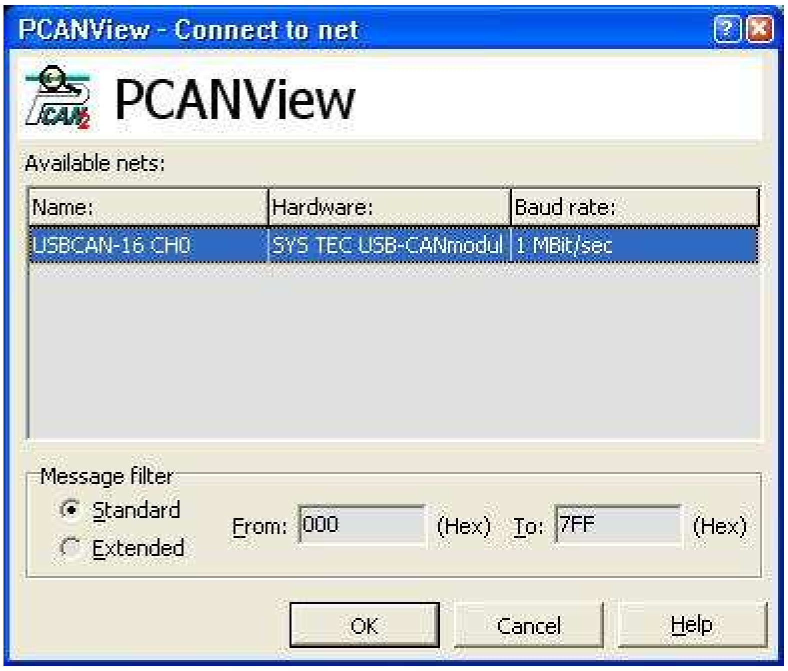


Device-Nr.: 17
Baudrate: 1MBaud
listen only: ☐
BTR0: 00
BTR1: 14
BTR Ext: 00020354
CAN Channel 0 ●
CAN Channel 1 ○

Figure 5: Dialog Box with Hardware Configuration

- Select the baud rate of your CAN network in the *Baudrate* box and the *Device Number*. The entry ***any*** selects the USB-CANmodul that is found first by Windows.
- If "user" is selected in the baud rate field, then the values for registers BTR0 and BTR1 of the SJA1000 can be entered directly. The SJA1000 operates with a 16 MHz clock speed. Refer to the SJA1000 manual for calculation of values for other baud rates.
- When using a sysWORXX USB-CANmodul please enter the user-specific baud rate into field “BTR Ext” (*also see Section 2.3.4*) and select the CAN-channel you want to use.
- Click on the *OK* button to enable these settings.

- A new window *PCANView – Connect to net* will appear.



PCANView - Connect to net

PCANView

Available nets:

Name:	Hardware:	Baud rate:
USBCAN-16 CH0	SYS TEC USB-CANmodul	1 MBit/sec

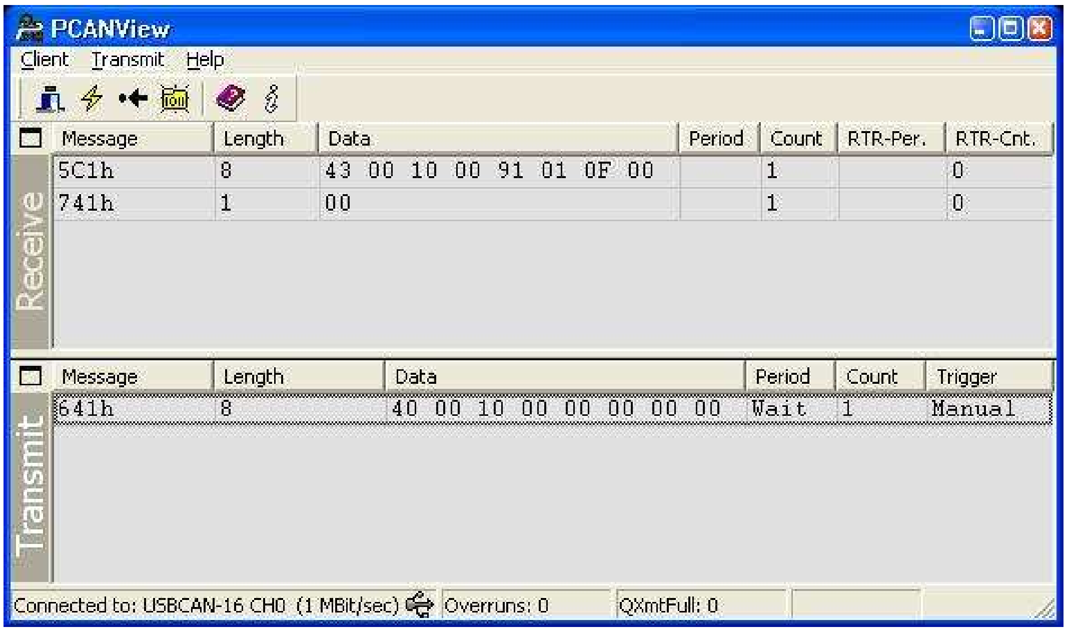
Message filter

(● Standard) ○ Extended   From: 000 (Hex)   To: 7FF (Hex)

Figure 6: Dialog Box Message Filter Configuration

- In this message box you can select Standard (11-bit) or Extended (29-bit) CAN identifier and message filters, if desired. Click on the *OK* button to enable these settings.

- The PCANView (USBCAN) main window will appear:



Message	Length	Data	Period	Count	RTR-Per.	RTR-Cnt.
5C1h	8	43 00 10 00 91 01 0F 00		1		0
741h	1	00		1		0

Message	Length	Data	Period	Count	Trigger
641h	8	40 00 10 00 00 00 00 00	Wait	1	Manual

Connected to: USBCAN-16 CH0 (1 MBit/sec) Overruns: 0 QXmtFull: 0

Figure 7: PCANView (USBCAN) Main Window

This screen is divided into two sections: Receive and Transmit

- Receive: monitors CAN signals that are received from a node
- Transmit: monitors CAN signals sent from the host-PC to the CAN network via the USB-CANmodul

Note:

The tool PCANView is not suitable for sending precisely timed CAN messages to the CAN bus by using the USB-CANmodul.

Since Software version V4.09, in PCANView it is possible to configure cyclic CAN messages which are automatically sent by the firmware of the USB-CANmodul. This feature can be used for instance when precisely timed CAN messages have to be sent to the CAN bus (e.g. SYNC messages). For this purpose, the system menu of PCANView includes a command "SYSTEC cyclic CAN messages" (*see Figure 8*). A dialog box opens up where all cyclic CAN messages can be configured (*see Figure 9*). Use button **Add** to add a new cyclic CAN message. With button **Edit** a previously marked cyclic CAN message can be edited. Each sysWORXX USB-CANmodul supports up to 16 cyclic CAN messages. Choose option **parallel** if the cycle time of each CAN message should refer to itself (*see Figure 18*). With option **sequential** the cycle time of each CAN message refers to its subsequent CAN message (*see Figure 19*).

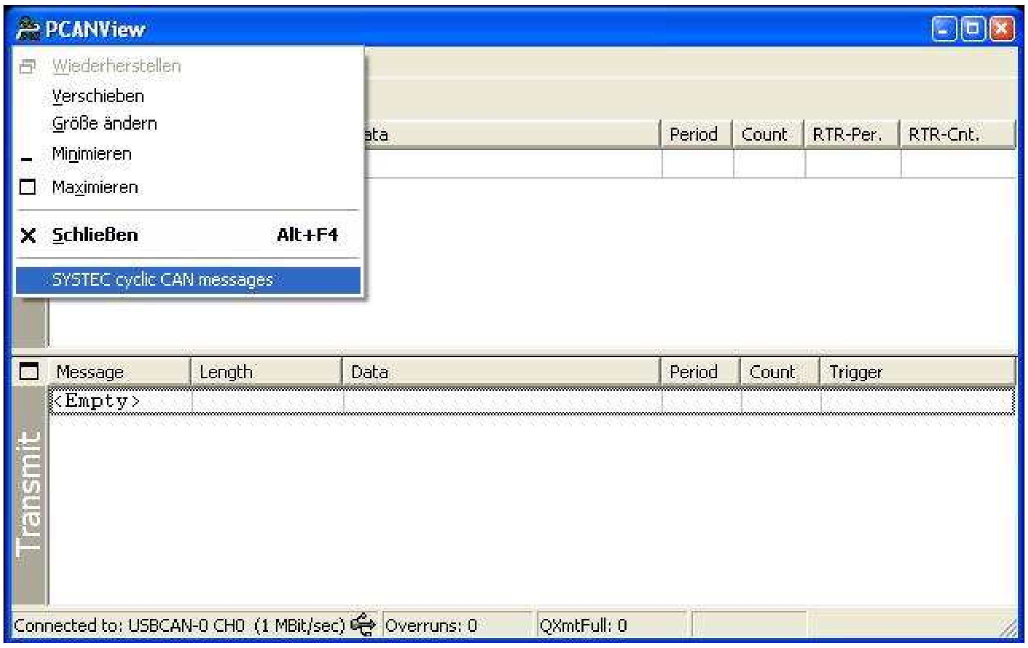
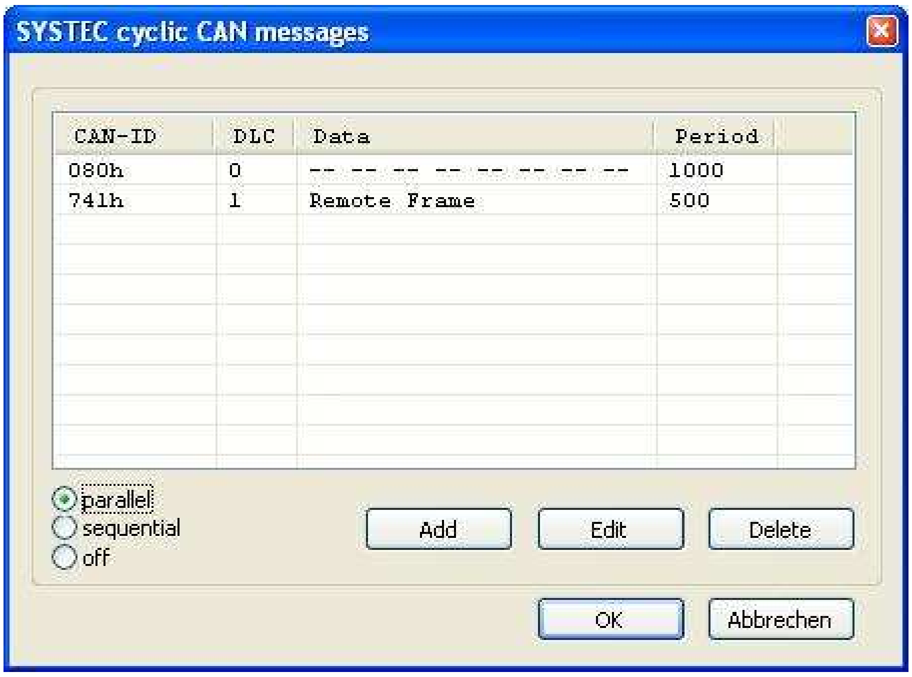


Figure 8: precisely timed CAN messages in PCANView (USBCAN)



CAN-ID	DLC	Data	Period
080h	0	-- -- -- -- -- -- -- --	1000
741h	1	Remote Frame	500

[✓ parallel] sequential off

Figure 9: configuration of cyclic CAN messages in PCANView (USBCAN)

Since Software version V4.09 it is possible to call PCANView by using command line parameters e.g. for using a batch file. If at least one of these command line parameters is used then the dialog box for hardware parameters is not shown (*see Figure 5*).

The following command line parameters are available in PCANView:

Syntax:

```
PCANView.exe [-d<devicenr>] [-c<channelnr>] -b<baudrate> [-l]
             [-x<x-pos> -y<y-pos>] [-n]
```

-d	<devicenr> defines the device number of the USB-CANmodul. The range of values is between 0 and 254. Any USB-CANmodul that is found first is allocated value 255 (default value).
-c	<channelnr> defines the CAN channel that is to be used for multi-channel USB-CANmoduls. The range of values is between 0 and 1. Default value is 0 (first CAN channel).
-b	<baudrate> defines the bit rate on CAN bus in kBit/sec. Possible values are 1000, 800, 500, 250, 125, 100, 50, 20, 10. There is no default value for this parameter. It must be set if the dialog box for setting hardware parameters shall not be shown.
-l	If this parameter is set, the USB-CANmodul will be initialized in listen-only mode. In this case CAN messages cannot be sent.
-x,-y	<x-pos> and <y-pos> define the position of the PCANView main window. If those parameters are missing, the position of the PCANView main window is read from the registry and is the same for all instances.
-n	If this parameter is set, the dialog box to configure the message filter will be skipped (*see Figure 6*). In this case all CAN messages will always be received.

Example:

```
PCANView.exe –d1 -b1000 –n
```

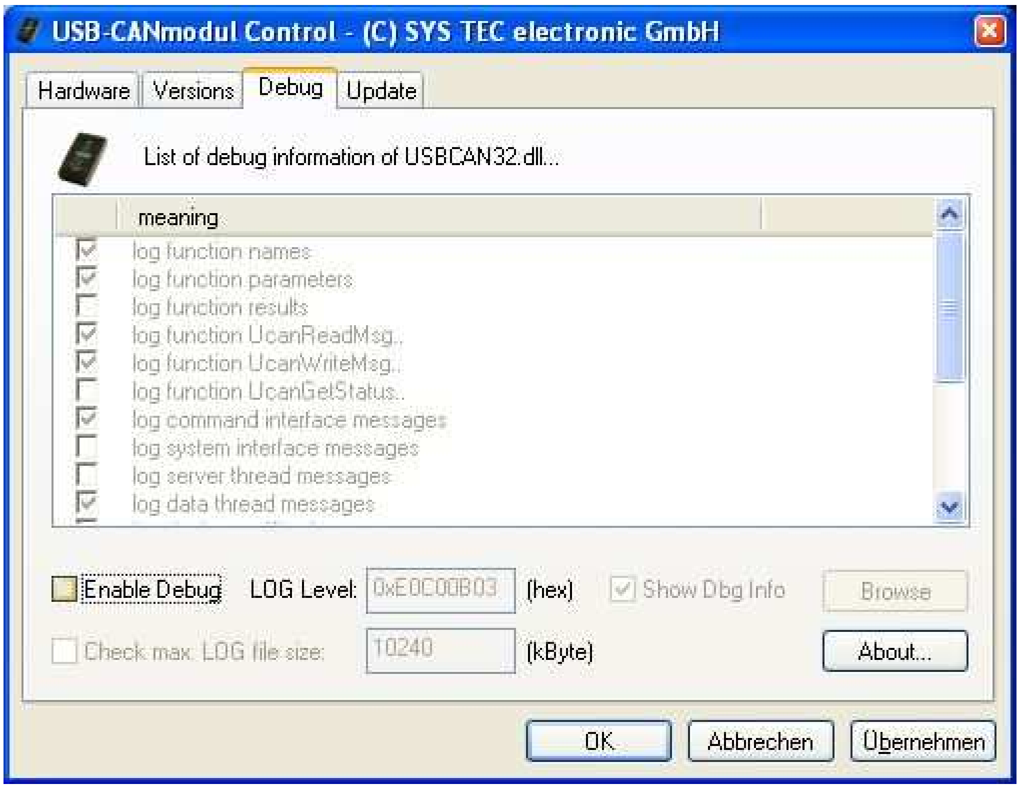
The following command allows for starting more than one instance of PCANView from a batch file. To run the batch file, it is not necessary to wait until the previous PCANView instance is closed:

```
start PCANView.exe –d1 -b1000 –n –x20 –y35
start PCANView.exe –d2 -b1000 –n –x600 –y460
```

1.1.8 Creating a debug file from DLL

If problems with the software drivers should occur, there is a possibility to create a debug log file from USBCAN32.DLL. You should always send this log file to our support email address so that we can find a solution for your problem.

To activate the feature please open **USB-CANmodul Control** from the control panel. At the tab sheet **Debug** you will find the following window:


USB-CANmodul Control - (C) SYS TEC electronic GmbH
Hardware
Versions
Debug
Update
List of debug information of USBCAN32.dll...
meaning
Enable Debug
LOG Level:
(hex)
Show Dbg Info
Browse
Check max. LOG file size:
(kByte)
About...
OK
Abbrechen
Übernehmen

☑ log function names
☑ log function parameters
☐ log function results
☑ log function UcanReadMsg..
☑ log function UcanWriteMsg..
☐ log function UcanGetStatus..
☑ log command interface messages
☐ log system interface messages
☐ log server thread messages
☑ log data thread messages

☐ Enable Debug — LOG Level: 0xE0C00B03 (hex) — ☑ Show Dbg Info

☐ Check max. LOG file size: 10240 (kByte)

Figure 10: Debug settings in USB-CANmodul Control

Enable the feature by ticking the box “**Enable Debug**”. In the list above you can activate different debug information that should be added to the debug log file. Click to “Browse” for choosing the folder in which the debug log file should be stored to. The default setting is the "Documents" folder.

Apply the new settings and close USB-CANmodul Control. Start your application using an USB-CANmodul and wait until the problem will occur. After this close your application.

Afterwards, you will find a file named USBCAN_XXXXXXXX_YYYYYY_ZZZ.LOG. XXXXXXXX represents the creation date of the log file in format YYYYMMDD (year month day) and YYYYYY stands for the creation time in format HHMMSS (hour minute second). ZZZ is the name of the application executed.

Note:

Enabling this feature decreases the performance of the software because API functions have to execute much more code to generate debug outputs. Limitating the debug information by changing the LOG-Level can help to increase performance again. But note that in this case important information could be missing in the log file.

Furthermore, the debug log file may increase in size. Activate the feature “Check max. LOG file size”. This way, USBCAN32.DLL will monitor the file size of the debug log file. If it is exceeded, the previous (older) debug outputs will be deleted from the debug log file. Default setting of the maximum debug file size is 10240 Kbytes (means 10 Mbytes).

Since version V3.11 of USBCAN-library, an application can call the function *UcanSetDebugMode()* for subsequent activation of the feature. Refer to *section 2.3.2.1* for more information.

1.1.9 Activation of the network driver

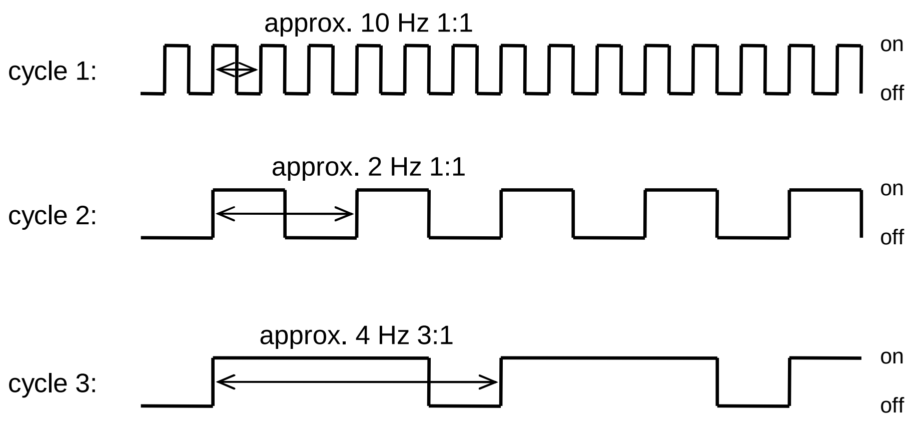
The network driver UCANNET.SYS was developed for connecting several applications to one physical USB-CANmodul. Therefore, the kernel mode driver creates a virtual CAN network for each physical module to which several applications can connect to. All CAN messages that are sent by an application are not only sent to the physical CAN bus but also to all the other connected applications. Received CAN messages are passed on to all applications.

The network driver can only be activated for sysWORXX USB-CANmoduls – but not for the older modules GW-001 and GW-002.

To activate the network driver for an USB-CANmodul, open the USB-CANmodul Control from the Control Panel. Mark that module within the hardware list that you want to use for the network driver. Push the button "**Change…**" to open the dialog box shown in Figure 4. Tick the box "**use USB-CANnetwork driver**" and confirm with "**OK**". After pushing the button "**Apply**" or "**OK**" in the main window of the USB-CANmodul Control, the USB-CANmodul automatically reconnects to the host PC. This results in exchanging the kernel mode driver. Now you can use several applications with this USB-CANmodul.

1.2 Status LEDs on the USB-CANmodul

The state of each CAN-channel on the USB-CANmodul is displayed via 2 resp. 3 LEDs. In order to distinguish the states, different blinking cycles were defined respectively.



(Not to scale)

A description of the power and status LEDs is shown in the table below:

USB-CANmodul connected?	LED green (Power)	LED red (Status)	Description
no	off	Off	No voltage is supplied to the device.
yes	on	Blinking cycle 1	Device logs in to the host-PC
yes	on	On	Log-in successful, CAN is not initialized, no error.
yes	on	Off	CAN is initialized, no error.
yes	on	Blinking cycle 2	A CAN-bus error occurred on the device.

Table 1: States of the LEDs on the USB-CANmodul GW–001/GW–002

On the Multiport CAN-to-USB each CAN-channel has one Status-LED. Furthermore, each channel has a Traffic-LED (green), indicating CAN-bus traffic, once the channel has been initialized. The Multiport CAN-to-USB, USB-CANmodul2, USB-CANmodul8 and USB-CANmodul16 have the same LED assignment. On USB-CANmodul1 there is only one traffic LED, as there is only one channel available. *Table 2* contains a list of all LED states.

USB-CANmodul connected?	**LED yellow (Power)**	**LED red (Status)**	**Description**
no	off	off	No voltage is supplied to the device.
no	on	blinking cycle 1	USB cable not connected.
yes	on	blinking cycle 1	Device logs in to the host-PC.
yes	on	on	Log-in successful, CAN-channel is not initialized, no error.
yes	on	off	CAN-channel is initialized, no error.
yes	on	blinking cycle 2	A CAN-bus error occurred on this channel.
yes	on	blinking cycle 3	Firmware update running. The device must not be powered-off or disconnected while the firmware update is running.

Table 2: States of the LEDs on the sysWORXX modules

All sysWORXX modules also have a green traffic LED for each CAN-channel. While it is blinking it shows an active traffic on CAN bus.

1.3 CAN Supply Voltage

No external CAN supply voltage is necessary for the standard version GW-002 or the GW-002-xx0 versions. The low-speed versions GW-002-xx1 and GW-002-xx2 require an external supply voltage for the CAN transceiver. Be sure to note the limitations for the CAN transceivers when connecting the external supply voltage.

The pin assignment for the DB-9 CAN plug is shown in the table below:

Pin	**Pinout of DB-9 plug**	
	with 82C251, 82C252, TJA1041, TJA1054 (differential)	with AU5790 (single wire)
1	N/C	N/C
2	CAN-L	N/C
3	GND	GND
4	N/C	N/C
5	CAN shield	CAN shield
6	GND	GND
7	CAN-H	CAN-H
8	N/C	N/C
9	Vcc (+7 to +30 VDC)*	Vcc (+5.3 to +13 VDC)*

Table 3: Pinout of the CAN DB-9 Plug

Note:

The value for Vcc depends on the alternative CAN transceiver that populates the device.

For the standard low-speed version (GW-002-xx2, *refer to section 1.7*) an input voltage between 12V and 30V can be supplied at pin 9 (VCC). The nominal voltage amounts to 24V +/-25%. A temporary maximum voltage of up to 35V is allowed. The CAN transceiver starts functioning with supply voltages as low as 8V. The GW-002-xx2 version features an internal protective circuit and a voltage reduction circuit for the input voltage. This means that when supplying the device from an external 12V (+/-20%) source at VCC, the CAN transceiver's supply voltage CANVBAT can drop below 8V. In this case recognition of the standby mode can not be guaranteed.

We recommend using the GW-002-xx1 version in 12V systems. This version is specifically designed for an external 12V voltage. It has no extra circuitry for supply voltage reduction; hence the CAN transceivers standby mode recognition will function. The USB-CANmodul in the GW-002-xx1 version can also operate at 24V (+/-20%). Implementation in 24V systems is possible, but not recommended. Use the GW-002-xx2 instead.

1.4 CAN-port with Low-Speed CAN Transceiver

The high-speed CAN transceiver Philips 82C251 is implemented in the standard configuration of the device. As an alternative, other CAN transceiver can be populated on the USB-CANmodul. In this case only the behavior on the CAN bus changes, not the behavior in relation to the software. From the software point of view (e.g. using the included PCANView) any transceiver can be used.

The optional low-speed transceivers TJA1054 or the single wire transceiver AU5790 have multiple signals for setting the operating mode of the transceivers and displaying the operating state. The following signals are supported:

Signal	Name	Meaning	Type	Default value
EN	Enable	turn-on signal	high-active	high level
/STB	Standby	turn-off signal	low-active	high level
/ERR	Error	error signal	low-active	high level
TRM	Termination	termination resistor	high-active	low level

Table 4: Signals available for low-speed CAN port

Note:

It is only possible to read the state of the termination resistor.by software using USB-CANmodul2.

The standard levels are set so that the transceivers function in normal operating mode. Thus operation with the PCANview tool is possible immediately. The Error signal is not evaluated. Functions for setting the operating modes and for reading the Error signal are supported by the USBCAN-library and are described in the section on software support (*refer to section 2*).

Please refer to the data sheet for the CAN transceiver in question when setting the operating mode. The AU5790 does not have an error output.

An additional pin header connector in 2.54 mm pitch (male or female) is provided for support of additional CAN transceivers such as the B10011S.

Resistors with 1 kOhm are populated at pins RTL or RTH when using the TJA1054. When using the AU5790 device a 9.1 kOhm resistor at space R_t is used and a 220 pF capacitor at C_{ul}.

This CAN port connector has the following pinout:

Signal	Pin	Pin	Signal
/STB	1	2	EN
/ERR	3	4	SPLIT
CAN_RX	5	6	CAN_TX
CAN_5V	7	8	CAN_GND
INH	9	10	CAN_LX
CAN_HX	11	12	CANVBAT
RTH	13	14	RTL

Table 5: CAN Port Pin Assignment for External Transceiver on the GW-002

1.5 Expansion Port

The USB-CANmodul features an 8-bit port for functional expansion which can be used to add digital inputs (e.g. push buttons) and digital outputs (e.g. LEDs) to the device. An additional 2*5-pin header connector in 2.54 mm pitch (male or female) is provided on the USB-CANmodul. The connector has the following pinout:

Signal	Pin	Pin	Signal
PB0	1	2	PB1
PB2	3	4	PB3
PB4	5	6	PB5
PB6	7	8	PB7
GND	9	10	Vcc Output

Table 6: Expansion Port Pin Assignment on the GW-002 and USB-CANmodul2

The microcontroller's port pins are connected directly to the expansion port. Make sure that external circuitry connected to this port does not exceed the maximum load tolerance of the corresponding port pins! The port pins can be configured to be used as inputs or outputs. The 5V supply voltage DC5V is turned on only after the CAN interface in the USB-CANmodul is initialized (following the function call of *UcanInitCan()* or *UcanInitCanEx()*). External circuitry supplied by this voltage should not draw more than 2mA current in order to not destroy the microcontroller.

Please do not hesitate to contact us for additional hardware and software implementation support. The following figure depicts the positions of the connectors and sockets. A detailed diagram is available on request.

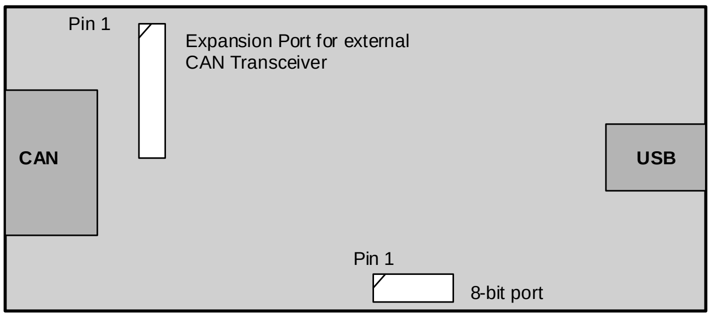


Figure 11: Location of CAN and Expansion Port on GW-002

Symbol	Parameter	Condition	min.	max.	Unit
V_{IH}	Input High Voltage		2.0	5.25	V
V_{IL}	Input Low Voltage		-0.5	0.8	V
V_{OH}	Output High Voltage	I_{OUT} = 1.6 mA	2.4		V
V_{OL}	Output Low Voltage	I_{OUT} = -1.6 mA		0.4	V
C_{IN}	Input Pin Capacity			10	pF
V_{CC}	Supply Voltage		4.75	5.25	V

Table 7: Expansion Port Signal Properties on GW-002

Functions for expansion port access are described in *section 2.3.2*.

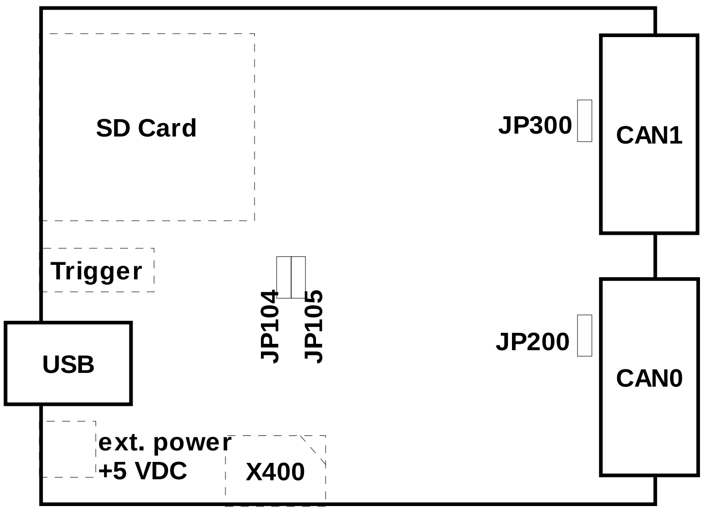


Figure 12: Location of the Expansion Port on USB-CANmodul2

The pinout of the Expansion Port X400 on USB-CANmodul2 is described in *Table 6*. Please note that pin 1 is located at the upper right corner of X400. This connector is not build in on all modules!

Symbol	Parameter	Condition	min.	max.	Unit
V_{IH}	Input High Voltage		2.0	5.5	V
V_{IL}	Input Low Voltage		-0.3	0.8	V
V_{OH}	Output High Voltage	I_{OUT} = 2 mA	2.9		V
V_{OL}	Output Low Voltage	I_{OUT} = 2 mA		0.4	V
C_{IN}	Input Pin Capacitance			14.1	pF
I_{OUT}	Output Current			2.0	mA
V_{CC}	Supply Voltage		3.2	3.4	V

Table 8: Expansion Port Signal Properties on USB-CANmodul2

A user circuit of the Expansion Port depends on the necessity to which level the hardware of USB-CANmodul has to be protected against destruction. You find an example of a user circuit without protection in the next figure.

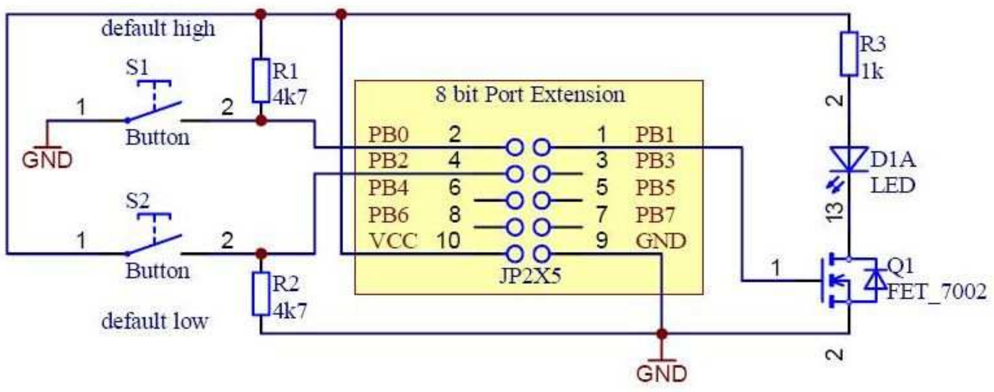


Figure 13: simple example circuit for Expansion Port

Please note that if Vcc is used as power supply for your circuit, the total current of an USB device may not exceed 500 mA (during plug-in the total current actually may not exceed 100 mA). If bus powered USB hubs are used, there could be problems even below 500 mA. Some USB hubs share its power supply with the number of available USB ports. Please note that there could also be problems below 500 mA if other USB devices are connected to these ports. Thus, we advice to implement a galvanic decoupled circuit that has its own power supply.

1.6 Termination resistor for CAN bus

Please note that there always has to be connected two termination resistors with value 120 Ohms, if you are using a USB-CANmodul with a high-speed CAN transceiver. These has to be connected to both ends of the CAN bus:

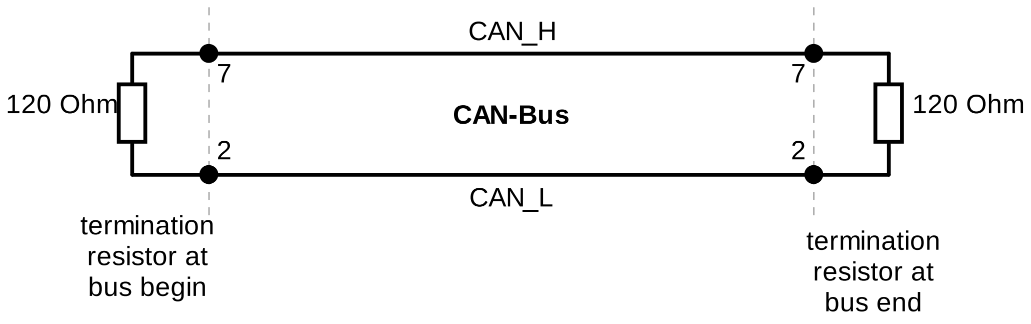

CAN_H
120 Ohm
7
7
CAN-Bus
120 Ohm
2
2
CAN_L
termination resistor at bus begin
termination resistor at bus end

Figure 14: termination resistors on CAN bus

Note:

When using a special version of the device featuring a low-speed CAN transceiver (e.g. TJA1054 etc.) no terminating resistor must be used because it is already integrated in the device.

On USB-CANmodul2, USB-CANmodul8, USB-CANmodul16 and Multiport CAN-to-USB a termination resistor with 120 Ohms is already build in for each CAN-channel. You can enable or disable it by closing a jumper (USB-CANmodul2) or by switching a switch on front panel (USB-CANmodul8, USB-CANmodul16 and Multiport CAN-to-USB). The default state of the termination resistors is: disabled.

If you decide to enable the termination resistor, change the appropriate switch to ON or close the appropriate jumper (*refer to Figure 12* - JP200 for CAN-channel 0; JP300 for CAN-channel 1).

The current state of the termination resistor can be indirectly read back by software only on USB-CANmodul (by calling function *UcanReadCanPort()* or by showing in Control Panel Application USB-CANmodul Control – *refer to Figure 16*). Please note that the jumper JP104 must have the same state like JP200 (for CAN-channel 0) and the jumper JP105 must have the same stat like JP300 (for CAN-channel 1). Otherwise the read state of the termination resistor is not correct. The reason of this solution is the optical isolation of the CAN-channels.

max. cable length [m]	max. bit rate [kBit/s]	specific resistance [kΩ/m]	Cable cross-section [mm²]
30	**1000**	**70**	**0,25..0,34**
100	**500**	**<60**	**0,34..0,60**
500	**100**	**<40**	**0,50..0,60**
1000	**20**	**<26**	**0,75..0,80**

Table 9: recommended cable parameters

1.7 Order Options

Obsolete USB-CANmoduls which are not available any more:

Part Number	***Option***
GW-002	Standard version, high-speed (82C251)
GW-002-x0x	high-speed with Philips 82C251 transceiver
GW-002-x1x	low-speed with Philips TJA1054 transceiver
GW-002-x2x	low-speed, single-wire with Philips AU5790*
GW-002-x3x	low-speed with Philips TJA1041 transceiver
GW-002-0xx	without optical CAN signal isolation
GW-002-1xx	with optical CAN signal isolation
GW-002-xx0	internal supply via USB
GW-002-xx1	external supply 7 - 27V**
GW-002-xx2	external supply 12 - 30V**
GW-002-KSMxx	customer-specific version, MOQ=25

* AU5790 requires external supply voltage

** External supply not available with standard 82C251 transceiver

The USB-CANmodul is available in different options:

Part Number	***Option***
3004006	Multiport CAN-to-USB 16 CAN-channels, high-speed transceiver 82C251, galvanic isolation, separated into 8 logical devices with 2 channels each
3204000	USB-CANmodul1 One CAN-channel, high-speed transceiver 82C251
3204001	USB-CANmodul1 with galvanic isolation One CAN-channel, high-speed transceiver 82C251
3204002	USB-CANmodul2 Two CAN-channels, high-speed transceiver 82C251
3204003	USB-CANmodul2 with galvanic isolation Two CAN-channels, high-speed transceiver 82C251
3204007	USB-CANmodul2, same as 3204000 but with 8 bit Expansion Port
3204008	USB-CANmodul2, same as 3204003 but with single-wire CAN transceiver (Philips AU5790) at the first CAN channel.
3204009	USB-CANmodul2, same as 3204002 but with low-speed CAN transceiver (Philips TJA1054) at the first CAN channel.

Part Number	***Option***
3204011	USB-CANmodul2, same as 3204002 but with low-speed CAN transceiver (Philips TJA1054) at both CAN channels.
3204017	USB-CANmodul2, same as 3204003 but without housing and with wired LEDs.
3204019	USB-CANmodul2, same as 3204003 but with high-speed CAN transceiver (NXP TJA1054) at th first CAN channel.
3404000	USB-CANmodul8 with galvanic isolation 8 CAN-channels, high-speed transceiver 82C251
3404001	USB-CANmodul16 with galvanic isolation 16 CAN-channels, high-speed transceiver 82C251

Currently available order numbers:

GW-002, GW-002-010, GW-002-021, GW-002-030, GW-002-100, GW-002-110, GW-002-121, GW-002-130, GW-002-142, GW-002-150, 3004006, 3204000, 3204001, 3204002, 3204003, 3204007, 3204008, 3204009, 3204011, 3204017, 3204018, 3204019, 3404000, 3404001

Other accessories:

WK054	Unshielded CAN bus cable for max. 5 nodes, with removable 120 Ohm terminating resistors and configured for supply voltage input
WK-004	Shielded CAN cable for direct connection of 2 nodes with integrated 120 Ohm terminating resistors
GW-002-Z01	Wall mounting plate
GW-002-Z02	DB-9 to 5-pin Combicon pin adapter, pinout according to *DeviceNet* specification
GW-002-Z03	USB cable 3 m (A-B)
GW-002-Z04	USB cable 4.5 m (A-B)
GW-002-Z05	Mounting plate for DIN rail

1.8 The new sysWORXX USB-CANmoduls

1.8.1 The Multiport CAN-to-USB

The Multiport CAN-to-USB 3004006 is an industrial USB-CAN interface with 16 CAN-channels coming in a 19” rack mounted housing. The device is structured into 8 logical USB/CAN devices with 2 CAN-channels each. The logical devices are combined by 2 USB-hubs and connected to the PC via two USB ports (see picture below).

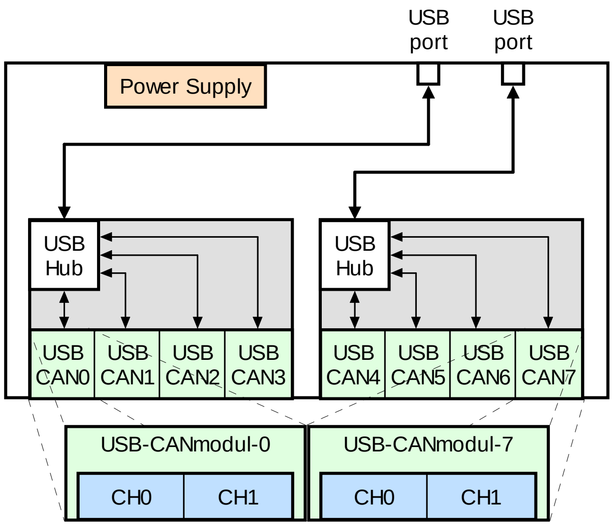


Figure 15: Internal structure of the Multiport CAN-to-USB

There is no separate software driver for the Multiport CAN-to-USB as it is supported by the standard drivers used for USB-CANmodul. A special API function set was implemented to support the extended functions of the Multiport CAN-to-USB, such as multiple CAN-channels, baud rate configuration and acceptance mask filtering. Please also refer to *sections 2.3.4, 2.3.5 and 2.3.7*. In a limited scope these extended function are also applicable to GW-002 devices and the standard functions are applicable to the Multiport CAN-to-USB.

The USB device numbers of the 8 logical devices are assigned sequentially. The first logical device (counted from left side) device number 0, the second logical device has number 1 and so on. The device numbers can be reconfigured using the "USB-CANmodul Control" icon in the Windows Control Panel.

1.8.2 The USB-CANmodul1

The USB-CANmodul1 (ordering number 3204000 or 324001) is a cost optimized variant of the new sysWORXX USB-CANmodul series including only one CAN-channel. Optionally you can order this device with or without a galvanic isolation (*refer to section 1.7*). Both variants has built in a high-speed CAN transceiver. There is no Expansion Port for connecting digital inputs or outputs.

1.8.3 The USB-CANmodul2

The USB-CANmodul2 (ordering number 3204002 or 324003) is a extended variant of the new sysWORXX USB-CANmodul series including two CAN-channels. Optionally you can order this device with or without a galvanic isolation (*refer to section 1.7*). Both variants has built in a high-speed CAN transceiver. There is Expansion Port for connecting digital inputs or outputs like the GW-002 does have too. With order number 3204007 you will get an USB-CANmodul2 including an Expansion Port which is described in *section 1.5*.

1.8.4 The USB-CANmodul8 and USB-CANmodul16

Both USB-CANmodul8 (ordering number 3404000) and USB-CANmodul16 (ordering number 3404001) are identical to the Multiport CAN-to-USB but are shipped with a table case. The USB-CANmodul16 consists of two circuit cards of the same type like is build in on USB-CANmodul8.

2 Software Support for Windows OS

2.1 File Structure

If during the installation of the USB-CANmodul utilities no other target path is given, then all files will be installed in the folder C:\Programs\SYSTEC-electronic\USB-CANmodul Utility Disk. The contents of this folder are given in *Table 10*. Some folders are created depending on selected installation options during setup process.

Sub Folder	Contents
`Bin\`	Program files (PCANView)
`Contrib\`	Files contributed by other companies
`   Borland Delphi\`	Delphi class with demo in source (is removed)
`   LabView\`	LabView driver with demo
`Demo\`	MFC demo in source for GW-002 and MS Visual Studio 6.0 or higher
`DemoGW006\`	MFC demo in source for a USB-CANmodul including two CAN-channels and MS Visual Studio 6.0 or higher
`DemoCyclicMsg\`	MFC demo in source for MS Visual Studio 6.0 or higher and automatically transmitting of cyclic CAN messages using a sysWORXX USB-CANmodul.
`Docu\`	Manuals
`Drv\`	Windows Kernel drivers
`Include\`	C header files for USBCAN32.DLL. The demo applications for MS Visual Studio 6.0 refer to these files.
`Lib\`	Common USBCAN32.DLL and import- library for MS Visual Studio. The demo applications refer to this import-library.
`UcanDotNET\`	Wrapper-DLL in source code for use with Microsoft .NET projects.
`USBcanDemoNET\`	MS Visual Basic .NET demo application in source code (using the Wrapper-DLL UcanDotNET.dll)

Table 10: Software File Structure

2.2 Tools for the USB-CANmodul

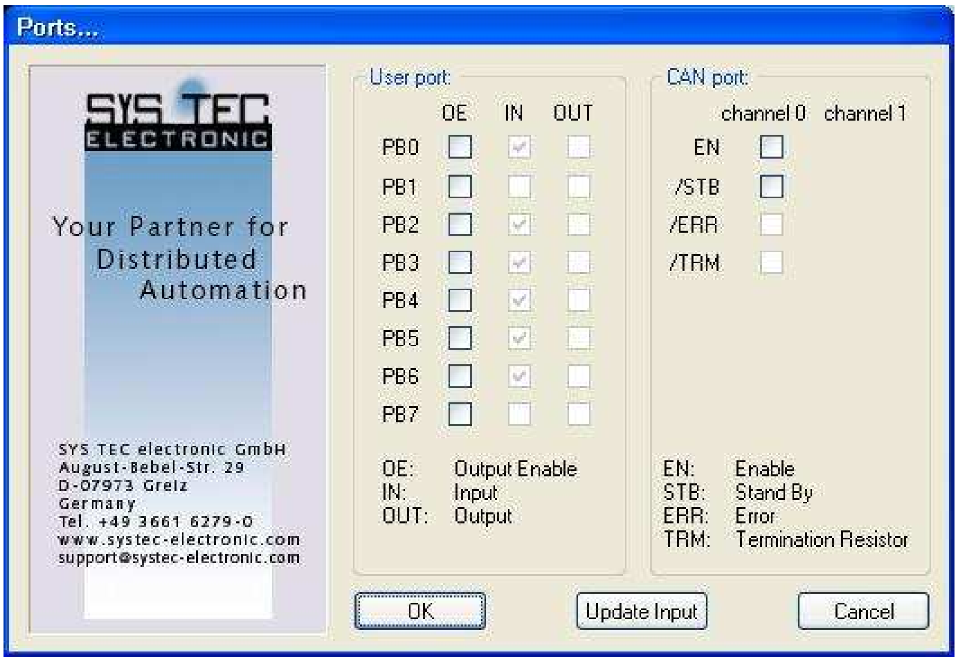
2.2.1 USB-CANmodul Control

The USB-CANmodul Control tool replaces the UCAN Config tool starting at version 2.18. This tool can be started either from the Control Panel or from the program group **"USB-CANmodul Utilities"**. *Figure 3* shows the tool after start up.

This tool can be used to modify the device number of the USB-CANmoduls (*also refer to section 1.1.5*).

In addition, this tool can also be used to manipulate the 8-bit port expansion (*refer to section 1.5*) and the CAN port for low-speed CAN transceivers (*refer to section 1.4*). To do this you have to select the corresponding USB-CANmodul from the list and then click on the "**Ports…**" button.

Figure 16 shows the dialog box that will appear when choosing this option.



SYS TEC electronic GmbH
August-Bebel-Str. 29
D-07973 Greiz
Germany
Tel. +49 3661 6279-0
www.systec-electronic.com
support@systec-electronic.com

	OE	IN	OUT
PB0		✓	
PB1			
PB2		✓	
PB3		✓	
PB4		✓	
PB5		✓	
PB6		✓	
PB7			

	channel 0	channel 1
EN		
/STB		
/ERR		
/TRM		

OE:	Output Enable
IN:	Input
OUT:	Output

EN:	Enable
STB:	Stand By
ERR:	Error
TRM:	Termination Resistor

Figure 16: Dialog Box for Manipulating the Port Expansion and the CAN Port

Initially all 8 signals are configured as inputs. With the column OE, the corresponding signal is switched to an output. This activates the box for the output value in the OUT column. If a signal is switched to a logical 1 in this column, then the corresponding signal on the port expansion will be set to high. With every modification the current state of the expansion port will be read again and shown in the IN column for the inputs. To read the current input states without having to change an output, click on the "**Update Input"** button.

The current state of the CAN port for the low-speed CAN transceiver is displayed on the right side of the window. The signals EN and /STB are outputs and the signal /ERR is an input. For more information refer to *section 1.4*.

2.2.2 PCANView (USBCAN) for Windows

The Windows utility **PCANView (USBCAN)** can be used to display CAN messages transmitted via the CAN bus.

After execution of the tool a dialog box is shown for configuring the hardware parameters (*refer to Figure 5*). The device number of the logical USB-CANmodul has to be filled in to the edit field “Device-Nr.”. This device number was previously programmed with the Windows Control Panel symbol USB-CANmodul Control (*refer to section 1.1.5*). Within the drop down box “Baudrate” the baud rate on CAN bus can be selected. The option “listen only” configures the CAN controller for only receiving CAN messages. This also means that no acknowledge will be sent back to the sending remote CAN device. For a logical USB-CANmodul including two CAN-channels the channel has to be selected which should be used by the tool.

After applying the settings by clicking to the “OK” button a dialog box is shows like displayed in *Figure 6*. The filter setting depends on the CAN message format you whishes to receive: CAN identifier with 11 bits (standard frame = CAN Spec. 2.0A) or CAN identifier with 29 bits (extended frame = CAN Spec. 2.0B). Please choose one of both possibilities and enter the range of the CAN messages which has to be shown on receive section of the tool. If you do not change this range, then all CAN messages will be shown. Apply this setting by clickint to the button “OK”.

The main window of the tool appears (*refer to Figure 7*). This screen is divided into two sections: Receive and Transmit:

- Receive: monitors CAN signals that are received from a node
- Transmit: monitors CAN signals sent from the host-PC to the CAN network via the USB-CANmodul

Receive Section

The *Receive* section provides the following information:

- Message: identifier of the CAN message, hexadecimal format, ranging from 0 to 7FFh for 11-bit identifiers and from 0 to 1FFFFFFFh for 29-bit identifiers
- Length: data length code of the message (ranges from 0 to 8)
- Data: values of the messages’ data bytes (up to 8) or the text Remote request, if a remote frame has been received
- Period: period of time between the reception of the last two messages with this identifier
- Count: number of messages received with this identifier (no remote frames) since last user reset
- RTR-Per.: period of time between the reception for the last two remote frames
- RTR-Cnt.: number of remote frames with this identifier

Transmit Section

The Transmit section provides the following information:

• Message:	identifier of the CAN message, hexadecimal format, ranging from 0 to 7FFh for 11-bit identifiers and from 0 to 1FFFFFFFh fro 29-bit identifiers
• Length:	data length code of the message (ranges from 0 to 8)
• Data:	values of the messages' data bytes (up to 8) or the text *Remote request*, if a remote frame shall be sent
• Period:	period of time between the last two message sent

Note:

If, at creation of the message, the period was set to 0, the text *Wait* is shown. In this case the message can only be transmitted manually using the <Space> bar. Or the message is sent automatically after a matching remote frame has been received.

• Count:	number of messages sent with this identifier (no remote frames) since last user reset
• Trigger:	reason for the last transmission of the message Manual: manual transmission by the user pressing the <Space> bar
• Time:	period of time has passed for periodical sending
• RTR:	remote frame has been received

Note:

Both sections are sorted by the CAN identifiers. That means no chronology is displayed.

In order to edit the *Transmit* list, the following menu commands are available:

• Transmit → New...:	Create a new transmit message. The editor window for the new message is shown.
• Transmit → Delete:	Delete the currently selected message from the transmit list.
• Transmit → Edit...:	Edit the currently selected message.
• Transmit → Clear all:	Delete the entire transmit list.
• Client → Reset:	Reset the message counters and reset the connected USB-CANmodul. Deletes the receive list.

2.3 Description of the USBCAN-library

The USBCAN-library is a function library for application programs. At Windows 2000/XP or higher it is a Dynamic Linked Library (DLL – with the file name USBCAN32.DLL) It serves as an interface between the system driver layer and an application program. The USBCAN-Library for the USB-CANmodul enables easy access to the USB-CAN system driver functions. It administers the opened USB-CANmodul and translates the USB data into CAN messages.

Add the file USBCAN32.LIB to your project for linking the USBCAN32.DLL to your own Microsoft Visual C/C++ project. Starting the application program automatically loads the DLL. If the USBCAN32.LIB is not linked to the project, or you are using another environment (e.g. Borland C++ Builder), load the DLL manually with the Windows function *LoadLibrary()* and add the library functions with the function *GetProcAdress()* (*refer to the demo application "DemoGW-006"*). There was a .NET wrapper DLL implemented for Microsoft .NET applications described in *section 2.4*.

The PUBLIC calling convention of the DLL functions provides a standardized interface to the user. This standard interface ensures that users of other programming languages than C/C++ (Pascal, etc.) are able to use these functions.

Within this manual the DLL is called USBCAN-library because the API functions (also called USBCAN-API) are also implemented for other platforms at which the library has another file name (e.g. "USBCANCE.DLL" under Windows CE) or at which the library is not a DLL (e.g. under Linux).

Folders <SETUP_DIR>\DEMO.API, <SETUP_DIR>\DEMOGW006 and <SETUP_DIR>\DEMOCYCLICMSG contains example programs written using MFC in Microsoft VisualC/C++6.0 and 7.0. These example projects demonstrates the use of the DLL API functions.

2.3.1 Attributes of the USBCAN-library

With USBCAN-library, it is possible to use 64 USB-CANmoduls simultaneously with one application program, as well as with several application programs (using Windows CE only 9 modules). However, it is not possible to use one USB-CANmodul with several application programs.

Three states within the software are generated for each USB-CANmodul when using this DLL.

After starting the application program and loading the DLL, the software is now in the DLL_INIT state. Concurrently, all required resources for the DLL have been created.

Calling the library function *UcanInitHardware()* and/or *UcanInitHardwareEx()* causes the software to change into the HW_INIT state. This state contains all resources required for communication with the USB-CANmodul. It is not possible to transmit or to receive CAN messages in this state.

If the application software calls the library function *UcanInitCan(), UcanInitCanEx()* or *UcanInitCanEx2()* the state changes into CAN_INIT. In this state it is possible to transmit or to receive CAN messages.

Return with the library function *UcanDeinitCan()* into the state HW_INIT and with the library function *UcanDeinitHardware()* into the state DLL_INIT. It is possible to close the application program only after this sequence is completed.

Note:

Make sure to return to the state DLL_INIT before closing the application program.

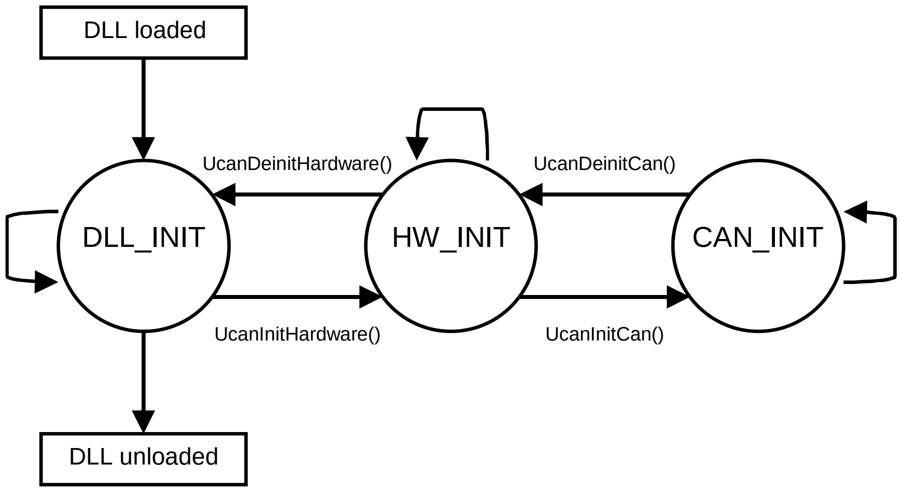


Figure 17: Software State Diagram

The number of functions differs in different software states. For example, the function *UcanWriteCanMsg()* causes an error at the state DLL_INIT. *Table 11* shows the different functions within each state.

If multiple USB-CANmoduls are used in one application, these states have to be considered for each USB-CANmodul that is used. If the first USB-CANmodul is in the state CAN_INIT, the second one can still be in the DLL_INIT state.

State	Functions	overage		sysWORXX	
		GW-001	GW-002	multi-channel	single-channel
DLL_INIT	*UcanSetDebugMode()*	X	X	X	X
	UcanGetVersion()	X	X	X	X
	UcanGetVersionEx()	X	X	X	X
	UcanInitHwConnectControl()	X	X	X	X
	UcanInitHwConnectControlEx()	X	X	X	X
	UcanInitHardware()	X	X	X	X
	UcanInitHardwareEx()	X	X	X	X
	UcanDeinitHwConnectControl()	X	X	X	X
	UcanGetModuleTime()	X	X	X	X
HW_INIT	*UcanGetFwVersion()*	X	X	X	X
	UcanGetHardwareInfo()	X	X	X	X
	UcanGetHardwareInfoEx2()	XH0	XH0	X	XH0
	UcanGetStatus()	X	X	CH0	X
	UcanGetStatusEx()	XH0	XH0	X	XH0
	UcanResetCan()	X	X	CH0	X
	UcanResetCanEx()	XH0	XH0	X	XH0
	UcanInitCan()	X	X	CH0	X
	UcanInitCanEx()	X	X	CH0	X
	UcanInitCanEx2()	XH0	XH0	X	XH0
	UcanWriteCanPort()	-	X	CH0	-
	UcanWriteCanPortEx()	-	XH0	X	-
	UcanReadCanPort()	-	X	CH0	-
	UcanReadCanPortEx()	-	XH0	X	-
	UcanConfigUserPort()	-	X	X	-
	UcanWriteUserPort()	-	X	X	-
	UcanReadUserPort()	-	X	X	-
	UcanReadUserPortEx()	-	X	X	-
	UcanDefineCyclicCanMsg()	-	-	X	XH0
	UcanReadCyclicCanMsg()	-	-	X	XH0
	UcanDeinitHardware()	X	X	X	X
CAN_INIT	*UcanSetTxTimeout()*	-	-	X	-
	UcanSetBaudrate()	X	X	CH0	X
	UcanSetBaudrateEx()	XH0	XH0	X	XH0
	UcanSetAcceptance()	X	X	CH0	X
	UcanSetAcceptanceEx()	XH0	XH0	X	XH0
	UcanReadCanMsg()	X	X	CH0	X
	UcanReadCanMsgEx()	XH0	XH0	X	XH0
	UcanWriteCanMsg()	X	X	CH0	X
	UcanWriteCanMsgEx()	XH0	XH0	X	XH0
	UcanGetMsgCountInfo()	-	X	CH0	X
	UcanGetMsgCountInfoEx()	-	XH0	X	XH0
	UcanEnableCyclicCanMsg()	-	-	X	XH0
	UcanGetMsgPending()	-	-	X	XH0
	UcanGetCanErrorCounter()	-	-	X	XH0
	UcanDeinitCan()	X	X	CH0	X
	UcanDeinitCanEx()	XH0	XH0	X	XH0

Table 11: Software State Functions

Meaning of entries in Table 11:

"-"	Function not supported
"X"	Function supported without limitations
"CH0"	Function supported for each module with one CAN-channel and/or for CAN-channel 0 of a logical module with two CAN-channels, because the function parameter for selecting the channel number is missing.
"XH0"	Function only supported with function parameter selecting CAN-channel 0 of a logical module, because the hardware does only have one CAN-channel.

2.3.2 Functions of the USBCAN-library

This section describes the various functions provided by USBCAN-library. Most of the functions return a value of the type UCANRET containing an error code. The meaning of this code is the same for each function. Besides the syntax, the meaning and the parameters of each function, the possible error codes are shown.

Some of the extended functions have an additional parameter for support of multi CAN instances and enable operations on a single CAN-channel on Multiport CAN-to-USB 3004006 or USB-CANmodul2 3204002/3204003. These extended functions are also applicable on GW-002 or GW-001, as long as CAN cannel 0 is used. Otherwise the functions returns with error code USBCAN_ERR_ILLCHANNEL (*see section 2.3.3*). All standard (single-instance) functions are applicable for Multiport CAN-to-USB 3004006 as well, but do not provide the possibility to access other CAN-channels than CAN 0. If channel other than CAN0 is used, the function returns with error code USBCAN_ERR_ILLCHANNEL (*see section 2.3.3*).

2.3.2.1 General functions

UcanSetDebugMode

Syntax:

```
BOOL PUBLIC UcanSetDebugMode (DWORD dwDbgLevel_p,
     _TCHAR*                 pszFilePathName_p,
     DWORD                   dwFlags_p);
```

Usability:

DLL_INIT, HW_INIT, CAN_INIT since version 3.11

Description:

This function enables the creation of a debug log file out of the USBCAN-library. If this feature has already been activated via the USB-CANmodul Control, the content of the “old” log file will be copied to the new file. Further debug information will be appended to the new file.

Parameter:

dwDbgLevel_p:	Bit mask which enables the activation of debug information to be written into the debug log file. This Bit mask has the same meaning as the “LOG-Level” of the USB-CANmodul Control and therefore is not referred to in detail.
pszFilePathName_p:	Path leading to a text-based file which is written by the USBCAN-library with debug information. This parameter may be set to NULL. In this case only the new value of parameter dwDbgLevel_p will be set.
dwFlags_p:	Additional flag parameter. Value 0 will create a new debug log file. If the file referring to parameter pszFilePathName_p does already exist, the old content will be deleted upon opening. Value 1 though will append all new debug information to an existing file.

Return value:

If FALSE returns, the debug log file could not be created. A possible reason could be that the directory path which is set by the parameter *pszFilePathName_p* does not exist.

Example:

```
// set debug mode for USBCAN API
UcanSetDebugMode (0xE0C00B03L,          // = default Debug-Level
    _T("C :\\MyAppPath\\MyApp.log"),
    0);                                 // = no append mode
```

UcanGetVersion

Syntax:

```
DWORD PUBLIC UcanGetVersion (void);
```

Usability:

DLL_INIT, HW_INIT, CAN_INIT

Description:

This function returns the software version number of the USBCAN-library. It is overage an should not be used in current projects. Use the function *UcanGetVersionEx()* instead of.

Parameter:

none

Return value:

Software version number as DWORD with the following format:

Bit 0 to 7:	least significant digits of the version number in binary format
Bit 8 to 15:	most significant digits of the version number in binary format
Bit 16 to 30:	reserved
Bit 31:	1 = customer specific version

Example:

```
DWORD   dwVersion;
_TCHAR  szVersion[8];
...
// get version number
dwVersion = UcanGetVersion ();

// convert into a string
_stprintf (szVersion, _T(„V%d.%2d“), (dwVersion & 0xff00) >> 8,
    dwVersion & 0xff);
...
```

UcanGetVersionEx

Syntax:

```
DWORD PUBLIC UcanGetVersionEx (tUcanVersionType VerType_p);
```

Usability:

DLL_INIT, HW_INIT, CAN_INIT, version 2.16 and higher only

Description:

This function returns the version numbers of the individual software modules.

Parameter:

VerType_p: Type of version information shows from which software module the version is to be returned. *Table 12* lists all possible values for this parameter. The format of the version information differs from that of the *UcanGetVersion()* function.

VerType_p	Value	Meaning
kVerTypeUserDll kVerTypeUserLib	0x0001	Returns the version of the file USBCAN-library.
kVerTypeSysDrv	0x0002	Reterns the version of the file USBCAN.SYS (device driver).
kVerTypeNetDrv	0x0004	Returns the version of the file UCANNET.SYS (network driver).
kVerTypeSysLd	0x0005	Returns the version of the file USBCANLD.SYS (firmware loader of USB-CANmodul GW-001).
kVerTypeSysL2	0x0006	Returns the version of the file USBCANL2.SYS (firmware loader of USB-CANmodul GW-002).
kVerTypeSysL3	0x0007	Returns the version of the file USBCANL3.SYS (firmware loader of Multiport CAN-to-USB).
kVerTypeSysL4	0x0008	Returns the version of the file USBCANL4.SYS (firmware loader of USB-CANmodul1 3204000 / 3204001).
kVerTypeSysL5	0x0009	Returns the version of the file USBCANL5.SYS (firmware loader of USB-CANmodul2 3204002 / 3204003).
kVerTypeCpl	0x000A	Returns the version of the file USBCANCL.CPL (USB-CANmodul Control from Windows Control Panel).

Table 12: Constants for the type of version information

Return value:

Software version number as DWORD using the following format:

Bit 0-7:	Version (Macro USBCAN_MAJOR_VER)
Bit 8-15:	Revision(Macro USBCAN_MINOR_VER)
Bit 16-31:	Release (Macro USBCAN_RELEASE_VER)

Example:

```
DWORD dwVersion;
_TCHAR  szVersion[16];
...
// Get USBCAN-library version number.
dwVersion = UcanGetVersionEx (kVerTypeUserDll);

// convert into a string.
_stprintf (szVersion, _T(„V%d.%02d.%d“),
    USBCAN_MAJOR_VER(dwVersion),
    USBCAN_MINOR_VER(dwVersion),
    USBCAN_RELEASE_VER(dwVersion));
...
```

UcanGetFwVersion

Syntax:

```
DWORD PUBLIC UcanGetFwVersion (
                            tUcanHandle  UcanHandle_p);
```

Usability:

HW_INIT, CAN_INIT version 2.18 and higher

Description:

This function returns the version number of the software in the USB-CANmodul.

Parameter:

UcanHandle_p:	USB-CAN handle that was received with the function *UcanInitHardware()* or *UcanInitHardwareEx()*.

Return value:

Software version number as DWORD in the following format:

Bit 0-7:	Version	(Macro USBCAN_MAJOR_VER)
Bit 8-15:	Revision	(Macro USBCAN_MINOR_VER)
Bit 16-31:	Release	(Macro USBCAN_RELEASE_VER)

The version number format is the same format as in the function *UcanGetVersionEx()*.

UcanInitHwConnectControl

Syntax:

```
UCANRET PUBLIC UcanInitHwConnectControl (
                        tConnectControlFkt
                        fpConnectControlFkt_p);
```

Usability:

DLL_INIT, HW_INIT, CAN_INIT

Description:

Initializes the supervision for recently connected USB-CANmoduls. If a new module is connected to the PC, the callback function that is indicated in the parameter will be called. This callback function is also called if a module is disconnected from the PC.

Parameter:

fpConnectControlFkt_p:	Address to the callback function that has to be called if a new USB-CANmodul is connected or disconnected. This address may not be NULL!

The callback function must have the following format (*see section 2.3.7*):

```
void PUBLIC UcanConnectControlFkt (
    DWORD                   dwEvent_p,
    DWORD                   dwParam_p);
```

Return value:

Error code of the function.

USBCAN_SUCCESSFUL
USBCAN_WARN_NULL_PTR

UcanInitHwConnectControlEx

Syntax:

```
UCANRET PUBLIC UcanInitHwConnectControlEx (
     tConnectControlFktEx    fpConnectControlFktEx_p,
     void*                   pCallbackArg_p);
```

Usability:

DLL_INIT, HW_INIT, CAN_INIT (version 3.00 and above)

Description:

Initializes the supervision for recently connected USB-CANmoduls. If a new module is connected to the PC, the callback function that is indicated in the parameter will be called. This callback function is also called if a module is disconnected from the PC.

Unlike function *UcanInitHwConnectControl(),* this function has an additional parameter, which is also passed to the callback function. This parameter can be used to handle user-specific information, such as the used CAN instance for example.

Attention:

This function must not be used simultaneously with function *UcanInitHwConnectControl()* within the same application*!*

Parameter:

fpConnectControlFkt_p:	Address to the callback function that has to be called if a new USB-CANmodul is connected or disconnected. This address must not be NULL!
pCallbackArg_p:	User-specific parameter that is passed to the callback function as well.

The callback function must have the following format (*see section 2.3.7*):

```
void PUBLIC UcanConnectControlFktEx (
     DWORD                   dwEvent_p,
     DWORD                   dwParam_p,
     void*                   pArg_p);
```

Return Value:

Error code of the function.

USBCAN_SUCCESSFUL
USBCAN_WARN_NULL_PTR

UcanDeinitHwConnectControl

Syntax:

```
UCANRET PUBLIC UcanDeinitHwConnectControl (void);
```

Usability:

DLL_INIT, HW_INIT, CAN_INIT

Description:

This function finishes the supervision of the recently connected or disconnected USB-CANmoduls. This function must be called after the function *UcanInitHwConnectControl() or UcanInitHwConnectControlEx()* was called within an application and before closing this application.

Return value:

Error code of the function.

USBCAN_SUCCESSFUL

UcanInitHardware

Syntax:

```
UCANRET PUBLIC UcanInitHardware (
     tUcanHandle*             pUcanHandle_p,
     BYTE                     bDeviceNr_p,
     tCallbackFkt             fpCallbackFkt_p);
```

Usability:

DLL_INIT

Description:

Initializes a USB-CANmodul. The software changes into the state HW_INIT. From this point, other functions as they are defined in *Table 11* can be called. If the function was executed successfully, the function transfers a USB-CAN handle to the variable **pUcabHandle_p.* Other functions have to be called with this handle.

Parameter:

pUcanHandle_p:	Pointer to the variable for the USB-CAN Handle. This pointer may not be NULL!
bDeviceNr_p:	Device number of the USB-CANmodul (0 – 254). The value *USBCAN_ANY_MODULE* (= 255) makes sure that the first allocated USB-CANmodul is used.
fpCallbackFkt_p:	Address to the callback function of this USB-CANmodul. This value can be NULL. The callback function will not be called if corresponding events appear. This address can also be same as one that is already used from other USB-CANmoduls, because the callback function contains the associated USB-CAN Handle.

The callback function must have the following format (*see section 2.3.7*):

```
void PUBLIC UcanCallbackFkt (
     tUcanHandle              UcanHandle_p,
     DWORD                    bEvent_p);
```

Return value:

Error code of the function.
USBCAN_SUCCESSFUL
USBCAN_ERR_MAXINSTANCES
USBCAN_ERR_HWINUSE
USBCAN_ERR_ILLHW
USBCAN_ERR_MAXMODULES
USBCAN_ERR_RESOURCE
USBCAN_ERR_ILLVERSION
USBCAN_ERR_ILLPARAM
USBCAN_ERR_IOFAILED
USBCAN_ERR_BUSY
USBCAN_ERR_TIMEOUT
USBCAN_ERR_DATA
USBCAN_ERR_ABORT
USBCAN_ERR_DISCONNECT
USBCAN_ERRCMD_...

Example:

```
UCANRET bRet;
tUcanHandle UcanHandle;

...
// initializes a USB-CANmodul without callback function
bRet = UcanInitHardware (&UcanHandle, USBCAN_ANY_MODULE, NULL);
...
```

UcanInitHardwareEx

Syntax:

```
UCANRET PUBLIC UcanInitHardwareEx (
     tUcanHandle*             pUcanHandle_p,
     BYTE                     bDeviceNr_p,
     tCallbackFktEx           fpCallbackFktEx_p
     void*                    pCallbackArg_p);
```

Usability:

DLL_INIT (version 3.00 and higher)

Description:

Initializes a USB-CANmodul. The software changes into the state HW_INIT. From this point, other functions as they are defined in *Table 11* can be called. If the function was executed successfully, the function transfers a USB-CAN handle to the variable **pUcabHandle_p.* Other functions have to be called with this handle.

Unlike function *UcanInitHardware(),* this function has an additional parameter, which is also passed to the callback function.

Parameter:

pUcanHandle_p:	Pointer to the variable for the USB-CAN Handle. This pointer may not be NULL!
bDeviceNr_p:	Device number of the USB-CANmodul (0 – 254). The value *USBCAN_ANY_MODULE* (= 255) makes sure that the first allocated USB-CANmodul is used.
fpCallbackFkt_p:	Address to the callback function of this USB-CANmodul. This value can be NULL. The callback function will not be called if corresponding events appear. This address can also be same as one that is already used from other USB-CANmoduls, because the callback function contains the associated USB-CAN Handle.
pCallbackArg_p:	User-specific parameter that is passed to the callback function as well.

The callback function must have the following format (*see section 2.3.7*):

```
void PUBLIC UcanCallbackFktEx (
     tUcanHandle              UcanHandle_p,
     DWORD                    bEvent_p,
     BYTE                     bChannel_p,
     void*                    pArg_p);
```

Return value:

Error codes of the function.

USBCAN_SUCCESSFUL
USBCAN_ERR_MAXINSTANCES
USBCAN_ERR_HWINUSE
USBCAN_ERR_ILLHW
USBCAN_ERR_MAXMODULES
USBCAN_ERR_RESOURCE
USBCAN_ERR_ILLVERSION
USBCAN_ERR_ILLPARAM
USBCAN_ERR_IOFAILED
USBCAN_ERR_BUSY
USBCAN_ERR_TIMEOUT
USBCAN_ERR_DATA
USBCAN_ERR_ABORT
USBCAN_ERR_DISCONNECT
USBCAN_ERRCMD_...

UcanDeinitHardware

Syntax:

```
UCANRET PUBLIC UcanDeinitHardware (
     tUcanHandle             UcanHandle_p);
```

Usability:

HW_INIT, CAN_INIT

Description:

Shuts down an initialized USB-CANmodul that was initialized with *UcanInitHardware()* or *UcanInitHardwareEx().* The software returns to the state DLL_INIT. After the function call, the USB-CAN handle is not valid. That means, execution of the valid functions (*see Table 4*) for HW_INIT and CAN_INIT is no longer possible.

Parameter:

UcanHandle_p:	USB-CAN handle received with the function *UcanInitHardware()* or *UcanInitHardwareEx()*.

Return value:

Error code of the function.

USBCAN_SUCCESSFUL
USBCAN_ERR_MAXINSTANCES
USBCAN_ERR_ILLHANDLE
USBCAN_ERR_ILLHW

Note:

This function has to be called before closing the application, otherwise other applications are no longer able to access this specific USB-CANmodul.

UcanGetModuleTime

Syntax:

```
UCANRET PUBLIC UcanGetModuleTime (
     tUcanHandle             UcanHandle_p,
     DWORD*                  pdwTime_p);
```

Usability:

HW_INIT, CAN_INIT version 3.01 or higher

Description:

This function reads the current time stamp from the device.

Parameter:

UcanHandle_p:	USB-CAN-handle, that was returned by *UcanInitHardware()* or *UcanInitHardwareEx()*.
pdwTime_p:	Pointer to a variable where the time stamp is to be stored to.

Return value:

Error code of the function.

USBCAN_SUCCESSFUL
USBCAN_ERR_MAXINSTANCES
USBCAN_ERR_ILLHANDLE
USBCAN_ERR_ILLPARAM
USBCAN_ERR_ILLHW
USBCAN_ERR_BUSY
USBCAN_ERR_IOFAILED
USBCAN_ERR_TIMEOUT
USBCAN_ERR_DATA
USBCAN_ERR_ABORT
USBCAN_ERR_DISCONNECT
USBCAN_ERRCMD_...

Note:

The execution of this function as well as the transfer of the time stamp needs run-time. In other words, after this function has returned successfully, the time stamp might be outdated. The accuracy of this time stamp depends on many factors and is unpredictable on non real-time operating systems.

UcanInitCan

Syntax:

```
UCANRET PUBLIC UcanInitCan (
    tUcanHandle            UcanHandle_p,
    BYTE                   bBTR0_p,
    BYTE                   bBTR1_p,
    DWORD                  dwAMR_p,
    DWORD                  dwACR_p);
```

Usability:

HW_INIT

Description:

Initializes the CAN interface of a USB-CANmodul. The software changes into the state CAN_INIT. Now it is possible to transmit and receive CAN messages. *Table 11* shows the possible functions in this state.

Parameter:

UcanHandle_p:	USB-CAN handle received with the function *UcanInitHardware()* or *UcanInitHardwareEx()*.
bBTR0_p:	Baud rate register 0 (*refer to section 2.3.4*)
bBTR1_p:	Baud rate register 1 (*refer to section 2.3.4*)
dwAMR_p:	Acceptance Mask Register (*refer to section 2.3.5*)
dwACR_p:	Acceptance Code Register (*refer to section 2.3.5*)

Return value:

Error code of the function.

USBCAN_SUCCESSFUL
USBCAN_ERR_MAXINSTANCES
USBCAN_ERR_ILLHANDLE
USBCAN_ERR_RESOURCE
USBCAN_ERR_BUSY
USBCAN_ERR_IOFAILED
USBCAN_ERR_ILLPARAM
USBCAN_ERR_ILLHW
USBCAN_ERR_DATA
USBCAN_ERR_ABORT
USBCAN_ERR_DISCONNECT
USBCAN_ERR_TIMEOUT
USBCAN_ERRCMD_…

UcanInitCanEx

Syntax:

```
UCANRET PUBLIC UcanInitCanEx (
     tUcanHandle             UcanHandle_p,
     tUcanInitCanParam*      pInitCanParam_p);
```

Usability:

HW_INIT, version 2.16 or higher

Parameter:

UcanHandle_p:	USB-CAN handle, that was received with the function *UcanInitHardware()* or *UcanInitHardwareEx()*.
pInitCanParam_p:	Pointer to an initialization structure

```
typedef struct
{
  DWORD  m_dwSize;      // Size of this structure in bytes
  BYTE   m_bMode;       // CAN Transmission Mode
                        // (see able below)
  BYTE   m_bBTR0;       // Baud rate register 0 of the SJA1000
  BYTE   m_bBTR1;       // Baud rate register 1 of the SJA1000
  BYTE   m_bOCR;        // Output control register of the SJA1000
                        // (should always be 0x1A)
  DWORD  m_dwAMR;       // Acceptance filter mask of the SJA1000
  DWORD  m_dwACR;       // Acceptance filter code of the SJA1000
  DWORD  m_dwBaudrate;  // Baudrate register for Multiport,
                        // USB-CANmodul1 and USB-CANmodul2

  // number of entries in receive buffer in USBCAN-library
  WORD   m_wNrOfRxBufferEntries;

  // number of entries in transmit buffer in USBCAN-library
  WORD   m_wNrOfTxBufferEntries;

} tUcanInitCanParam;
```

Note:

The configuration of the baud rate differs significantly between the older USB-CANmodul versions (GW-001 and GW-002) and the new sysWORXX modules. For standardized baud rate values (*see section 2.3.4*), the baud rate registers BTR0 and BTR1 are as well applicable for the new sysWORXX modules. Therefore set *m_dwBaudrate* to USBCAN_BAUDEX_USE_BTR01.

The mode of CAN transmission is configured by an 8-bit bit-mask. The following table shows all possible constants/modes:

Constant	**Value**	**Meaning**
kUcanModeNormal	0x00	normal transmit- and receive mode
kUcanModeListenOnly	0x01	listen-only mode; transmitted CAN messages are not sent out via CAN-bus. Received CAN-messages of remote nodes are not acknowledged.
kUcanModeTxEcho	0x02	*UcanReadCanMsg()* also returns transmitted messages as transmit echo. (see function *UcanReadCanMsg()*) (not available for GW-001)

Table 13: Constants for CAN transmission mode

Description:

Initializes the CAN interface of a USB-CANmodul with expanded parameters. This function works like the function *UcanInitCan()*. However, it should not be called in combination with *UcanInitCan()*.

Return value:

Error code of the function.

USBCAN_SUCCESSFUL
USBCAN_ERR_MAXINSTANCES
USBCAN_ERR_ILLHANDLE
USBCAN_ERR_RESOURCE
USBCAN_ERR_BUSY
USBCAN_ERR_IOFAILED
USBCAN_ERR_ILLPARAM
USBCAN_ERR_ILLHW
USBCAN_ERR_DATA
USBCAN_ERR_ABORT
USBCAN_ERR_DISCONNECT
USBCAN_ERR_TIMEOUT
USBCAN_ERRCMD_…

UcanInitCanEx2

Syntax:

```
UCANRET PUBLIC UcanInitCanEx2 (
    tUcanHandle            UcanHandle_p,
    BYTE                   bChannel_p
    tUcanInitCanParam*     pInitCanParam_p);
```

Usability:

HW_INIT version 3.00 and higher

Parameter:

UcanHandle_p:	USB-CAN-handle, that was received with function *UcanInitHardware()* or *UcanInitHardwareEx()*.
bChannel_p:	CAN-channel, which is to be initialized. USBCAN_CHANNEL_CH0 for CAN-channel 0 USBCAN_CHANNEL_CH1 for CAN- channel 1
pInitCanParam_p:	Pointer a structure containing the initialization data

Structure *tUcanInitCanParam* is described with function *UcanInitCanEx().*

Description:

Initializes the specified CAN-channel of a USB-CANmodul. For GW-001 and GW-002 only CAN-channel 0 can be initialized. Use this function alternatively for function *UcanInitCanEx()*.

Return value:

Error codes of the function.

USBCAN_SUCCESSFUL
USBCAN_ERR_MAXINSTANCES
USBCAN_ERR_ILLHANDLE
USBCAN_ERR_RESOURCE
USBCAN_ERR_BUSY
USBCAN_ERR_IOFAILED
USBCAN_ERR_ILLPARAM
USBCAN_ERR_ILLHW
USBCAN_ERR_DATA
USBCAN_ERR_ABORT
USBCAN_ERR_DISCONNECT
USBCAN_ERR_TIMEOUT
USBCAN_ERRCMD_…

UcanSetTxTimeout

Syntax:

```
UCANRET PUBLIC UcanSetTxTimeout (
    tUcanHandle            UcanHandle_p,
    BYTE                   bChannel_p,
    DWORD                  dwTxTimeout_p);
```

Usability:

CAN_INIT since version 3.10, only for multi-channel modules

Description:

Is this function called with a timeout value bigger than 0 milliseconds then firmware controls all transmit CAN messages by this timeout. If a CAN message cannot be sent during this timeout then firmware changes to a special state whereas all further transmit CAN messages for the specified channel will be deleted automatically. At each deleted transmit CAN message firmware sets the new CAN driver state USBCAN_CANERR_TXMSGLOST. When the CAN message could be sent later then firmware leaves this special state.

This feature is to prevent that transmit CAN messages of a channel blocks transmit CAN messages of the other channel caused by not connected remote CAN device or any physical problems on CAN bus.

Parameter:

UcanHandle_p:	USB-CAN handle received with the function *UcanInitHardware()* or *UcanInitHardwareEx()*.
bChannel_p:	CAN-channel for setting the timeout USBCAN_CHANNEL_CH0 for channel 0 USBCAN_CHANNEL_CH1 for channel 1
dwTxTimeout_p:	Transmission Timeout in milliseconds. The value 0 switches off the timeout control.

Return value: ***Error code of the function.***

USBCAN_SUCCESSFUL
USBCAN_ERR_MAXINSTANCES
USBCAN_ERR_ILLHANDLE
USBCAN_ERR_CANNOTINIT
USBCAN_ERR_BUSY
USBCAN_ERR_IOFAILED
USBCAN_ERR_ILLHW
USBCAN_ERR_ILLCHANNEL
USBCAN_ERR_DATA
USBCAN_ERR_ABORT
USBCAN_ERR_DISCONNECT
USBCAN_ERR_TIMEOUT
USBCAN_ERRCMD_…

UcanResetCan

Syntax:

```
UCANRET PUBLIC UcanResetCan (tUcanHandle UcanHandle_p);
```

Usability:

HW_INIT, CAN_INIT

Description:

Resets the CAN controller in the USB-CANmodul and erases the CAN message buffer. This function needs to be called if a BUSOFF event occurred. Starting at version 2.17 a CAN status error (readable via *UcanGetStatus()*) is also cleared.

Parameter:

UcanHandle_p:	USB-CAN handle received with the function *UcanInitHardware()* or *UcanInitHardwareEx()*.

Return value:

Error code of the function.

USBCAN_SUCCESSFUL
USBCAN_ERR_MAXINSTANCES
USBCAN_ERR_ILLHANDLE
USBCAN_ERR_CANNOTINIT
USBCAN_ERR_BUSY
USBCAN_ERR_IOFAILED
USBCAN_ERR_ILLHW
USBCAN_ERR_ILLCHANNEL
USBCAN_ERR_DATA
USBCAN_ERR_ABORT
USBCAN_ERR_DISCONNECT
USBCAN_ERR_TIMEOUT
USBCAN_ERRCMD_…

UcanResetCanEx

Syntax:

```
UCANRET PUBLIC UcanResetCanEx (
     tUcanHandle             UcanHandle_p,
     BYTE                    bChannel_p,
     DWORD                   dwResetFlags_p);
```

Usability:

HW_INIT, CAN_INIT, version 3.00 and higher

Description:

Resets parametered global features of features of a separate CAN-channel of a USB-CANmodul (*see function UcanResetCan()*). For GW-001, GW-002 and USB-CANmodul1 only features of CAN-channel 0 can be reset.

Parameter:

UcanHandle_p:	USB-CAN handle received with the function *UcanInitHardware()* or *UcanInitHardwareEx()*.
bChannel_p:	CAN-channel that is to be reset USBCAN_CHANNEL_CH0 for channel 0 USBCAN_CHANNEL_CH1 for channel 1
dwResetFlags_p:	The flags of this parameter specify what components are to be reset (see list below). The logical combination of different flags is possible.

USBCAN_RESET_ALL 0x00000000:

Reset all components. However, the firmware is not reset completely.

USBCAN_RESET_NO_STATUS 0x00000001:

Skip reset of the CAN error status (not supported for GW-001 and/or GW-002).

USBCAN_RESET_NO_CANCTRL 0x00000002:

Skip reset of the CAN controller.

USBCAN_RESET_NO_TXCOUNTER 0x00000004:

Skip reset of the transmit message counter.

USBCAN_RESET_NO_RXCOUNTER 0x00000008:

Skip reset of the receive message counter.

USBCAN_RESET_NO_TXBUFFER_CH...0x00000010:

Skip reset of the transmit buffers of a specific CAN-channel (CAN-channel is specified by parameter *bChannel_p*).

USBCAN_RESET_NO_TXBUFFER_DLL 0x00000020:

Skip reset of the transmit buffer for both CAN-channels within the DLL.

USBCAN_RESET_NO_TXBUFFER_FW 0x00000080:

Skip reset of the transmit buffers of both CAN-channels within the device's firmware.

USBCAN_RESET_NO_RXBUFFER_CH... 0x00000100:

Skip reset of the receive buffers of a specific CAN-channel (CAN-channel is specified by parameter *bChannel_p*).

USBCAN_RESET_NO_RXBUFFER_DLL 0x00000200:

Skip reset of both receive message counters within the DLL

USBCAN_RESET_NO_RXBUFFER_SYS.. 0x00000400:

Skip reset of the receive message counter of both CAN-channels within the Kernel-Mode driver.

USBCAN_RESET_NO_RXBUFFER_FW 0x00000800:

Skip reset of receive message counters of both CAN-channels within the device's firmware.

USBCAN_RESET_FIRMWARE..0xFFFFFFFF:

Complete reset of the device firmware.

There are the following predefined combinations:

USBCAN_RESET_ONLY_STATUS:... 0x0000FFFE

Reset of the CAN error status only.

USBCAN_RESET_ONLY_CANCTRL:.. 0x0000FFFD

Only resets the CAN controller of the USB-CANmodul. This has to be done after each bus-off state because the CAN controller cannot leave this state automatically.

USBCAN_RESET_ONLY_RXBUFFER_FW:...0x0000F7FF

Only resets the receive buffer within the firmware of the USB-CANmodul.

USBCAN_RESET_ONLY_TXBUFFER_FW:..0x0000FF7F

Only resets the transmit buffer within the firmware of the USB-CANmodul.

USBCAN_RESET_ONLY_RXCHANNEL_BUFF: 0x0000FEFF

Reset of the receive buffer of only one CAN-channel.

USBCAN_RESET_ONLY_TXCHANNEL_BUFF:... 0x0000FFEF

Reset of the transmit buffer of o nly one CAN-cannel.

USBCAN_RESET_ONLY_RX_BUFF: 0x0000F0F7

Reset of the receive buffers in all software parts and reset of the receive message counter.

USBCAN_RESET_ONLY_TX_BUFF:..0x0000FF0B

Reset of the transmit buffers in all software parts and reset of the transmit message counter.

USBCAN_RESET_ONLY_ALL_BUFF:... 0x0000F003

Reset off all message buffers (receive and transmit buffers) in all software parts and reset of the receive and transmit message counter.

USBCAN_RESET_ONLY_ALL_COUNTER: 0x0000FFF3

Reset of the receive and transmit counter.

Important:

If the constants USBCAN_RESET_NO_... should be combined, a logical OR has to be used.

Exapmle:

```
    dwFalgs =  USBCAN_RESET_NO_COUNTER_ALL  |
               USBCAN_RESET_NO_BUFFER_ALL;
```

If the constants USBCAN_RESER_ONLY_... has to be combined, a logical AND has to be used.

Example:

```
    dwFalgs =  USBCAN_RESET_ONLY_RX_BUFF  &
               USBCAN_RESET_ONLY_STATUS;
```

For GW-002 the constant USBCAN_RESET_ONLY_RX_BUFF_GW02 has to be used instead of USBCAN_RESET_ONLY_RX_BUFF. But in this case the transmit buffer in module firmware will be reset too.

Return value:

Error code of the function.

USBCAN_SUCCESSFUL
USBCAN_ERR_MAXINSTANCES
USBCAN_ERR_ILLHANDLE
USBCAN_ERR_CANNOTINIT
USBCAN_ERR_BUSY
USBCAN_ERR_IOFAILED
USBCAN_ERRCMD_…
USBCAN_ERR_ILLHW
USBCAN_ERR_ILLCHANNEL
USBCAN_ERR_DATA
USBCAN_ERR_ABORT
USBCAN_ERR_DISCONNECT
USBCAN_ERR_TIMEOUT

UcanDeinitCan

Syntax:

```
UCANRET PUBLIC UcanDeinitCan (tUcanHandle UcanHandle_p);
```

Usability:

CAN_INIT

Description:

Shuts down the CAN interface of a USB-CANmodul. This function sets the operating voltage of the CAN controller to 0 V. After calling this function, all CAN messages received from CAN bus are ignored and not transferred to the PC.

Parameter:

UcanHandle_p:	USB-CAN handle received with the function *UcanInitHardware()* or *UcanInitHardwareEx()*.

Return value:

Error code of the function.

USBCAN_SUCCESSFUL
USBCAN_ERR_MAXINSTANCES
USBCAN_ERR_ILLHANDLE
USBCAN_ERR_CANNOTINIT
USBCAN_ERR_BUSY
USBCAN_ERR_IOFAILED
USBCAN_ERR_ILLHW
USBCAN_ERR_ILLCHANNEL
USBCAN_ERR_DATA
USBCAN_ERR_ABORT
USBCAN_ERR_DISCONNECT
USBCAN_ERR_TIMEOUT
USBCAN_ERRCMD_…

UcanDeinitCanEx

Syntax:

```
UCANRET PUBLIC UcanDeinitCanEx (
     tUcanHandle             UcanHandle_p ,
     BYTE                    bChannel_p);
```

Usability:

CAN_INIT

Description:

Shuts down a selective CAN interface of a USB-CANmodul. This function sets the operating voltage of the CAN controller to 0 V. After calling this function, all CAN messages received from CAN bus are ignored and not transferred to the PC.

Parameter:

UcanHandle_p:	USB-CAN handle received with the function *UcanInitHardware()* or *UcanInitHardwareEx()*.
bChannel_p:	CAN-channel that is to be shut down. USBCAN_CHANNEL_CH0 for CAN-channel 0 USBCAN_CHANNEL_CH1 for CAN-channel 1

Return value:

error code of the function.

USBCAN_SUCCESSFUL
USBCAN_ERR_MAXINSTANCES
USBCAN_ERR_ILLHANDLE
USBCAN_ERR_CANNOTINIT
USBCAN_ERR_BUSY
USBCAN_ERR_IOFAILED
USBCAN_ERR_ILLHW
USBCAN_ERR_ILLCHANNEL
USBCAN_ERR_DATA
USBCAN_ERR_ABORT
USBCAN_ERR_DISCONNECT
USBCAN_ERR_TIMEOUT
USBCAN_ERRCMD_…

UcanGetHardwareInfo

Syntax:

```
UCANRET PUBLIC UcanGetHardwareInfo (
     tUcanHandle              UcanHandle_p,
     tUcanHardwareInfo*       pHwInfo_p);
```

Usability:

HW_INIT, CAN_INIT

Description:

This function returns the hardware information of a USB-CANmodul. This function is especially useful if a USB-CANmodul has been initialized with the device number *USBCAN_ANY_MODULE*. Afterwards, the hardware information contains the device number of the initialized USB-CANmodul.

Parameter:

UcanHandle_p:	USB-CAN handle received with the function *UcanInitHardware()* or *UcanInitHardwareEx()*.
pHwInfo_p:	Address to the hardware information structure (*see description below*).

Return value:

Error code of the function.

USBCAN_SUCCESSFUL
USBCAN_ERR_MAXINSTANCES
USBCAN_ERR_ILLHANDLE
USBCAN_ERR_ILLPARAM
USBCAN_ERR_ILLHW

```
typedef struct
{
  BYTE        m_bDeviceNr;  // Device number
  tUcanHandle m_UcanHandle; // USB-CAN handle
  DWORD       m_dwReserved; // reserved
  BYTE        m_bBTR0;      // Baud rate register 0
  BYTE        m_bBTR1;      // Baud rate register 1
  BYTE        m_bOCR;       // Output control register
  DWORD       m_dwAMR;      // Acceptance mask register
  DWORD       m_dwACR;      // Acceptance code register
  BYTE        m_bMode;      // CAN controller mode
                            // (see tUcanMode)
  DWORD       m_dwSerialNr; // Serial number
                            // of the USB-CANmoduls
} tUcanHardwareInfo;
```

Note:

The parameters *m_bMode* and *m_dwSerialNr* are only available with the software version 2.16 and higher.

Example:

```
UCANRET bRet;
tUcanHandle UcanHandle;
tUcanHardwareInfo HwInfo;
_TCHAR szDeviceNr[24];

...
// initialize USB-CANmodul
bRet = UcanInitHardware (&UcanHandle, USBCAN_ANY_MODULE, NULL);

// no error?
if (bRet == USBCAN_SUCCESSFUL)
{
  // get hardware information
  UcanGetHardwareInfo (UcanHandle, &HwInfo);

  // change the device number into a string
  _stprintf (szDeviceNr, _T(„device number = %d“),
      HwInfo.m_bDeviceNr);
  ...
}
...
```

UcanGetHardwareInfoEx2

Syntax:

```
UCANRET PUBLIC UcanGetHardwareInfoEx2 (
     tUcanHandle             UcanHandle_p,
     tUcanHardwareInfoEx*    pHwInfoEx_p,
     tUcanChannelInfo*       pCanInfoCh0_p,
     tUcanChannelInfo*       pCanInfoCh1_p);
```

Usability:

HW_INIT, CAN_INIT ,version 3.00 and higher

Description:

This function returns the extended hardware information of a USB-CANmodul. For the Multiport CAN-to-USB 3004006, USB-CANmodul1 and USB-CANmodul2, the hardware information of each CAN-channel is returned separately.

Parameter:

UcanHandle_p:	USB-CAN handle received with the function *UcanInitHardware()* or *UcanInitHardwareEx()*.
pHwInfoEx_p:	Pointer to extended hardware information structure (see description below).
pCanInfoCh0_p:	Pointer to information structure used for CAN-channel 0. This parameter may be set to NULL.
pCanInfoCh1_p:	Pointer to information structure used for CAN-channel 1. This parameter may be set to NULL.

Return value:

Error code of the function.

USBCAN_SUCCESSFUL
USBCAN_ERR_MAXINSTANCES
USBCAN_ERR_ILLHANDLE
USBCAN_ERR_ILLPARAM
USBCAN_ERR_ILLHW

```
typedef struct
{
    DWORD       m_dwSize;           // number of Bytes of
                                    // this structure
    tUcanHandle m_UcanHandle;       // USB-CAN-Handle
    BYTE        m_bDeviceNr;        // device number
    DWORD       m_dwSerialNr;       // serial number
    DWORD       m_dwFwVersionEx;    // Firmware Version
    DWORD       m_dwReserved;       // reserved
    DWORD       m_dwProductCode;    // Hardware Type

} tUcanHardwareInfoEx;


typedef struct
{
  DWORD    m_dwSize;        // size of this structure in bytes
  BYTE     m_bMode;         // CAN-mode (see tUcanMode)
  BYTE     m_bBTR0;         // Bus Timing Register 0
  BYTE     m_bBTR1;         // Bus Timing Register 1
  BYTE     m_bOCR;          // Output Control Register
  DWORD    m_dwAMR;         // Acceptance Mask Register
  DWORD    m_dwACR;         // Acceptance Code Register
  DWORD    m_dwBaudrate;    // Baudrate Register for Multport,
                            // USB-CANmodul1 und USB-CANmodul2
  BOOL     m_fCanIsInit;    // is TRUE when CAN-channel was
                            // initialised
  WORD     m_wCanStatus;    // last CAN state
                            // (see UcanGetStatus())

} tUcanChannelInfo;
```

Use the following macros for getting information about the support of several new features:

USBCAN_CHECK_SUPPORT_CYCLIC_MSG(pHwIndoEx)
: This Macro checks whether the logical USB-CANmodul supports the automatic transmission of cyclic CAN messages.

USBCAN_CHECK_SUPPORT_TWO_CHANNEL(pHwIndoEx)
: This Macro checks whether the logical USB-CANmodul supports two CAN-channels.

USBCAN_CHECK_SUPPORT_TERM_RESISTOR(pHwIndoEx)
: This Macro checks whether the logical USB-CANmodul supports to read back the state of the termination resistor.

USBCAN_CHECK_SUPPORT_USER_PORT(pHwIndoEx)
: This Macro checks whether the logical USB-CANmodul supports a programmable Expansion Port (*refer to section 1.5*).

USBCAN_CHECK_SUPPORT_RBUSER_PORT(pHwIndoEx)
: This Macro checks whether the logical USB-CANmodul supports a programmable Expansion Port including the storing of the last output configuration to a non-volatile memory. After next power-on this configuration will be automatically set to the Expansion Port.

USBCAN_CHECK_SUPPORT_RBCAN_PORT(pHwIndoEx)

This Macro checks whether the logical USB-CANmodul supports a programmable CAN Port (for low-speed CAN transceivers) including the storing of the last output configuration to a non-volatile memory. After next power-on this configuration will be automatically set to the CAN Port.

Example:

```
UCANRET             bRet;
tUcanHandle         UcanHandle;
tUcanHardwareInfoEx HwInfoEx;

...
// init USB-CANmodul
bRet = UcanInitHardware (&UcanHandle, USBCAN_ANY_MODULE, NULL);
if (bRet == USBCAN_SUCCESSFUL)
{
    memset (&HwInfoEx, 0, sizeof (HwInfoEx));
    HwInfoEx.m_dwSize = sizeof (HwInfoEx);

    // get the extended hardware information
    bRet = UcanGetHardwareInfoEx2 (UcanHandle, &HwInfoEx,
        NULL, NULL);
    if (bRet == USBCAN_SUCCESSFUL)
    {
        TRACE1 ("product code = 0x%04X\n",
            HwInfoEx->m_dwProductCode & USBCAN_PRODCODE_MASK_PID);

        // check whether two CAN-channels are supported
        if (USBCAN_CHECK_SUPPORT_TWO_CHANNEL (&HwInfoEx))
        {
            ...
        }
        ...
    }
    ...
}
```

UcanGetMsgCountInfo

Syntax:

```
UCANRET PUBLIC UcanGetMsgCountInfo (
     tUcanHandle             UcanHandle_p,
     tUcanMsgCountInfo*      pMsgCountInfo_p);
```

Usability:

CAN_INIT , version 3.00 and higher

Description:

Reads the counters for transmitted and received CAN messages from the device.

Parameter:

UcanHandle_p:	USB-CAN-handle, that was returned by *UcanInitHardware()* or *UcanInitHardwareEx()*.
pMsgCountInfo_p:	Pointer to a structure of type *tUcanMsgCountInfo* where the counters are to be stored to

Return value:

Error code of the function.

USBCAN_SUCCESSFUL
USBCAN_ERR_MAXINSTANCES
USBCAN_ERR_ILLHANDLE
USBCAN_ERR_BUSY
USBCAN_ERR_IOFAILED
USBCAN_ERRCMD…
USBCAN_ERR_ILLHW
USBCAN_ERR_ILLPARAM
USBCAN_ERR_DATA
USBCAN_ERR_ABORT
USBCAN_ERR_DISCONNECT
USBCAN_ERR_TIMEOUT

```
typedef struct
{
  WORD m_wSentMsgCount; // Counter for transmitted CAN-messages
  WORD m_wRecvdMsgCount;// Counter for received CAN-messages

} tUcanMsgCountInfo;
```

UcanGetMsgCountInfoEx

Syntax:

```
UCANRET PUBLIC UcanGetMsgCountInfoEx (
     tUcanHandle              UcanHandle_p,
     BYTE                     bChannel_p,
     tUcanMsgCountInfo*       pMsgCountInfo_p);
```

Usability:

CAN_INIT, version 2.16 and higher

Description:

Reads the counters for transmitted and received CAN messages of a specific CAN-channel from the device.

Parameter:

UcanHandle_p:	USB-CAN-handle, that was returned by *UcanInitHardware()* or *UcanInitHardwareEx()*.
bChannel_p:	CAN-channel to read the counters from. USBCAN_CHANNEL_CH0 for CAN-channel 0 USBCAN_CHANNEL_CH1 for CAN-channel 1
pMsgCountInfo_p:	Pointer to a structure of type *tUcanMsgCountInfo* where the counters are to be stored to

Return value:

Error code of the function.

USBCAN_SUCCESSFUL
USBCAN_ERR_MAXINSTANCES
USBCAN_ERR_ILLHANDLE
USBCAN_ERR_BUSY
USBCAN_ERR_IOFAILED
USBCAN_ERR_ILLHW
USBCAN_ERR_ILLPARAM
USBCAN_ERR_DATA
USBCAN_ERR_ABORT
USBCAN_ERR_DISCONNECT
USBCAN_ERR_TIMEOUT
USBCAN_ERRCMD…

Structure *tUcanMsgCountInfo* is described with function *UcanGetMsgCountInfo()*.

UcanGetStatus

Syntax:

```
UCANRET PUBLIC UcanGetStatus (
     tUcanHandle                 UcanHandle_p,
     tStatusStruct*              pStatus_p);
```

Usability:

HW_INIT, CAN_INIT

Description:

This function returns the error status from the USB-CANmodul. If an error occurred on the USB-CANmodul, the red status LED starts blinking and a status message is sent to the PC. If a callback function has been handed over to the function *UcanInitHardware()* or *UcanInitHardwareEx()*, this callback function is called, as well as the event *USBCAN_EVENT_STATUS*. After calling the function *UcanGetStatus(),* the error state on the USB-CANmodul is erased and the red status LED stops blinking. Starting at version 2.17 a CAN status error must be cleared by calling the function UcanResetCan.

Parameter:

UcanHandle_p:	USB-CAN handle received with the function *UcanInitHardware()* or *UcanInitHardwareEx()*.
pStatus_p:	Error status of the USB-CANmodul.

```
typedef struct
{
  WORD    m_wCanStatus;   // present CAN status
  WORD    m_wUsbStatus;   // present USB status

} tStatusStruct;
```

Return value:

Error code of the function.

USBCAN_SUCCESSFUL
USBCAN_ERR_MAXINSTANCES
USBCAN_ERR_ILLHANDLE
USBCAN_ERR_ILLPARAM
USBCAN_ERR_ILLHW

The WORD *m_wCanStatus,* found in the *tStatusStruct* structure, returns the following values:

USBCAN_CANERR_OK	= 0x0000
No error	
USBCAN_CANERR_XMTFULL	= 0x0001
Transmit buffer in CAN controller is overrun	
USBCAN_CANERR_OVERRUN	= 0x0002
Receive buffer in CAN controller is overrun	
USBCAN_CANERR_BUSLIGHT	= 0x0004
Error limit 1 in CAN controller exceeded, CAN controller is in state “Warning limit” now	
USBCAN_CANERR_BUSHEAVY	= 0x0008
Error limit 2 in CAN controller exceeded, CAN controller is in state “Error Passive” now	
USBCAN_CANERR_BUSOFF	= 0x0010
CAN controller is in BUSOFF state	
USBCAN_CANERR_QOVERRUN	= 0x0040
Receive buffer in module is overrun	
USBCAN_CANERR_QXMTFULL	= 0x0080
Transmit buffer in module is overrun	
USBCAN_CANERR_REGTEST	= 0x0100
CAN controller not found (hardware error)	
USBCAN_CANERR_TXMSGLOST	= 0x0400
A transmit CAN message was deleted automatically by the firmware because transmission timeout run over (*refer to function UcanSetTxTimeout()*).	

This WORD is bit oriented; it can indicate multiple errors simultaneously.

WORD *m_wUsbStatus* is becoming obsolete and is retained only for compatibility purposes. It retains the value 0.

UcanGetStatusEx

Syntax:

```
UCANRET PUBLIC UcanGetStatusEx (
     tUcanHandle               UcanHandle_p,
     BYTE                      bChannel_p
     tStatusStruct*            pStatus_p);
```

Usability:

HW_INIT, CAN_INIT , version 3.00 and higher

Description:

This function returns the error status of a specific CAN-channel from the USB-CANmodul. This function may be used alternatively for function *UcanGetStatus()*.

Structure *tStatusStruct* is described in section *UcanGetStatus()*.

Parameter:

UcanHandle_p:	USB-CAN handle received with the function *UcanInitHardware()* or *UcanInitHardwareEx()*.
pStatus_p:	Error status of the USB-CANmodul.
bChannel_p:	Specifies the CAN-channel of which the status is to be returned. USBCAN_CHANNEL_CH0 for CAN-channel 0 USBCAN_CHANNEL_CH1 for CAN-channel 1

Return value:

error code of the function.

USBCAN_SUCCESSFUL
USBCAN_ERR_MAXINSTANCES
USBCAN_ERR_ILLHANDLE
USBCAN_ERR_ILLPARAM
USBCAN_ERR_ILLHW

UcanSetBaudrate

Syntax:

```
UCANRET PUBLIC UcanSetBaudrate (
    tUcanHandle              UcanHandle_p,
    BYTE                     bBTR0_p,
    BYTE                     bBTR1_p);
```

Usability:

CAN_INIT

Description:

Changes the baud rate configuration of the USB-CANmodul.

Parameter:

UcanHandle_p:	USB-CAN handle received with the function *UcanInitHardware()* or *UcanInitHardwareEx()*.
bBTR0_p:	Baud rate register 0 (*refer to section 2.3.4*)
bBTR1_p:	Baud rate register 1 (*refer to section 2.3.4*)

Return value:

Error code of the function.

USBCAN_SUCCESSFUL
USBCAN_ERR_MAXINSTANCES
USBCAN_ERR_ILLHANDLE
USBCAN_ERR_CANNOTINIT
USBCAN_ERR_BUSY
USBCAN_ERR_IOFAILED
USBCAN_ERR_ILLHW
USBCAN_ERR_ILLCHANNEL
USBCAN_ERR_DATA
USBCAN_ERR_ABORT
USBCAN_ERR_DISCONNECT
USBCAN_ERR_TIMEOUT
USBCAN_ERRCMD_…

UcanSetBaudrateEx

Syntax:

```
UCANRET PUBLIC UcanSetBaudrateEx (
     tUcanHandle            UcanHandle_p,
     BYTE                   bChannel_p
     BYTE                   bBTR0_p,
     BYTE                   bBTR1_p,
     DWORD                  dwBaudrate_p);
```

Usability:

CAN_INIT , version 3.00 and higher

Description:

Changes the baud rate configuration of a specific CAN-channel of the USB-CANmodul. This function may be used alternatively for function *UcanSetBaudrate()*.

Parameter:

UcanHandle_p:	USB-CAN handle received with the function *UcanInitHardware()* or *UcanInitHardwareEx()*.
bChannel_p:	CAN-channel that is to be changed USBCAN_CHANNEL_CH0 for CAN-channel 0 USBCAN_CHANNEL_CH1 for CAN-channel 1
bBTR0_p:	Baud rate register BTR0 (*refer to section 2.3.4*)
bBTR1_p:	Baud rate register BTR1 (*refer to section 2.3.4*)
dwBaudrate_p:	Baud rate register for all sysWORXX modules (*refer to section 2.3.4*)

Note:

The configuration of the baud rate differs significantly between the older USB-CANmodul versions (GW-001 and GW-002) and the new sysWORXX modules. For standardized baud rate values (*see section 2.3.4*), the baud rate registers BTR0 and BTR1 are as well applicable for the new sysWORXX modules. Therefore set *m_dwBaudrate* to USBCAN_BAUDEX_USE_BTR01.

Return value:

Error code of the function.

USBCAN_SUCCESSFUL
USBCAN_ERR_MAXINSTANCES
USBCAN_ERR_ILLHANDLE
USBCAN_ERR_CANNOTINIT
USBCAN_ERR_BUSY
USBCAN_ERR_IOFAILED
USBCAN_ERR_ILLHW
USBCAN_ERR_ILLCHANNEL
USBCAN_ERR_DATA
USBCAN_ERR_ABORT
USBCAN_ERR_DISCONNECT
USBCAN_ERR_TIMEOUT
USBCAN_ERRCMD_…

UcanSetAcceptance

Syntax:

```
UCANRET PUBLIC UcanSetAcceptance (
     tUcanHandle             UcanHandle_p,
     DWORD                   dwAMR_p,
     DWORD                   dwACR_p);
```

Usability:

CAN_INIT

Description:

Changes the acceptance Mask Register of the USB-CANmodul.

Parameter:

UcanHandle_p:	USB-CAN handle received with the function *UcanInitHardware()* or *UcanInitHardwareEx()*.
dwAMR_p:	Acceptance Mask Register (*see section 2.3.5*)
dwACR_p:	Acceptance Code Register (*see section 2.3.5*)

Return value:

Error code of the function.

USBCAN_SUCCESSFUL
USBCAN_ERR_MAXINSTANCES
USBCAN_ERR_ILLHANDLE
USBCAN_ERR_CANNOTINIT
USBCAN_ERR_BUSY
USBCAN_ERR_IOFAILED
USBCAN_ERR_ILLHW
USBCAN_ERR_ILLCHANNEL
USBCAN_ERR_DATA
USBCAN_ERR_ABORT
USBCAN_ERR_DISCONNECT
USBCAN_ERR_TIMEOUT
USBCAN_ERRCMD_…

UcanSetAcceptanceEx

Syntax:

```
UCANRET PUBLIC UcanSetAcceptanceEx (
    tUcanHandle             UcanHandle_p,
    BYTE                    bChannel_p
    DWORD                   dwAMR_p,
    DWORD                   dwACR_p);
```

Usability:

CAN_INIT version 3.00 and higher

Description:

Changes the acceptance Mask Register of a specific CAN-channel of the USB-CANmodul. This function may be used alternatively for function *UcanSetAcceptance()*.

Parameter:

UcanHandle_p:	USB-CAN handle received with the function *UcanInitHardware()* or *UcanInitHardwareEx()*.
bChannel_p:	CAN-channel that is to be changed USBCAN_CHANNEL_CH0 for CAN-channel 0 USBCAN_CHANNEL_CH1 for CAN-channel 1
dwAMR_p:	Acceptance Mask Register (*see section 2.3.5*)
dwACR_p:	Acceptance Code Register (*see section 2.3.5*)

Return value:

Error code of the function.

USBCAN_SUCCESSFUL
USBCAN_ERR_MAXINSTANCES
USBCAN_ERR_ILLHANDLE
USBCAN_ERR_CANNOTINIT
USBCAN_ERR_BUSY
USBCAN_ERR_IOFAILED
USBCAN_ERR_ILLHW
USBCAN_ERR_ILLCHANNEL
USBCAN_ERR_DATA
USBCAN_ERR_ABORT
USBCAN_ERR_DISCONNECT
USBCAN_ERR_TIMEOUT
USBCAN_ERRCMD_…

UcanReadCanMsg

Syntax:

```
UCANRET PUBLIC UcanReadCanMsg (
     tUcanHandle            UcanHandle_p,
     tCanMsgStruct*         pCanMsg_p);
```

Usability:

CAN_INIT

Description:

Reads a CAN message from the buffer. If the buffer contains no CAN messages, this function returns a warning. If a buffer overrun occurred, this function returns a valid CAN message and a warning.

Parameter:

UcanHandle_p:	USB-CAN handle received with the function *UcanInitHardware()* or *UcanInitHardwareEx()*.
pCanMsg_p:	Address to a CAN message structure. This address must not be NULL.

```
typedef struct
{
  DWORD   m_dwID;      // CAN identifier
  BYTE    m_bFF;       // CAN frame format
  BYTE    m_bDLC;      // CAN data length code
  BYTE    m_bData[8];  // CAN data
  DWORD   m_dwTime;    // Receipt time in ms

} tCanMsgStruct;
```

The CAN-frame format is a bit mask that specifies the format of the CAN-message. The following table lists all valid values:

Constant	**Value**	**Description**
USBCAN_MSG_FF_STD	0x00	CAN2.0A message with 11-bit CAN-ID
USBCAN_MSG_FF_ECHO	0x20	transmit echo; Is only received if mode kUcanModeTxEcho was enabled at initialization time.
USBCAN_MSG_FF_RTR	0x40	CAN Remote Frame
USBCAN_MSG_FF_EXT	0x80	CAN2.0B message with 29-bit CAN-ID

Table 14: Constants for the CAN-frame format

Return value:

Error code of the function.

USBCAN_SUCCESSFUL
USBCAN_ERR_MAXINSTANCES
USBCAN_ERR_ILLHANDLE
USBCAN_ERR_CANNOTINIT
USBCAN_ERR_ILLPARAM
USBCAN_ERR_ILLHW
USBCAN_ERR_ILLCHANNEL
USBCAN_WARN_NODATA
USBCAN_WARN_SYS_RXOVERRUN
USBCAN_WARN_DLL_RXOVERRUN
USBCAN_WARN_FW_RXOVERRUN

Example:

```
tUcanHandle UcanHandle;
tCabMsgStruct CanMsg;
UCANRET bRet;

...
while (1)
{
    // read CAN-message
    bRet = UcanReadCanMsg (UcanHandle, &CanMsg);

    // No error? Print CAN-message
    if (USBCAN_CHECK_VALID_RXCANMSG (bRet))
    {
        PrintCanMsg (&CanMsg);
        if (USBCAN_CHECK_WARNING (bRet))
        {
            PrintWarning (bRet);
        }
    }
    // No warning? Print error
    else if (USBCAN_CHECK_ERROR (bRet))
    {
        PrintError (bRet);
        break;
    }
    else
    {
        break;
    }
}
...
```

Note:

In order to avoid receive buffer overflows it is recommended to call function *UcanReadCanMsg()* cyclically (e.g. in a loop) as long as a valid CAN-message was received.

A valid CAN-message was read, even if a warning was returned (except USBCAN_WARN_NODATA). You can use the macro *USBCAN_CHECK_VALID_RXCANMSG()* for checking whether a valid CAN message was stored to the CAN message structure (like shown in upper example).

The variable *m_dwTime* of structure *tCanMsgStruct* contains a time stamp of 32 bit but only 24 bits are valid. The reason for this is that this time stamp is generated on the hardware. For compatibility reasons, this is the same implementation for all sysWORXX USB-CANmoduls. You have to keep this in mind if time differences are calculated by using this time stamp. We recommend using the macro USBCAN_CALC_TIMEDIFF().

Example:

```
DWORD dwTimeDiff, dwOldTime;
...
    dwTimeDiff = USBCAN_CALC_TIMEDIFF (dwOldTime,
        CanMsg.m_dwTime);
    dwOldTime = CanMsg.m_dwTime;
...
```

UcanReadCanMsgEx

Syntax:

```
UCANRET PUBLIC UcanReadCanMsgEx (
     tUcanHandle              UcanHandle_p,
     BYTE*                    pbChannel_p
     tCanMsgStruct*           pCanMsg_p,
     DWORD*                   pdwCount_p);
```

Usability:

CAN_INIT ,version 3.00 and higher

Description:

Reads a CAN message from the buffer of a specific CAN-channel. If the buffer contains no CAN messages, this function returns a warning. If a buffer overrun occurred, this function returns a valid CAN message and a warning. This function may be used alternatively for function *UcanReadCanMsg()*.

Parameter:

UcanHandle_p:	USB CAN handle received with the function *UcanInitHardware()* or *UcanInitHardwareEx()*.
pCanMsg_p:	Address to a CAN message structure. This address must not be NULL.
bChannel_p:	CAN-channel to read data from USBCAN_CHANNEL_CH0 for CAN-channel 0 USBCAN_CHANNEL_CH1 for CAN-channel 1 If USBCAN_CHANNEL_ANY is given, the function will write the number of the CAN-channel that received CAN messages to this parameter
pdwCount_p:	Address to a variable that specifies the maximum number of CAN messages to be read. This function writes the actual number of CAN messages that were read from the device to this variable. If this parameter is set to NULL, only one CAN message is read from the device.

The structure *tCanMsgStruct* is described in function *UcanReadCanMsg()*.

Return value:

Error code of the function.

USBCAN_SUCCESSFUL
USBCAN_ERR_MAXINSTANCES
USBCAN_ERR_ILLHANDLE
USBCAN_ERR_CANNOTINIT
USBCAN_ERR_ILLPARAM
USBCAN_ERR_ILLHW
USBCAN_ERR_ILLCHANNEL
USBCAN_WARN_NODATA
USBCAN_WARN_SYS_RXOVERRUN
USBCAN_WARN_DLL_RXOVERRUN
USBCAN_WARN_FW_RXOVERRUN

Example:

```
tUcanHandle UcanHandle;
tCabMsgStruct RxCanMsg[16];
UCANRET bRet;
BYTE  bChannel;
DWORD dwCount;

while (1)
{
    // read up to 16 CAN messages
    bChannel = USBCAN_CHANNEL_ANY;
    dwCount  = sizeof (RxCanMsg) / sizeof (tCabMsgStruct);
    bRet = UcanReadCanMsgEx (UcanHandle, &bChannel,
        &RxCanMsg, &dwCount);

    // No error? print CAN-message
    if (USBCAN_CHECK_VALID_RXMSG (bRet))
    {
        PrintCanMessages (&RxCanMsg[0], dwCount);
        if (USBCAN_CHECK_WARNING (bRet))
        {
            PrintWarning (bRet);
        }
    }
    // No warning? Print error
    else if (USBCAN_CHECK_WARNING (bRet))
    {
        PrintError (bRet);
        break;
    }
    else
    {
        break;
    }
}
...
```

Note:

In order to avoid receive buffer overflows it is recommended to call function *UcanReadCanMsg()* cyclically (e.g. in a loop) as long as a valid CAN-message was received.

A valid CAN-message was read, even if a warning was returned (except USBCAN_WARN_NODATA). You can use the macro *USBCAN_CHECK_VALID_RXCANMSG()* for checking whether a valid CAN message was stored to the CAN message structure (like shown in upper example).

Since software version 3.05 the size of the receive buffer (maximum number of CAN messages) is configurable (*see function UcanInitCanEx() and structure tUcanInitCanParam)*

The variable *m_dwTime* of structure *tCanMsgStruct* contains a time stamp of 32 bit but only 24 bits are valid. The reason for this is that this time stamp is generated on the hardware. For compatibility reasons, this is the same implementation for all sysWORXX USB-CANmoduls. You have to keep this in mind if time differences are calculated by using this time stamp. We recommend using the macro USBCAN_CALC_TIMEDIFF().

Example:

```
DWORD dwTimeDiff, dwOldTime;
...
    dwTimeDiff = USBCAN_CALC_TIMEDIFF (dwOldTime,
        CanMsg.m_dwTime);
    dwOldTime = CanMsg.m_dwTime;
...
```

UcanWriteCanMsg

Syntax:

```
UCANRET PUBLIC UcanWriteCanMsg (
     tUcanHandle             UcanHandle_p,
     tCanMsgStruct*          pCanMsg_p);
```

Usability:

CAN_INIT

Description:

Transmits a CAN message through the USB-CANmodul.

Parameter:

UcanHandle_p:	USB-CAN handle received with the function *UcanInitHardware()* or *UcanInitHardwareEx()*.
pCanMsg_p:	Address to a CAN message structure. This address must not be NULL.

```
typedef struct
{
  DWORD   m_dwID;      // CAN identifier
  BYTE    m_bFF;       // CAN frame format *)
  BYTE    m_bDLC;      // CAN data length code
  BYTE    m_bData[8]; // CAN data
  DWORD   m_dwTime;    // has no meaning in this function

} tCanMsgStruct;
```

*) The meaning of CAN frame format is given with function *UcanReadCanMsg*(). For transmission of CAN messages, bit USBCAN_MSG_FF_ECHO has no meaning.

Return value:

Error code of the function.

USBCAN_SUCCESSFUL
USBCAN_ERR_MAXINSTANCES
USBCAN_ERR_ILLHANDLE
USBCAN_ERR_CANNOTINIT
USBCAN_ERR_ILLPARAM
USBCAN_ERR_DLL_TXFULL
USBCAN_ERR_ILLHW
USBCAN_ERR_ILLCHANNEL
USBCAN_WARN_FW_TXOVERRUN

UcanWriteCanMsgEx

Syntax:

```
UCANRET PUBLIC UcanWriteCanMsgEx (
    tUcanHandle            UcanHandle_p,
    BYTE                   bChannel_p
    tCanMsgStruct*         pCanMsg_p
    DWORD*                 pdwCount_p);
```

Usability:

CAN_INIT , version 3.00 and higher

Description:

Transmits one ore more CAN messages through the specified CAN-channel of the USB-CANmodul. This function may be used alternatively for function *UcanWriteCanMsg()*.

Parameter:

UcanHandle_p:	USB CAN handle received with the function *UcanInitHardware()* or *UcanInitHardwareEx()*.
pCanMsg_p:	Address to a CAN message structure. This address must not be NULL.
bChannel_p:	CAN-channel to read data from USBCAN_CHANNEL_CH0 for CAN-channel 0 USBCAN_CHANNEL_CH1 for CAN-channel 1
pdwCount_p:	Address to a variable that specifies the maximum number of CAN messages to be transmitted. After calling, this function writes the actual number of CAN messages that were sent to this variable. If this parameter is set to NULL, only one CAN message will be transmitted.

The structure *tCanMsgStruct* is described with function *UcanWriteCanMsg()*.

Return value:

Error code of the function.

USBCAN_SUCCESSFUL
USBCAN_ERR_MAXINSTANCES
USBCAN_ERR_ILLHANDLE
USBCAN_ERR_CANNOTINIT
USBCAN_ERR_ILLPARAM
USBCAN_ERR_DLL_TXFULL
USBCAN_ERR_ILLHW
USBCAN_ERR_ILLCHANNEL
USBCAN_WARN_FW_TXOVERRUN
USBCAN_WARN_TXLIMIT

Note:

If this function is called for transmitting more than one CAN messages, then the return code has also to be checked for the warning USBCAN_WARN_TXLIMIT. Receiving this return value only a part of the CAN messages was stored to the transmit buffer in USBCAN32.DLL. The variable which is referenced by the parameter *pdwCount_p* gets the number of successfully stored CAN messages. The part which was not stored to the transmit buffer has to be tried to be sent again by the application. Otherwise they will be lost.

You can use the macro *USBCAN_CHECK_TX_NOTALL()* for checking the return value whether some CAN messages could not be copied to the transmit buffer (see lower example). The macro *USBCAN_CHECK_TX_SUCCESS()* checks whether all CAN messages could be stored to the transmit buffer while the macro *USBCAN_CHECK_TX_OK()* checks whether one CAN message at least was stored to the transmit buffer.

Example:

```
tUcanHandle UcanHandle;
tCabMsgStruct TxCanMsg[10];
UCANRET bRet;
DWORD dwCount;

...
{
    // transmit up to 10 CAN messages
    dwCount  = sizeof (TxCanMsg) / sizeof (tCabMsgStruct);
    bRet = UcanWriteCanMsgEx (UcanHandle, USBCAN_CHANNEL_CH0,
        &TxCanMsg, &dwCount);

    // Check whether no error occurred
    if (USBCAN_CHECK_TX_OK (bRet))
    {
        // check whether a part of the array was not sent
        if (USBCAN_CHECK_TX_NOTALL (bRet))
        {
            ...
        }
        // check whether there was another warning
        else if (USBCAN_CHECK_WARNING (bRet))
        {
            PrintWarning (bRet);
        }
    }
    // check wheher an error occurred
    else if (USBCAN_CHECK_ERROR (bRet))
    {
        PrintError (bRet);
    }
}
...
```

UcanGetMsgPending

Syntax:

```
UCANRET PUBLIC UcanGetMsgPending (
    tUcanHandle            UcanHandle_p,
    BYTE                   bChannel_p,
    DWORD                  dwFlags_p,
    DWORD*                 pdwCount_p);
```

Usability:

CAN_INIT , version 3.06 and higher, only sysWORXX modules

Description:

This function returns the number of the CAN messages which are currently stored to the buffers within the several software parts. The parameter *dwFlags_p* specifies which buffers should be checked. Should the function check more than one buffer, then the number of CAN messages will be added before writing to the variable which is referenced by the parameter *pdwCount_p*.

Parameter:

UcanHandle_p:	USB CAN handle received with the function *UcanInitHardware()* or *UcanInitHardwareEx()*.
bChannel_p:	CAN-channel to read data from USBCAN_CHANNEL_CH0 for CAN-channel 0 USBCAN_CHANNEL_CH1 for CAN-channel 1
dwFlags_p:	Specifies which buffers should be checked (*refer to Table 15*). The several flags can be combined. In that case the number of CAN messages will be added.
pdwCount_p:	Address to a variable. After calling this function writes the number of CAN messages stored to the specified buffer(s) to this variable. This parameter must not be NULL.

Constant USBCAN_PENDING…	**Value**	**Meaning**
..._FLAG_RX_DLL	0x00000001	Checks the number of messages of receive buffer within USBCAN-libarary.
..._FLAG_RX_FW	0x00000004	Checks the number of messages of receive buffer within module formware.
..._FLAG_TX_DLL	0x00000010	Checks the number of messages of transmit buffer within USBCAN-library
..._FLAG_TX_FW	0x00000040	Checks the number of messages of transmit buffer within module firmware.

Table 15: Constants for the flags parameter in function UcanGetMsgPending()

Return value:

Error code of the function.

USBCAN_SUCCESSFUL
USBCAN_ERR_ILLPARAM
USBCAN_ERR_ILLHANDLE
USBCAN_ERR_MAXINSTANCES
USBCAN_ERR_ILLHW
USBCAN_ERR_ILLHWTYPE
USBCAN_ERR_ILLCHANNEL
USBCAN_ERR_CANNOTINIT
USBCAN_ERRCMD_…

Note:

After function *UcanGetMsgPending()* returned to the application, the number of the CAN messages can already be changed within the several software parts. When the application calls this function to often, the performance can spiral downward.

UcanGetCanErrorCounter

Syntax:

```
UCANRET PUBLIC UcanGetCanErrorCounter (
    tUcanHandle           UcanHandle_p,
    BYTE                  bChannel_p,
    DWORD *               pdwTxCount_p,
    DWORD*                pdwRxCount_p);
```

Usability:

CAN_INIT , version 3.06 and higher, only sysWORXX modules

Description:

Returns the current error counters from CAN controller. This values are directly read from the hardware.

Parameter:

UcanHandle_p:	USB-CAN handle received with the function *UcanInitHardware()* or *UcanInitHardwareEx()*.
bChannel_p:	CAN-channel to read data from USBCAN_CHANNEL_CH0 for CAN-channel 0 USBCAN_CHANNEL_CH1 for CAN-channel 1
pdwTxCount_p:	Address to a DWORD variable. After calling this function writes the current state of the transmit error counter to this variable. This parameter must not be NULL.
pdwRxCount_p:	Address to a DWORD variable. After calling this function writes the current state of the receive error counter to this variable. This parameter must not be NULL.

Return value:

Error code of the function.

USBCAN_SUCCESSFUL
USBCAN_ERR_ILLPARAM
USBCAN_ERR_ILLHANDLE
USBCAN_ERR_MAXINSTANCES
USBCAN_ERR_ILLHW
USBCAN_ERR_ILLHWTYPE
USBCAN_ERR_ILLCHANNEL
USBCAN_ERR_CANNOTINIT
USBCAN_ERRCMD_…

2.3.2.2 Functions for automatic transmission

The following functions can only be used for the new sysWORXX modules (not for GW-001 and GW-002). They are used to automatic transmission of cyclic CAN messages by the module firmware. This results a better cycle time as a Windows PC application could realize.

Note:

The accuracy of the cycle time also depends on the configured CAN baud rate. A jitter of 10 milliseconds is a result of using a CAN baud rate of 10 kbit per sec.

There is a maximum of 16 CAN messages which can be defined for the automatic transmission of cyclic CAN messages. Two modes are available for the automatic transmission. The first mode is called “parallel mode” the second one is called “sequential mode”.

In parallel mode the cycle times of all defined CAN messages are checked within a process cycle. When a cycle time of a defined CAN message is over it will be sent to the CAN bus. The cycle time of a defined CAN message relates to the previous transmission of the same CAN message (*refer to Figure 18*).

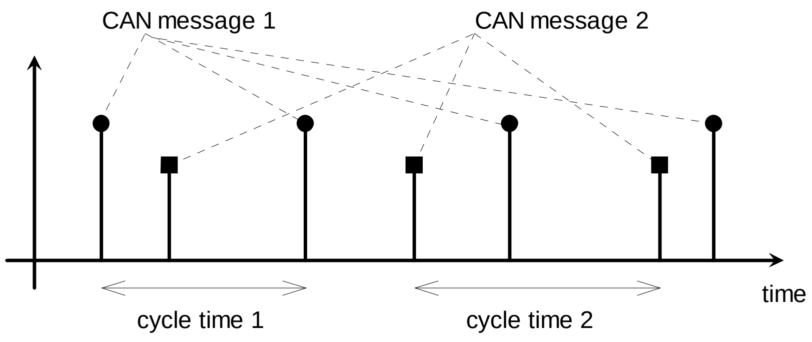


Figure 18: Examle for parallel mode with two defined CAN messages

In sequential mode the defined CAN messages are considered as a list of CAN messages which should be sent sequentially to the CAN bus. The cycle time of a defined CAN message relates to the transmission of the previously defined CAN message (refer to XXX). You can define a CAN message including the same CAN identifier but different data bytes more than once in sequential mode.

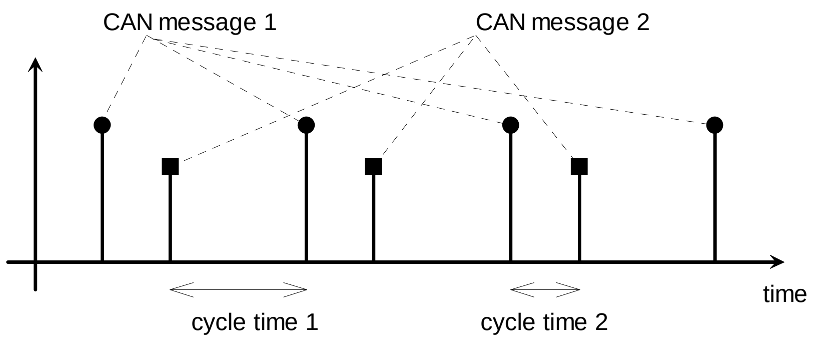


Figure 19: Example of sequential mode with two defined CAN messages

Important:

The transmission of CAN messages by calling the function *UcanWriteCanMsg()* or *UcanWriteCanMsgEx()* can be influenced by the automatic transmission of cyclic CAN messages. When the CAN bus load is much increased (50% and more) the CAN messages from application are processed more rarely. The result can be that the function *UcanWriteCanMsg()* or *UcanWriteCanMsgEx()* returns the warning indicating a receive overrun.

UcanDefineCyclicCanMsg

Syntax:

```
UCANRET PUBLIC UcanDefineCyclicCanMsg (
     tUcanHandle              UcanHandle_p,
     BYTE                     bChannel_p,
     tCanMsgStruct*           pCanMsgList_p,
     DWORD                    dwCount_p);
```

Usability:

HW_INIT, CAN_INIT, version 3.06 and higher, only sysWORXX modules

Description:

The function defines a set of up to 16 CAN messages within firmware of a USB-CANmodul for the automatic transmission of cyclic CAN messages. Call function *UcanEnableCyclicCanMsg()* for enabling the automatic transmission. Please note that *UcanDefineCyclicCanMsg()* completely exchanges a previously defined set of CAN messages.

Parameter:

UcanHandle_p:	USB-CAN handle received with the function *UcanInitHardware()* or *UcanInitHardwareEx()*.
bChannel_p:	CAN-channel to transmit to USBCAN_CHANNEL_CH0 for CAN-channel 0 USBCAN_CHANNEL_CH1 for CAN-channel 1
pCanMsgList_p:	Address to an array of type *tCanMsgStruct* containing a set of CAN messages for automatic transmission. The member *m_dwTime* of the structure *tCanMsgStruct* specifies the cycle time. This parameter may only be NULL when *dwCount_p* is zero too.
dwCount_p:	Specifies the number of CAN messages included within the array. The value range is 0 to 16. A previously defined set of CAN messages will only be deleted by specifying the number of 0 CAN messages.

Refer to the function *UcanWriteCanMsg()* for the definition of the structure *tCanMsgStruct*.

Return value:

Error code of the function.

USBCAN_SUCCESSFUL
USBCAN_ERR_ILLPARAM
USBCAN_ERR_ILLHANDLE
USBCAN_ERR_MAXINSTANCES
USBCAN_ERR_ILLHW
USBCAN_ERR_ILLHWTYPE
USBCAN_ERR_ILLCHANNEL
USBCAN_ERRCMD_…

UcanReadCyclicCanMsg

Syntax:

```
UCANRET PUBLIC UcanReadCyclicCanMsg (
    tUcanHandle            UcanHandle_p,
    BYTE                   bChannel_p,
    tCanMsgStruct*         pCanMsgList_p,
    DWORD*                 pdwCount_p);
```

Usability:

HW_INIT, CAN_INIT, version 3.06 and higher, only sysWORXX modules

Description:

The function reads back the set of CAN messages which was previously defined for automatic transmission of cyclic CAN messages.

Parameter:

UcanHandle_p:	USB-CAN handle received with the function *UcanInitHardware()* or *UcanInitHardwareEx()*.
bChannel_p:	CAN-channel to transmit to USBCAN_CHANNEL_CH0 for CAN-channel 0 USBCAN_CHANNEL_CH1 for CAN-channel 1
pCanMsgList_p:	Address to an array of type *tCanMsgStruct* receiving the set of CAN messages for automatic transmission. This parameter must not be NULL.
pdwCount_p:	Address to a variable of type DWORD where the function should copy to the number of defined CAN messages within the set.

Refer to the function *UcanWriteCanMsg()* for the definition of the structure *tCanMsgStruct*.

Return value:

Error code of the function.

USBCAN_SUCCESSFUL
USBCAN_ERR_ILLPARAM
USBCAN_ERR_ILLHANDLE
USBCAN_ERR_MAXINSTANCES
USBCAN_ERR_ILLHW
USBCAN_ERR_ILLHWTYPE
USBCAN_ERR_ILLCHANNEL
USBCAN_ERRCMD_…

UcanEnableCyclicCanMsg

Syntax:

```
UCANRET PUBLIC UcanEnableCyclicCanMsg (
    tUcanHandle             UcanHandle_p,
    BYTE                    bChannel_p,
    DWORD                   dwFlags_p);
```

Usability:

HW_INIT, CAN_INIT, version 3.06 and higher, only sysWORXX modules

Description:

This function specifies the mode of the automatic transmission and specifies whether the automatic transmission of a previous defined set of defined CAN messages should be enabled or disabled. Additionally separate CAN messages of the set can be locked or unlocked.

Parameter:

UcanHandle_p:	USB-CAN handle received with the function *UcanInitHardware()* or *UcanInitHardwareEx()*.
bChannel_p:	CAN-channel USBCAN_CHANNEL_CH0 for CAN-channel 0 USBCAN_CHANNEL_CH1 for CAN-channel 1
dwFlags_p:	Specifies flags containing the mode, the enable state and the locking state (*refer to Table 16*). These flags can be combined.

Return value:

Error code of the function.

USBCAN_SUCCESSFUL
USBCAN_ERR_ILLPARAM
USBCAN_ERR_ILLHANDLE
USBCAN_ERR_MAXINSTANCES
USBCAN_ERR_ILLHW
USBCAN_ERR_ILLHWTYPE
USBCAN_ERR_ILLCHANNEL
USBCAN_ERR_CANNOTINIT
USBCAN_ERRCMD_…

Constant USBCAN_CYCLE…	**Value**	**Meaning**
..._FLAG_START	0x80000000	When this flag is set, the automatic transmission will be started, otherwise it will be stopped.
..._FLAG_SEQUMODE	0x40000000	When this flag is set, the “sequential mode” is processed, otherwise the “parallel mode” is processed.
..._FLAG_NOECHO	0x00010000	When this flag is set, the sent cyclic CAN messages are not received back using transmit echo.
..._FLAG_LOCK_0 until ..._FLAG_LOCK_15	0x00000001 - 0x00008000	When same of these flags are set, the appropriate CAN message from the set is not sent to the CAN bus (locked state).

Table 16: Constants for the flags parameter in function UcanEnableCyclicCanMsg()

2.3.2.3 Functions for the CAN Port

The following functions can only be used with the GW-002-XXX, Multiport CAN-to-USB and USB-CANmodul2 (not applicable for GW-001). They are an expansion for using the USB-CANmodul with a low-speed CAN transceiver (e.g.: GW-002-010, GW-002-020, GW-002-030). If these functions are used with the GW-001 variant, then the error code **USBCAN_ERRCMD_ILLCMD** will be returned. Use of these functions with the GW-002 (82C251 high-speed CAN transceiver) or USB-CANmodul1 has no effect. However no error message will be returned either. In order to be able to use these functions, the header file USBCANLS.H must be included in addition to the USBCAN32.H header file.

UcanWriteCanPort

Syntax:

```
UCANRET PUBLIC UcanWriteCanPort (
     tUcanHandle             UcanHandle_p,
     BYTE                    bValue_p);
```

Usability:

HW_INIT, CAN_INIT, version 2.15 or higher

Description:

Writes a value to the CAN port interface. Thus additional signals such as Standby (STB) and Enable (EN) on a low-speed CAN transceiver can be controlled.

Parameter:

UcanHandle_p:	USB-CAN handle received with the function *UcanInitHardware()* or *UcanInitHardwareEx()*.
bOutValue_p:	New output value for the CAN port interface (*see Table 17*).

Return value:

Error code of the function.

USBCAN_SUCCESSFUL
USBCAN_ERR_MAXINSTANCES
USBCAN_ERR_ILLHANDLE
USBCAN_ERR_BUSY
USBCAN_ERR_IOFAILED
USBCAN_ERR_ILLHW
USBCAN_ERR_DATA
USBCAN_ERR_ABORT
USBCAN_ERR_DISCONNECT
USBCAN_ERR_TIMEOUT
USBCAN_ERRCMD_…

Note:

Following initialization of the USB-CANmodul with the function *UcanInitCan()*, these signals are already set for immediate operation of the USB-CANmodul.

Since software version 3.00, the last saved output values are restored after power-on on the sysWORXX modules.

UcanWriteCanPortEx

Syntax:

```
UCANRET PUBLIC UcanWriteCanPortEx (
    tUcanHandle            UcanHandle_p,
    BYTE                   bChannel_p,
    BYTE                   bOutValue_p);
```

Usability:

HW_INIT, CAN_INIT , version 3.00 and higher

Description:

Writes a value to the CAN port interface. Thus additional signals such as Standby (STB) and Enable (EN) on a low-speed CAN transceiver can be controlled. This function may be used alternatively for function *UcanWriteCanPort()*.

Parameter:

UcanHandle_p:	USB-CAN handle received with the function *UcanInitHardware()* or *UcanInitHardwareEx()*.
bChannel_p:	CAN-channel to read data from USBCAN_CHANNEL_CH0 for CAN-channel 0 USBCAN_CHANNEL_CH1 for CAN-channel 1
bOutValue_p:	New output value for the CAN port interface (*see Table 17*).

Return value:

Error code of the function.

USBCAN_SUCCESSFUL
USBCAN_ERR_MAXINSTANCES
USBCAN_ERR_ILLHANDLE
USBCAN_ERR_BUSY
USBCAN_ERR_IOFAILED
USBCAN_ERR_ILLHW
USBCAN_ERR_DATA
USBCAN_ERR_ABORT
USBCAN_ERR_DISCONNECT
USBCAN_ERR_TIMEOUT
USBCAN_ERRCMD_…

UcanReadCanPort

Syntax:

```
UCANRET PUBLIC UcanReadCanPort (
    tUcanHandle             UcanHandle_p,
    BYTE*                   pbValue_p);
```

Usability:

HW_INIT, CAN_INIT, version 2.15 or higher

Description:

Reads the current input value from the CAN port interface. Thus the additional signal (ERR for error) can be read on a low-speed CAN transceiver. It is also possible to read the state/constant for the terminating resistor on devices with high-speed transceivers (currently only supported for USB-CANmodul2).

Parameter:

UcanHandle_p:	USB-CAN handle received with the function *UcanInitHardware()* or *UcanInitHardwareEx()*.
pbInValue_p:	Address pointing to a variable that contains the read input value following the successful return of this function. This variable then has the following meanings (*see also section 1.4*):

Constant	**Bit value**	**Description**
UCAN_CANPORT_TRM	0x10	[IN] termination resistor
UCAN_CANPORT_ERR	0x20	[IN] error signal of low speed CAN transceiver
UCAN_CANPORT_STB	0x40	[OUT] stand-by signal of low speed CAN transceiver
UCAN_CANPORT_EN	0x80	[OUT] enable signal of low speed CAN transceiver

Table 17: Constants for low speed CAN port

Return value:

Error code of the function.

USBCAN_SUCCESSFUL
USBCAN_ERR_MAXINSTANCES
USBCAN_ERR_ILLHANDLE
USBCAN_ERR_BUSY
USBCAN_ERR_IOFAILED
USBCAN_ERR_ILLHW
USBCAN_ERR_ILLPARAM
USBCAN_ERR_DATA
USBCAN_ERR_ABORT
USBCAN_ERR_DISCONNECT
USBCAN_ERR_TIMEOUT
USBCAN_ERRCMD_…

UcanReadCanPortEx

Syntax:

```
UCANRET PUBLIC UcanReadCanPortEx (
    tUcanHandle             UcanHandle_p,
    BYTE                    bChannel_p
    BYTE*                   pbInValue_p,
    BYTE*                   pbLastOut_p);
```

Usability:

HW_INIT, CAN_INIT , version 3.00 or higher

Description:

Reads the current input value from the specified CAN-channel. This function may be used alternatively for function *UcanReadCanPort()*.

Parameter:

UcanHandle_p:	USB CAN handle received with the function *UcanInitHardware()* or *UcanInitHardwareEx()*.
bChannel_p:	CAN-channel to read data from USBCAN_CHANNEL_CH0 for CAN-channel 0 USBCAN_CHANNEL_CH1 for CAN-channel 1
pbInValue_p:	Address pointing to a variable that contains the read input value following the successful return of this function (*see Table 17*).
pbLastOut_p:	Address pointing to a variable that contains the last written output value (using *UcanWriteCanPort()* or *UcanWRiteCanPortEx()*) following the successful return of this function. This parameter may be NULL.

Return value:

Error code of the function.

USBCAN_SUCCESSFUL
USBCAN_ERR_MAXINSTANCES
USBCAN_ERR_ILLHANDLE
USBCAN_ERR_BUSY
USBCAN_ERR_IOFAILED
USBCAN_ERR_ILLHW
USBCAN_ERR_ILLPARAM
USBCAN_ERR_DATA
USBCAN_ERR_ABORT
USBCAN_ERR_DISCONNECT
USBCAN_ERR_TIMEOUT
USBCAN_ERRCMD_…

2.3.2.4 Functions for the Expansion Port

The following functions can only be used with the GW-002-XXX, Multiport CAN-to-USB 3004006 and USB-CANmodul2 3204002/3204003. They are an expansion for the use of the USB-CANmodul with the expansion port. If these functions are used with the GW-001, then the error code **USBCAN_ERRCMD_ILLCMD** will be returned. In order to use these functions, the file USBCANUP.H must be included in addition to the USBCAN32.H header file.

Note:

These functions are also applicable to for USB-CANmodul1. But as the USB-CANmodul1 does not feature an Expansion Port, these functions are ignored and the functions return with error code USBCAN_SUCCESSFUL.

UcanConfigUserPort

Syntax:

```
UCANRET PUBLIC UcanConfigUserPort (
    tUcanHandle             UcanHandle_p,
    BYTE                    bOutEn_p);
```

Usability:

HW_INIT, CAN_INIT, version 2.16 or higher

Description:

Configures the expansion port (*refer to section 1.5*). Each individual pin of the 8-bit port can be used as an input or an output. The logical value 0 of a bit in the parameter *bOutputEnable_p* defines the corresponding pin on the expansion port to function as an input and a logical 1 defines it as an output.

Parameter:

UcanHandle_p:	USB-CAN handle received with the function *UcanInitHardware()* or *UcanInitHardwareEx()*.
bOutEn_p:	Configuring the 8-bit port as input or output. Bit X = 0: Pin X = input Bit Y = 1: PinY = output

Return value:

Error code of the function.

USBCAN_SUCCESSFUL
USBCAN_ERR_MAXINSTANCES
USBCAN_ERR_ILLHANDLE
USBCAN_ERR_BUSY
USBCAN_ERR_IOFAILED
USBCAN_ERR_ILLHW
USBCAN_ERR_DATA
USBCAN_ERR_ABORT
USBCAN_ERR_DISCONNECT
USBCAN_ERR_TIMEOUT
USBCAN_ERRCMD_…

Note:

After connecting the USB-CANmodul to the PC all expansion port pins are configured as inputs.

Since software version 3.00, the last saved configuration is restored after power-on on sys WORXX modules.

UcanWriteUserPort

Syntax:

```
UCANRET PUBLIC UcanWriteUserPort (
    tUcanHandle           UcanHandle_p,
    BYTE                  bOutValue_p);
```

Usability:

HW_INIT, CAN_INIT, version 2.16 or higher

Description:

Writes a value to the expansion port. In order to write to output lines, the corresponding bits resp. port pins must be configured as outputs using the *UcanConfigUserPort()* function.

Parameter:

UcanHandle_p:	USB-CAN handle received with the function *UcanInitHardware()* or *UcanInitHardwareEx()*.
bOutValue_p:	New output value for the expansion port outputs. Each bit in this parameter corresponds to matching pin on the expansion port.

Return value:

Error code of the function.

USBCAN_SUCCESSFUL
USBCAN_ERR_MAXINSTANCES
USBCAN_ERR_ILLHANDLE
USBCAN_ERR_BUSY
USBCAN_ERR_IOFAILED
USBCAN_ERR_ILLHW
USBCAN_ERR_DATA
USBCAN_ERR_ABORT
USBCAN_ERR_DISCONNECT
USBCAN_ERR_TIMEOUT
USBCAN_ERRCMD_…

Note:

For GW-002 the supply voltage for the expansion port (pin 10, *refer to Table 6*) is connected only after the function *UcanInitCan()* is called. After the USB-CANmodul has been connected to the PC, all expansion port pins are configured as inputs. No time critical switching procedures can be performed with this function using the expansion port, since the reaction time is influenced by multiple factors.

Since software version 3.00, the last saved configuration of the outputs is restored after power-on on sysWORXX modules.

UcanReadUserPort

Syntax:

```
UCANRET PUBLIC UcanReadUserPort (
    tUcanHandle              UcanHandle_p,
    BYTE*                    pbInValue_p);
```

Usability:

HW_INIT, CAN_INIT, version 2.16 or higher

Description:

Reads the current input value from the expansion port.

Parameter:

UcanHandle_p:	USB-CAN handle received with the function *UcanInitHardware()* or *UcanInitHardwareEx()*.
pbInValue_p:	Address pointing to a variable that contains the read input value following the successful return of this function. This variable then contains the state of the 8-bit expansion port. Each bit in this parameter corresponds to matching pin on the expansion port.

Return value:

Error code of the function.

USBCAN_SUCCESSFUL
USBCAN_ERR_MAXINSTANCES
USBCAN_ERR_ILLHANDLE
USBCAN_ERR_BUSY
USBCAN_ERR_IOFAILED
USBCAN_ERR_ILLHW
USBCAN_ERR_ILLPARAM
USBCAN_ERR_DATA
USBCAN_ERR_ABORT
USBCAN_ERR_DISCONNECT
USBCAN_ERR_TIMEOUT
USBCAN_ERRCMD_…

Note:

After the USB-CANmodul has been connected to the PC, all expansion port pins are configured as inputs (except sysWORXX modules, see above). This function can also be used to read back the states of ports configured as outputs.

UcanReadUserPortEx

Syntax:

```
UCANRET PUBLIC UcanReadUserPortEx (
    tUcanHandle          UcanHandle_p,
    BYTE*                pbInValue_p,
    BYTE*                pbLastOutEn_p,
    BYTE*                pbLastOutVal_p);
```

Usability:

HW_INIT, CAN_INIT , version 3.00 and higher

Description:

Reads the current input value from the expansion port. This function may be used alternatively for function *UcanReadUserPort()*.

Parameter:

UcanHandle_p:	USB-CAN handle received with the function *UcanInitHardware()* or *UcanInitHardwareEx()*.
pbInValue_p:	Address pointing to a variable that contains the read input value following the successful return of this function. This variable then contains the state of the 8-bit expansion port. Each bit in this parameter corresponds to matching pin on the expansion port.
pbLastOutEn_p:	Address pointing to a variable that contains the configuration data following the successful return of this function (configuration that was previously done with *UcanConfigUserPort()*). This parameter may be NULL.
pbLastOutVal_p:	Address pointing to a variable that contains the last output value following the successful return of this function. (output value that was written with *UcanWriteUserPort()*).This parameter may be NULL.

Return value: ***Error code of the function.***

USBCAN_SUCCESSFUL
USBCAN_ERR_MAXINSTANCES
USBCAN_ERR_ILLHANDLE
USBCAN_ERR_BUSY
USBCAN_ERR_IOFAILED
USBCAN_ERR_ILLHW
USBCAN_ERR_ILLPARAM
USBCAN_ERR_DATA
USBCAN_ERR_ABORT
USBCAN_ERR_DISCONNECT
USBCAN_ERR_TIMEOUT
USBCAN_ERRCMD_…

2.3.3 Error Codes of the Functions

The functions of the USBCAN-library return an error code with the type of UCANRET. Each return value represents an error. The only exception is the function *UcanReadCanMsg()* which can also return warnings. The warning *USBCAN_WARN_NODATA* indicates that no CAN messages are in the buffer. Other warnings show the calling function that an event has occurred but a valid CAN message is transferred.

All possible return codes for the functions of the USBCAN-library are listed below:

USBCAN_SUCCESSFUL

Value: 0x00

Description:

This message returns if the function is executed successfully.

USBCAN_ERR_RESOURCE

Value: 0x01

Description:

This error message returns if one resource could not be generated. In this case the term resource means memory and handles provided by Windows.

USBCAN_ERR_MAXMODULES

Value: 0x02

Description:

An application has tried to open more than 64 USB-CANmoduls. The standard version of the USBCAN-library only supports up to 64 USB-CANmoduls at the same time (under Windows CE only 9). This error also appears if several applications try to access more than 64 USB-CANmoduls. For example, application 1 has opened 60 modules, application 2 has opened 4 modules and application 3 wants to open a module. Application 3 receives this error message.

USBCAN_ERR_HWINUSE

Value: 0x03

Description:

An application tries to initialize a USB-CANmodul with the device number x. If this module has already been initialized by its own or by another application, this error message is returned.

USBCAN_ERR_ILLVERSION

Value: 0x04

Description:

This error message returns if the firmware version of the USB-CANmodul is not compatible to the software version of the USBCAN-library. In this case, install the USB-CAN driver again.

USBCAN_ERR_ILLHW

Value: 0x05

Description:

This error message returns if a USB-CANmodul with the device number x is not found. If the function *UcanInitHardware()* or *UcanInitHardwareEx()* has been called with the device number *USBCAN_ANY_MODULE,* and the error code appears, it indicates that no module is connected to the PC or all connected modules are already in use.

USBCAN_ERR_ILLHANDLE

Value: 0x06

Description:

This error message returns if a function received an incorrect USB-CAN handle. The function first checks which USB-CANmodul is initialized to this handle. This error occurs if no module has been initialized to this handle.

USBCAN_ERR_ILLPARAM

Value: 0x07

Description:

This error message returns if a wrong parameter is transferred to this function. For example, the value NULL has been handed over to a pointer variable instead of a valid address.

USBCAN_ERR_BUSY

Value: 0x08

Description:

This error message can occur if several threads are accessing a USB-CANmodul within a single application. After the other threads have finished their tasks, the function may be called again.

USBCAN_ERR_TIMEOUT

Value: 0x09

Description:

This error message occurs if the function transmits a command to the USB-CANmodul but no answer is returned. To solve this problem, close the application, disconnect the USB-CANmodul, and connect it again.

USBCAN_ERR_IOFAILED

Value: 0x0A

Description:

This error message occurs if the communication to the USB-CAN driver was interrupted. This happens, for example, if the USB-CANmodul is disconnected during the execution of a function.

USBCAN_ERR_DLL_TXFULL

Value: 0x0B

Description:

The function *UcanWriteCanMsg()* or *UcanWriteCanMsgEx()* first checks if the transmit buffer within the USBCAN-library has enough capacity to store new CAN messages. If the buffer is full, this error message returns. The CAN message transferred to the function *UcanWriteCanMsg()* or *UcanWriteCanMsgEx()* will not be written into the transmission buffer in order to protect other CAN messages from overwriting. Since software driver version 3.05 the size of the transmit buffer is configurable (*see function UcanInitCanEx() and Struktur tUcanInitCanParam*)

USBCAN_ERR_MAXINSTANCES

Value: 0x0C

Description:

In this software version, a maximum amount of 64 applications are able to have access to the USBCAN-library (under Windows CE only 9). If more applications attempt access to the DLL, this error message will occur. In this case, it is not possible to initialize a USB-CANmodul.

USBCAN_ERR_CANNOTINIT

Value: 0x0D

Description:

If a USB-CANmodul is initialized with the function *UcanInitHardware()* or *UcanInitHardwareEx(),* the software changes into the state HW_INIT. Functions like *UcanReadCanMsg()* or *UcanWriteCanMsg()* return this error message while in HW_INIT state. With the function *UcanInitCan(),* the software changes into CAN_INIT state. In this state, it is possible to read and transmit CAN messages.

USBCAN_ERR_DISCONNECT

Value: 0x0E

Description:

This error code occurs if a function from USBCAN-library was called for a USB-CANmodul that was plugged-off from the computer recently.

USBCAN_ERR_NOHWCLASS

Value: 0x0F

Description:

This error code is deprecated and is not used any more.

USBCAN_ERR_ILLCHANNEL

Value: 0x10

Description:

This error code is returned if an extended function of the USBCAN-library was called with parameter bChannel_p = USBCAN_CHANNEL_CH1, but USB-CANmodul GW-001, GW-002 or USB-CANmodul1 was used.

USBCAN_ERR_ILLHWTYPE

Value: 0x12

Description:

This error code occurs if an extended function of the USBCAN-library was called for a Hardware which does not support the feature.

USBCAN_ERRCMD_NOTEQU

Value: 0x40

Description:

This error code occurs during communication between the PC and a USB-CANmodul. The PC sends a command to the USB-CANmodul, then the module executes the command and returns a response to the PC. This error message returns if the answer does not correspond to the command.

USBCAN_ERRCMD_REGTST

Value: 0x41

Description:

The software tests the CAN controller on the USB-CANmodul when the CAN interface is initialized. Several registers of the CAN controller are checked. This error message returns if an error appears during this register test.

USBCAN_ERRCMD_ILLCMD

Value: 0x42

Description:

This error message returns if the USB-CANmodul receives a non-defined command. This error shows a version conflict between the firmware in the USB-CANmodul and the USBCAN-library.

USBCAN_ERRCMD_EEPROM

Value: 0x43

Description:

The USB-CANmodul has a serial EEPROM. This EEPROM contains the device number and the serial number. If an error occurs while reading these values, this error message is returned.

USBCAN_ERRCMD_ILLBDR

Value: 0x47

Description:

The Multiport CAN-to-USB 3004006, USB-CAnmodul1 3204000/3204001 or USB-CANmodul2 3204002/3204003 has been initialized with an invalid baud rate (BTR0 und BTR1).

USBCAN_ERRCMD_NOTINIT

Value: 0x48

Description:

It was tried to access a CAN-channel of Multiport CAN-to-USB 3004006 or USB-CANmodul2 3204002/3204003 that was not initialized.

USBCAN_ERRCMD_ALREADYINIT

Value: 0x49

Description:

The accessed CAN-channel of Multiport CAN-to-USB 3004006 or USB-CANmodul2 3204002/3204003 was already initialized.

USBCAN_ERRCMD_ILLSUBCMD

Value: 0x4A

Description:

An internal error occurred in USBCAN Library. In this case an unknown sub-command was called instead of a main command (e.g for the cyclic CAN message-feature).

USBCAN_ERRCMD_ILLIDX

Value: 0x4B

Description:

An internal error occurred in USBCAN Library. In this case an invalid index for a list was delivered to the firmware (e.g. for the cyclic CAN message-feature).

USBCAN_ERRCMD_RUNNING

Value: 0x4C

Description:

The caller tries to define a new list of cyclic CAN messages but this feature was already started. For defining a new list, it is necessary to switch off the feature beforehand.

USBCAN_WARN_NODATA

Value: 0x80

Description:

If the function *UcanReadCanMsg()* returns with this warning, it is an indication that the receive buffer contains no CAN messages.

USBCAN_WARN_SYS_RXOVERRUN

Value: 0x81

Description:

If an overrun in the receive buffer on the USB-CAN system driver occurred, the USBCAN-library is informed about this event. The function *UcanReadCanMsg()* returns this warning and a valid CAN message. The warning indicates that CAN messages are lost. However, it does not indicate the position of the lost CAN messages.

USBCAN_WARN_DLL_RXOVERRUN

Value: 0x82

Description:

The USBCAN-library automatically requests CAN messages from the USB-CANmodul and stores the messages into a buffer of the DLL. If more CAN messages are received than the DLL buffer size allows, this error message returns and CAN messages are lost. However, it does not indicate the position of the lost CAN messages.

Since software driver version 3.05 the size of the receive buffer is configurable (*see function UcanInitCanEx() and structure tUcanInitCanParam*)

USBCAN_WARN_FW_TXOVERRUN

Value: 0x85

Description:

This warning is returned by function *UcanWriteCanMsg()* and/or *UcanWriteCanMsgEx()* if flag USBCAN_CANERR_QXMTFULL is set in the CAN driver status. However, the transmit CAN message could be stored to the DLL transmit buffer. This warning indicates that at least one transmit CAN message got lost in the device firmware layer. This warning does not indicate the position of the lost CAN message.

USBCAN_WARN_FW_RXOVERRUN

Value: 0x86

Description:

This warning is returned by function *UcanWriteCanMsg()* and/or *UcanWriteCanMsgEx()* if flag USBCAN_CANERR_QOVERRUN or flag USBCAN_CANERR_OVERRUN are set in the CAN driver status. The function has returned with a valid CAN message. This warning indicates that at least one reveived CAN message got lost in the firmware layer. This warning does not indicate the position of the lost CAN message.

USBCAN_WARN_NULL_PTR

Value: 0x90

Description:

This warning message is returned by functions: *UcanInitHwConnectControl()* and/or *UcanInitHwConnectControlEx()* if a NULL pointer was passed as callback function address.

USBCAN_WARN_TXLIMIT

Value: 0x91

Description:

This warning message is returned by the function *UcanWriteCanMsgEx()* if it was called to transmit more than one CAN message, but a part of them could not be stored to the transmit buffer within USBCAN-library (because the buffer is full). The parameter *pdwCount_p* includes the number of CAN messages which could be stored successfully to the transmit buffer.

2.3.4 Baud Rate Configuration

The baud rate configuration for USB-CANmodul GW-001 and GW-002 is transferred to the function *UcanInitCan()* and *UcanInitCanEx2()* as parameter *bBTR0_p* and *bBTR1_p*. The configuration can also be changed later by calling the function *UcanSetBaudrate()* resp. *UcanSetBaudrateEx()*. The following values are recommended:

USBCAN_BAUD_10kBit:	0x672f	// CAN baud rate 10 kBit/sec
USBCAN_BAUD_20kBit:	0x532f	// CAN baud rate 20 kBit/sec
USBCAN_BAUD_50kBit:	0x472f	// CAN baud rate 50 kBit/sec
USBCAN_BAUD_100kBit:	0x432f	// CAN baud rate 100 kBit/sec
USBCAN_BAUD_125kBit:	0x031c	// CAN baud rate 125 kBit/sec
USBCAN_BAUD_250kBit:	0x011c	// CAN baud rate 250 kBit/sec
USBCAN_BAUD_500kBit:	0x001c	// CAN baud rate 500 kBit/sec
USBCAN_BAUD_800kBit:	0x0016	// CAN baud rate 800 kBit/sec
USBCAN_BAUD_1MBit:	0x0014	// CAN baud rate 1 MBit/sec

Example:

```
tUcanHandle UcanHandle;
UCANRET bRet;

...
// initializes the hardware
bRet = UcanInitHardware (&UcanHandle, 0, NULL);
...

// initializes the CAN interface
bRet = UcanInitCan (UcanHandle,
  HIBYTE (USBCAN_BAUD_1MBit),    // BTR0 for 1MBit/s
  LOBYTE (USBCAN_BAUD_1MBit),    // BTR1 for 1MBit/s
  0xFFFFFFFF,                    // AMR: all messages received
  0x00000000);                   // ACR

// Error? print error
if (bRet != USBCAN_SUCCESSFUL)
    PrintError (bRet);
...
```

Configuration of other baud rates is also possible. The structure of the BTR0 and BTR1 registers is described below. Refer to the SJA1000 Data Sheet for detailed description.

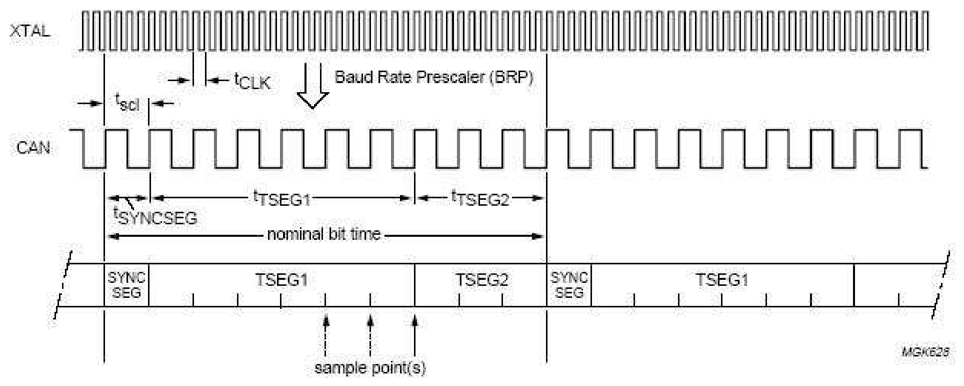
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
SJW		BPR					

Figure 20: Structure of baud rate register BTR0

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
SAM	TSEG2			TSEG1			

Figure 21: Structure of baud rate register BTR1

BPR:	*Baudrate Prescaler* specifies the ratio between system clock of the SJA1000 and the bus clock on the CAN-bus.
SJW:	*Synchronization Jump Width* specifies the compensation of the phase-shift between the system clock and the different CAN-controllers connected to the CAN-bus.
SAM:	*Sampling* specifies the number of sample points used for reading the bits on the CAN-bus. If SAM=1 three sample points are used, otherwise only one sample point is used.
TSEG:	*Time Segment* specifies the number of clock cycles of one bit on the CAN-bus as well as the position of the sample points.



Possible values are BRP = 000001, TSEG1 = 0101 and TSEG2 = 010.

Figure 22: General structure of a single bit on the CAN-bus (source: SJA1000 manual)

The following mathematical connections apply:

t_{CLK}	=	1 / 16MHz	(system clock)
t_{scl}	=	$2 * t_{CLK} * (BRP + 1)$	(bus clock)
$t_{SYNCSEG}$	=	$1 * t_{scl}$	
t_{TSEG1}	=	$t_{scl} * (TSEG1 + 1)$	
t_{TSEG2}	=	$t_{scl} * (TSEG2 + 1)$	
t_{Bit}	=	$t_{SYNCSEG} + t_{TSEG1} + t_{TSEG2}$	(time of one Bit on the CAN-bus)

Example for 125 kBit/s (TSEG1 = 1, TSEG2 = 12, BPR = 3):

t_{scl}	=	$2 * t_{CLK} * 4$	= 500 ns
$t_{SYNCSEG}$	=	$1 * t_{scl}$	= 500 ns
t_{TSEG1}	=	$t_{scl} * 2$	= 1000 ns
t_{TSEG2}	=	$t_{scl} * 13$	= 6500 ns
t_{Bit}	=	$t_{SYNCSEG} + t_{TSEG1} + t_{TSEG2}$	= 8000 ns
$1 / t_{Bit}$	=	125 kBit/sec	

Note:

The configuration of the baud rate differs significantly between the older USB-CANmodul versions (GW-001 and GW-002) and the new sysWORXX modules. For standardized baud rate values (*see section 2.3.4*), the baud rate registers BTR0 and BTR1 are as well applicable for the new sysWORXX modules. Therefore set *m_dwBaudrate* to USBCAN_BAUDEX_USE_BTR01.

The following default values are available for sysWORXX modules:

USBCAN_BAUDEX_1MBit	0x00020354	// CAN baud rate 1 MBit/sec
USBCAN_BAUDEX_800kBit	0x00030254	// CAN baud rate 800 kBit/sec
USBCAN_BAUDEX_500kBit	0x00050354	// CAN baud rate 500 kBit/sec
USBCAN_BAUDEX_250kBit	0x000B0354	// CAN baud rate 250 kBit/sec
USBCAN_BAUDEX_125kBit	0x00170354	// CAN baud rate 125 kBit/sec
USBCAN_BAUDEX_100kBit	0x00171466	// CAN baud rate 100 kBit/sec
USBCAN_BAUDEX_50kBit	0x002F1466	// CAN baud rate 50 kBit/sec
USBCAN_BAUDEX_20kBit	0x00771466	// CAN baud rate 20 kBit/sec
USBCAN_BAUDEX_10kBit	0x80771466	// CAN-baud rate 10 kBit/sec

The following values have a sample point between 85 and 90%:

USBCAN_BAUDEX_SP2_1MBit	*0x00020741*	*// CAN-Baudrate 1 MBit/sec*
USBCAN_BAUDEX_SP2_800kBit	*0x00030731*	*// CAN-Baudrate 800 kBit/sec*
USBCAN_BAUDEX_SP2_500kBit	*0x00050741*	*// CAN-Baudrate 500 kBit/sec*
USBCAN_BAUDEX_SP2_250kBit	*0x000B0741*	*// CAN-Baudrate 250 kBit/sec*
USBCAN_BAUDEX_SP2_125kBit	*0x00170741*	*// CAN-Baudrate 125 kBit/sec*
USBCAN_BAUDEX_SP2_100kBit	*0x001D1741*	*// CAN-Baudrate 100 kBit/sec*
USBCAN_BAUDEX_SP2_50kBit	*0x003B1741*	*// CAN-Baudrate 50 kBit/sec*
USBCAN_BAUDEX_SP2_20kBit	*0x00771772*	*// CAN-Baudrate 20 kBit/sec*
USBCAN_BAUDEX_SP2_10kBit	*0x80771772*	*// CAN-Baudrate 10 kBit/sec*

Configuration of baud rates other than the values given above is possible. The register structure for extended baud rate configuration is given below.

Bit 31	**30**	**29**	**28**	**27**	**26**	**25**	**24**
CLK	-						SMP
23	**22**	**21**	**20**	**19**	**18**	**17**	**16**
-	BPR						
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**
-		SYNC		-	PROPAG		
7	**6**	**5**	**4**	**3**	**2**	**1**	**Bit 0**
-	PHASE1			-	PHASE2		

Figure 23: Format of the extended baud rate register for Multiport and USB-CANmodul1/2

BPR:	*Baudrate Prescaler* specifies the ration between system clock of the microcontroller and the bus clock on CAN-bus.
SYNC:	*Synchronization Jump Width* specifies the compensation of phase shift between the system clock and the different CAN controllers connected to the CAN-bus.
SAM:	*Sampling* specifies the number of sample points used for reading the bits on the CAN-bus. If SAM=1 three sample points are used, otherwise only one sample point is used.
PROPAG:	*Programming Time Segment* specifies the compensation of the physical delay time on the CAN-bus.
PHASE:	*Time Segment* specifies the number of clock cycles of one bit on the CAN-bus as well as the position of the sample points.
CLK:	*Clock* specifies the frequency of the microcontroller. If set to 0, then the microcontroller runs with 48 MHz clock cycle internally, otherwise with 24 MHz. This influences the CAN-bus baud rate (see system clock t_{MCK} in the example below)

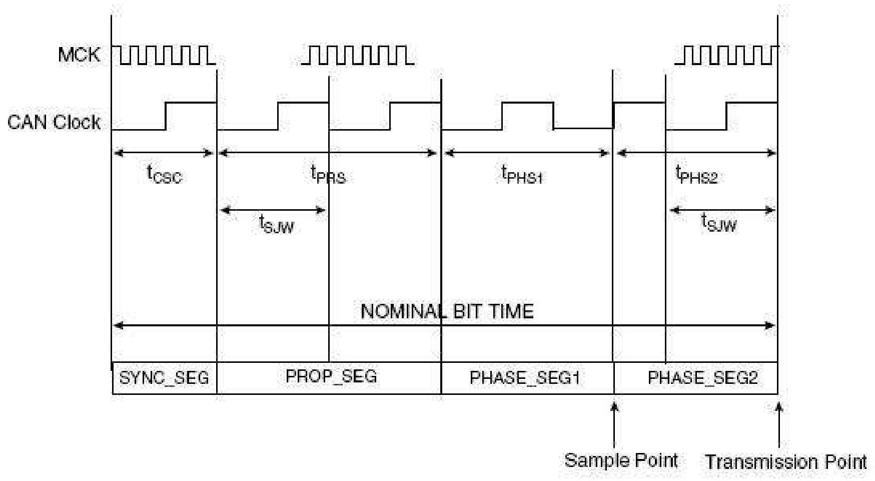


Figure 24: Generic structure of one bit on the CAN-bus (source: Atmel AT91SAM7A3 manual)

The following mathematical connections apply:

t_{MCK0}	=	1 / 48MHz	(system clock CLK=0)
t_{MCK1}	=	1 / 24MHz	(system clock CLK=1)
t_{CSC}	=	t_{MCKx} * (BRP + 1)	(bus clock)
$t_{SYNCSEG}$	=	1 * t_{CSC}	
t_{PRS}	=	t_{CSC} * (PROPAG + 1)	
t_{PHS1}	=	t_{CSC} * (PHASE1 + 1)	
t_{PHS2}	=	t_{CSC} * (PHASE2 + 1)	
t_{Bit}	=	$t_{SYNCSEG} + t_{PRS} + t_{PHS1} + t_{PHS2}$	(time of one bit on CAN-bus)

Example for 125 kBit/s (PROPAG = 3, PHASE1 = 5, PHASE2 = 4, BPR = 23, CLK=0):

t_{CSC}	= t_{MCK0} * 24	= 500 ns
$t_{SYNCSEG}$	= 1 * t_{scl}	= 500 ns
t_{PRS}	= t_{CSC} * 4	= 2000 ns
t_{PHS1}	= t_{CSC} * 6	= 3000 ns
t_{PHS2}	= t_{CSC} * 5	= 2500 ns
t_{Bit}	= $t_{SYNCSEG}$ + t_{PRS} + t_{PHS1} + t_{PHS2}	= 8000 ns
1 / t_{Bit}	= 125 kBit/sec	

Note:

For compatibility reasons, constant USBCAN_BAUDEX_USE_BTR01 was defined. If this constant is used for baud rate configuration of sysWORXX modules, the BTR0 and BTR1 registers become available for configuration. In this case, only the baud rates given in this manual are available. Configuration of user-specific baud rates is not possible (error code USBCAN_ERRCMD_ILLBDR)

Example 1:

```
tUcanHandle UcanHandle;
UCANRET bRet;
tUcanInitCanParam InitParam;

...

// preset init parameters
memset (&InitParam, 0, sizeof (InitParam));
InitParam.m_dwSize       = sizeof (InitParam);
InitParam.m_bMode        = kUcanModeNormal;
InitParam.m_bBTR0        = HIBYTE (USBCAN_BAUD_1MBit);
InitParam.m_bBTR1        = LOBYTE (USBCAN_BAUD_1MBit);
InitParam.m_bOCR         = USBCAN_OCR_DEFAULT;
InitParam.m_dwAMR        = USBCAN_AMR_ALL;
InitParam.m_dwACR        = USBCAN_ACR_ALL;
InitParam.m_dwBaudrate   = USBCAN_BAUDEX_USE_BTR01;
InitParam.m_wNrOfRxBufferEntries = USBCAN_DEFAULT_BUFFER_ENTRIES;
InitParam.m_wNrOfTxBufferEntries = USBCAN_DEFAULT_BUFFER_ENTRIES;

// initialize CAN-channel
bRet = UcanInitCanEx2 (UcanHandle, USBCAN_CHANNEL_CH0,
    &InitParam);
...
```

Example 2:

```
tUcanHandle UcanHandle;
UCANRET bRet;
tUcanInitCanParam InitParam;

...

// preset init parameters
memset (&InitParam, 0, sizeof (InitParam));
InitParam.m_dwSize       = sizeof (InitParam);
InitParam.m_bMode        = kUcanModeNormal;
InitParam.m_bBTR0        = HIBYTE (USBCAN_BAUD_USE_BTREX);
InitParam.m_bBTR1        = LOBYTE (USBCAN_BAUD_USE_BTREX);
InitParam.m_bOCR         = USBCAN_OCR_DEFAULT;
InitParam.m_dwAMR        = USBCAN_AMR_ALL;
InitParam.m_dwACR        = USBCAN_ACR_ALL;
InitParam.m_dwBaudrate   = USBCAN_BAUDEX_SP2_125kBit;
InitParam.m_wNrOfRxBufferEntries = USBCAN_DEFAULT_BUFFER_ENTRIES;
InitParam.m_wNrOfTxBufferEntries = USBCAN_DEFAULT_BUFFER_ENTRIES;

// initialize CAN-channel
bRet = UcanInitCanEx2 (UcanHandle, USBCAN_CHANNEL_CH0,
    &InitParam);
...
```

2.3.5 CAN Messages Filter Function

It is possible to filter the received CAN messages. The SJA1000 CAN controller automatically filters messages in PeliCAN mode (Single-Filter-Mode).

The configurations of the filter are transferred to the function *UcanInitCan()* as parameter *dwAMR_p* and *dwACR_p*. It is also possible to change these values later after calling the function *UcanInitCan()* with the function *UcanSetAcceptance()*.

The following mechanism is used for filtration:

AMR Bit	ACR Bit	Bit of the CAN ID
0	0	0
0	1	1
1	0	x
1	1	x

0	The corresponding bit of the CAN identifier has to be 0.
1	The corresponding bit of the CAN identifier has to be 1.
x	The corresponding bit of the CAN identifier can be either 0 or 1.

These bits correspond to:

a) Standard frame (11-bit identifier) for GW-001/GW-002 as well as sysWORXX modules since firmware version 3.10:

dwACR_p															
31	30	...	21	20	19	...	16	15	14	...	8	7	6	...	0

⇓

dwAMR_p															
31	30	...	21	20	19	...	16	15	14	...	8	7	6	...	0

⇓

CAN Identifier						Data Byte 0				Data Byte 1			
10	9	...	0	RTR	free	7	6	...	0	7	6	...	0

b) Extended frame (29-bit identifier) for GW-001/GW-002 as well as sysWORXX modules since firmware version 3.10:

dwACR_p						
31	30	...	3	2	1	0

⇓

dwAMR_p						
31	30	...	3	2	1	0

⇓

CAN Identifier					
28	27	...	0	RTR	free

c) Standard-Frame (11-Bit-Identifier) for all sysWORXX modules until firmware version 3.09:

dwACR_p															
31	30	...	21	20	19	...	16	15	14	...	8	7	6	...	0

⇓

dwAMR_p															
31	30	...	21	20	19	...	16	15	14	...	8	7	6	...	0

⇓

CAN-Identifier				
10	9	...	0	Not in use

d) Extended-Frame (29-Bit-Identifier) for all sysWORXX modules until firmware version 3.09:

dwACR_p						
31	30	...	3	2	1	0

⇓

dwAMR_p						
31	30	...	3	2	1	0

⇓

CAN-Identifier				
28	27	...	0	Not in use

The macros *USBCAN_SET_AMR(extended, can_id, rtr)* and *USBCAN_SET_AMR(extended,can_id, rtr)* as well as *USBCAN_CALCULATE_AMR(extended, from_id, to_id, rtr_only, rtr_too)* und *USBCAN_CALCULATE_ACR(extended, from_id, to_id, rtr_only, rtr_too)* can be used to calculate the filter values.

The parameter *extended* indicates if the parameters *can_id*, *from_id* and *to_id specify* a 29-bit message (TRUE) or an 11-bit message (FALSE). The parameter *can_id* shows the filter value as CAN identifier. The parameters *from_id* and *to_id* specify the filter range for CAN-identifiers. The parameters *rtr, rtr_only* and *rtr_too* can be used to filter RTR frames. These parameters can be TRUE (=1) or FALSE (=0). If the parameter rtr_only is TRUE, then only RTR Frames are received and parameter rtr_too is ignored. Otherwise also RTR Frames are received, if rtr_too is set to TRUE.

Example 1:

```
tUcanHandle UcanHandle;
UCANRET bRet;

...
// initializes the hardware
bRet = UcanInitHardware (&UcanHandle, 0, NULL);
...

// initializes the CAN interface
// filters 11-bit CAN messages with ID 0x300 to 0x3ff,
// RTR unimportant
bRet = UcanInitCan (UcanHandle,
  HIBYTE (USBCAN_BAUD_1MBit),
  LOBYTE (USBCAN_BAUD_1MBit),
  USBCAN_SET_AMR (FALSE, 0x0ff, 1),
  USBCAN_SET_ACR (FALSE, 0x300, 0));

// Error? print error
if (bRet != USBCAN_SUCCESSFUL)
    PrintError (bRet);
...
```

If, according to example 1, the CAN-controller receives a CAN-message with 29-bit identifier, then it is filtered with the same configuration. However, AMR and ACR are interpreted differently. Here, CAN-messages with 29-bit identifiers from 0x0C000000 to 0FFFFFFF are received too.

Example 2:

```
tUcanHandle UcanHandle;
UCANRET bRet;

...
// initializes the hardware
bRet = UcanInitHardware (&UcanHandle, 0, NULL);

// initializes the CAN-interface
// filters 11-bit CAN-messages with ID 0x600 to 0x67F,
// RTR-Frames unimportant
bRet = UcanInitCan (UcanHandle,
  HIBYTE (USBCAN_BAUD_1MBit),
  LOBYTE (USBCAN_BAUD_1MBit),
  USBCAN_CALCULATE_AMR (FALSE, 0x600, 0x67F, FALSE, FALSE),
  USBCAN_CALCULATE_ACR (FALSE, 0x600, 0x67F, FALSE, FALSE));

// error? Print error
if (bRet != USBCAN_SUCCESSFUL)
    PrintError (bRet);
...
```

Filter parameters for RTR frames and the first two data bytes are ignored with all sysWORXX modules.

2.3.6 Using multiple CAN-channels

The USB-CANmodul2 3204002/3204003/3204007 has 2 CAN-channels and both the Multiport CAN-to-USB 3004006 and USB-CANmodul16 have 16 CAN-channels which are divided into 8 logical devices with 2 channels each. In other words, each logical device provides 2 CAN-channels, which need to get initialized. In order to use all 16 channels, each logical device has to get initialized with function *UcanInitHardware()* and/or *UcanInitHardwareEx()*. Furthermore, both CAN-channels of each logical device have to be initialized by function *UcanInitCanEx2().* The USB-CANmodul8 behaves like USB-CANmodul16 but includes only 4 logical devices and 8 CAN-channels. USB-CANmodul2 has only one logical device and 2 CAN-channels.

There are 3 constants to select a CAN-channel:

Constant	Value	Meaning
USBCAN_CHANNEL_CH0	0	first CAN channel
USBCAN_CHANNEL_CH1	1	second CAN channel
USBCAN_CHANNEL_ANY	255	Any CAN channel
USBCAN_CHANNEL_CAN1	0	first CAN channel
USBCAN_CHANNEL_CAN2	1	second CAN channel

Table 18: Constants for CAN-channel selection

Constant USBCAN_CHANNEL_ANY can only be used with function *UcanReadCanMsgEx()* and/or *UcanGetMsgPending()*. For the function *UcanReadCanMsgEx()* it indicates that the function shall examine, from which CAN-channel the next CAN message is. If this function returns with error code USBCAN_SUCCESSFUL, then it also passes the respective CAN-channel to the calling function: USBCAN_CHANNEL_CH0 or USBCAN_CHANNEL_CH1 (*see function UcanReadCanMsgEx()*).

The constants USBCAN_CHANNEL_CAN1 and USBCAN_CHANNEL_CAN2 have the same values as USBCAN_CHANNEL_CH0 and USBCAN_CHANNEL_CH1. They were defined because on top of the housing of USB-CANmodul2, the first channel was named CAN1 but in the software the first channel is named 0.

2.3.7 Using the Callback Functions

The USBCAN-library provides two types of callback functions. The Connect Control Callback function announces Plug&Play events for the USB-CANmodul (e.g.: new USB-CANmodul connected with the PC; or taken off; ...). The second type announces events, which occur during the work with the USB-CANmodul (e.g.: CAN-message receive; Error status changed; ...).

From software version 3.00 an extended format (support of multiple CAN-channels) exists for both types of the callback function.

Note:

The "Connect control callback" function has a different format than callback functions for the other events. Make sure to use the correct format in your application. It is not possible to use the very same implementation for both types of callback function!

Also the format of the extended callback functions differs from the format of the standard functions. Make sure to use the extended callback functions if the extended API functions are used. Access violations will occur during runtime otherwise!

Also note that the callback functions are declared as PUBLIC, which is defined as “__stdcall” in Microsoft Visual Studio.

UcanConnectControlFkt

Syntax:

```
void PUBLIC UcanConnectControlFkt (
     BYTE                    bEvent_p,
     DWORD                   dwParam_p);
```

Description:

This callback function informs the application program if a new USB-CANmodul is connected to the PC, or a connected USB-CANmodul has been disconnected. This callback function is registered with the USBCAN-library by function *UcanInitHwConnectControl*() and may have a different name within the application.

Parameter:

bEvent_p:	Event which occurred. *USBCAN_EVENT_CONNECT = 6* *USBCAN_EVENT_DISCONNECT = 7* *USBCAN_EVENT_FATALDISCON = 8*
dwParam_p:	Additional parameter If bEvent_p = 8, then the parameter is returned from the USB-CAN-Handle of the disconnected module. No messages are received from this module and no messages can be sent. The corresponding USB CAN-Handle is invalid; in other instances, the parameter is 0.

UcanConnectControlFktEx

Syntax:

```
void PUBLIC UcanConnectControlFktEx (
    DWORD                dwEvent_p,
    DWORD                dwParam_p,
    void*                pArg_p);
```

Description:

This callback function informs the application program if a new device is connected to the PC, or a connected device has been disconnected. This callback function is registered with the USBCAN-library by function *UcanInitHwConnectControlEx*() and may have a different name within the application.

Parameter:

bEvent_p:	Event which occurred. *USBCAN_EVENT_CONNECT = 6* *USBCAN_EVENT_DISCONNECT = 7* *USBCAN_EVENT_FATALDISCON = 8*
dwParam_p:	Additional parameter If *bEvent p = 8*, then the parameter is returned from the USB-CAN-Handle of the disconnected module. No messages are received from this module and no messages can be sent. The corresponding USB-CAN-Handle is invalid; in other instances, the parameter is 0.
pArg_p:	Additional user-parameter, which was handed over to function *UcanInitHwConnectControlEx().* as parameter *pCallbackArg_p*

UcanCallbackFkt

Syntax:

```
void PUBLIC UcanCallbackFkt (
     tUcanHandle             UcanHandle_p,
     BYTE                    bEvent_p);
```

Description:

This callback function informs the application program if an event occurred on an initialized USB-CANmodul. This callback function is registered with the USBCAN-library by function *UcanInitHardware()* and may have a different name within the application.

Parameter:

UcanHandle_p:	USB-CAN handle of the USB-CANmodul where the event occurred. This handle is returned with the function *UcanInitHardware()*.
bEvent_p:	Event which occurred.

USBCAN_EVENT_INITHW	= 0
USBCAN_EVENT_INITCAN	= 1
USBCAN_EVENT_RECEIVE	= 2
USBCAN_EVENT_STATUS	= 3
USBCAN_EVENT_DEINITCAN	= 4
USBCAN_EVENT_DEINITHW	= 5

UcanCallbackFktEx

Syntax:

```
void PUBLIC UcanCallbackFktEx (
     tUcanHandle            UcanHandle_p,
     DWORD                  dwEvent_p,
     BYTE                   bChannel_p,
     void*                  pArg_p);
```

Description:

This callback function informs the application program if an event occurred on an initialized device. This callback function is registered with the USBCAN-library by function *UcanInitHardwareEx*() and may have a different name within the application.

Parameter:

UcanHandle_p:	USB-CAN handle of the USB-CANmodul where the event occurred. This handle is returned with the function *UcanInitHardwareEx()*.
bEvent_p:	Event which occurred.

USBCAN_EVENT_INITHW	= 0
USBCAN_EVENT_INITCAN	= 1
USBCAN_EVENT_RECEIVE	= 2
USBCAN_EVENT_STATUS	= 3
USBCAN_EVENT_DEINITCAN	= 4
USBCAN_EVENT_DEINITHW	= 5

bChannel_p: CAN-channel, which released the event

USBCAN_CHANNEL_CH0	= 0
USBCAN_CHANNEL_CH1	= 1
USBCAN_CHANNEL_ANY	= 255

pArg_p: Additional user-parameter, which was handed over to function *UcanInitHwConnectControlEx()* as parameter *pCallbackArg_p*

All callback functions have to be initialized before being called by the USBCAN-library.

The events have the following meaning:

USBCAN_EVENT_INITHW: 0x00

The USB-CANmodul is initialized successfully.

Parameter *bChannel_p* has no meaning here.

USBCAN_EVENT_INITCAN: 0x01

The CAN interface is initialized successfully.

Parameter *bChannel_p* returns the CAN-channel that was initialized.

USBCAN_EVENT_RECEIVE: 0x02

A CAN message is received.

Parameter *bChannel_p* returns the CAN-channel that was received last by the hardware.

USBCAN_EVENT_STATUS: 0x03

The error status at the USB-CANmodul has changed.

Parameter *bChannel_p* returns the CAN-channel, which CAN error state has been changed.

USBCAN_EVENT_DEINITCAN: 0x04

The CAN interface is shut down.
Parameter *bChannel_p* returns the CAN-channel that is being shut down.

USBCAN_EVENT_DEINITHW: 0x05

The USB-CANmodul is shut down.

Parameter *bChannel_p* has no meaning here.

USBCAN_EVENT_CONNECT: 0x06

A new USB-CANmodul is connected.

Parameter *dwParam_p* has no meaning here.

USBCAN_EVENT_DISCONNECT: 0x07

A USB-CANmodul is disconnected.

Parameter *dwParam_p* has no meaning here.

USBCAN_EVENT_FATALDISCON: 0x08

A USB-CANmodul in either HW_INIT or CAN_INIT state is disconnected from the computer. Data loss is possible.

The parameter *dwParam p* contains the USB-CAN handle of the disconnected module. The handle can no longer be used.

Note:

The callback functions should not call the functions of the USBCAN-library directly. This can lead to undesired results. The best method for using the callback functions is to wait for an event in the main program (e.g. with the Win32 function *WaitForMultipleObjects()*) and then to call the DLL functions from there after the event has occurred. The callback functions only set the corresponding event (i.e. with the Win32 function *SetEvent()*).

Example:

```
tUcanHandle UcanHandle_g;
tCanMsgStruct CanRxMsg_g;
...

void main (void)
{
UCANRET bRet;

  // initilizes the first callback function
  bRet = UcanInitHwConnectControl (UcanConnectControlFkt);

  if (bRet == USBCAN_SUCCESSFUL)
  {
      // wait for event
      // e.g. with WaitForMultipleObjects(...) function
      // react to events accordingly:

      case INIT:
      // open USB-CANmodul with USBCAN_ANY_MODULE and
      // initialize second callback function
      bRet = UcanInitHardware (&UcanHandle_g, USBCAN_ANY_MODULE,
          UcanCallbackFkt);

      // initialize CAN interface
      bRet = UcanInitCan (UcanHandle_g, 0x00, 0x14, 0xFFFFFFFFL,
          0x00000000L);

      case RECV:
      // read CAN message
      bRet = UcanReadCanMsg (UcanHandle_g, &CanRxMsg_g);
  }
  ...
}
void PUBLIC UcanConnectControlFkt (BYTE bEvent_p, DWORD
    dwParam_p)
{
UCANRET bRet;

  // which event did occur?
  switch (bEvent_p)
  {
    // new USB-CANmodul connected
    case USBCAN_EVENT_CONNECT:
      // Send signal to main function, so that the USB-CANmodul
      // can now be initialized.
      // i.e. with SetEvent(INIT)
      ...
      break;
    // USB-CANmodul disconnected
    case USBCAN_EVENT_DISCONNECT:
      ...
      break;
  }
}
```

```
void PUBLIC UcanCallbackFkt (tUcanHandle UcanHandle_p,
  BYTE bEvent_p)
{
  // what event appeared?
  switch (bEvent_p)
  {
    // CAN message received
    case USBCAN_EVENT_RECEIVE:
      // signal that the CAN message can be read
      // i.e. with SetEvent (RECV);
      break;

    // changes error status
    case USBCAN_EVENT_STATUS:
      // signal that the CAN status can be read
      // i.e. with SetEvent (STATUS);
      break;
    ...
  }
}
```

2.4 Class library for .NET programming languages

In order to use the USBCAN32.DLL with.NET programming languages such as Visual basic .NET, Managed C++ and C#, a Wrapper class UcanDotNET.USBcanServer was developed in VB .NET. This class lies in the dynamic link library (DLL) named UCANDOTNET.DLL. In order to include the DLL into own projects, the following steps under Visual Studio .NET become necessary:

- go to menu -> “Projects” and click on entry “Add reference…”, the dialog window “Add reference” appears
- Click on “Browse” button and select the UCANDOTNET.DLL
- Press OK to confirm

Sample for creating an object of class USBcanServer in Visual Basic .NET:

```
Dim WithEvents m_USBcan As UcanDotNET.USBcanServer
```

Note:

The Wrapper class is included as source code. That means it can be changed by user expanding new features of the USBCAN32.DLL or improving the Wrapper.

2.4.1 Methods of class USBcanServer

GetFwVersion

Syntax C#:

```
public int USBcanServer.GetFwVersion();
```

Syntax Visual Basic:

```
Public USBcanServer.GetFwVersion() as Integer
```

Usability:

HW_INIT, version 3.01 and higher

Description:

Returns the firmware version number of the device.

Parameter:

none

Return value:

Firmware version number as Integer with the following format:

Bit 0-7:	Version
Bit 8-15:	Revision
Bit 16-31:	Release

GetUserDllVersion

Syntax C#:

```
public int USBcanServer.GetUserDllVersion();
```

Syntax Visual Basic:

```
Public USBcanServer.GetUserDllVersion() as Integer
```

Usability:

DLL_INIT, HW_INIT, CAN_INIT, version 3.01 and higher

Description:

Returns the version number of the USBCAN-library.

Parameter:

none

Return value:

Software version number as Integer with the following format:

Bit 0-7:	Version
Bit 8-15:	Revision
Bit 16-31:	Release

InitHardware

Syntax C#:

```
public byte USBcanServer.InitHardware(byte bDeviceNr_p =
     USBCAN_ANY_MODULE);
```

Syntax Visual Basic:

```
Public Function USBcanServer.InitHardware( _
     Optional ByVal bDeviceNr_p As Byte _
       = USBCAN_ANY_MODULE) As Byte
```

Usability:

DLL_INIT, HW_INIT, CAN_INIT, version 3.01 and higher

Description:

Initializes the device with the corresponding device number.

Parameter:

bDeviceNr_p:	device number (0 – 254). Value *USBCAN_ANY_MODULE* (= 255) is used to indicate that the first available device shall be used.

Return value:

Error code of the function.

USBCAN_SUCCESSFUL
USBCAN_ERR_MAXINSTANCES
USBCAN_ERR_HWINUSE
USBCAN_ERR_ILLHW
USBCAN_ERR_MAXMODULES
USBCAN_ERR_RESOURCE
USBCAN_ERR_ILLVERSION

Shutdown

Syntax C#:

```
public byte USBcanServer.Shutdown();
```

Syntax Visual Basic:

```
Public Function USBcanServer.Shutdown() as Byte
```

Usability:

HW_INIT, version 3.01 and higher

Description:

Shuts down an initialized device that was initialized with method *InitHardware()* or *InitCan().* The software returns to the state DLL_INIT.

Parameter:

none

Return value:

Error code of the function.

USBCAN_SUCCESSFUL
USBCAN_ERR_MAXINSTANCES
USBCAN_ERR_ILLHANDLE

InitCan

Syntax C#:

```
public USBcanServer.InitCan(
     byte bChannel_p = USBCAN_CHANNEL_CH0,
     short wBTR_p = USBCAN_BAUD_1MBit,
     int dwBaudrate_p = USBCAN_BAUDEX_USE_BTR01,
     int dwAMR_p = USBCAN_AMR_ALL,
     int dwACR_p = USBCAN_ACR_ALL,
     byte bMode_p = tUcanMode.kUcanModeNormal,
     byte bOCR_p = USBCAN_OCR_DEFAULT);
```

Syntax Visual Basic:

```
Public Function InitCan(Optional ByVal bChannel_p As Byte = _
     USBCAN_CHANNEL_CH0, _
     Optional ByVal wBTR_p As Short = USBCAN_BAUD_1MBit, _
     Optional ByVal dwBaudrate_p As Integer = _
     USBCAN_BAUDEX_USE_BTR01, _
     Optional ByVal dwAMR_p As Integer = USBCAN_AMR_ALL, _
     Optional ByVal dwACR_p As Integer = USBCAN_ACR_ALL, _
     Optional ByVal bMode_p As Byte = _
     tUcanMode.kUcanModeNormal, _
     Optional ByVal bOCR_p As Integer = USBCAN_OCR_DEFAULT)
     As Byte
```

Usability:

HW_INIT, version 3.01 and higher

Description:

Initializes a specific CAN-channel of a device. With GW-001 and GW-002 only channel 0 is available.

Parameter:

bChannel_p:	CAN-channel, to be initialized. USBCAN_CHANNEL_CH0 for CAN- channel 0 USBCAN_CHANNEL_CH1 for CAN- channel 1
wBTR_p:	Baud rate register BTR0 as high byte, Baud rate register BTR1 as low byte
dwBaudrate_p:	Baud rate register of Multiport CAN-to-USB and USB-CANmodul1/2
dwAMR_p:	Acceptance filter mask *(see section 2.3.5)*
dwACR_p:	Acceptance filter code
bMode_p:	Transmission mode of CAN-channel
bOCR_p:	Output-Control-Register

Return value:

Error code of the function.

USBCAN_SUCCESSFUL
USBCAN_ERR_MAXINSTANCES
USBCAN_ERR_ILLHANDLE
USBCAN_ERR_RESOURCE
USBCAN_ERR_BUSY
USBCAN_ERR_IOFAILED
USBCAN_ERRCMD_…

ResetCan

Syntax C#:

```
public byte USBcanServer.ResetCan(byte pbChannel_p)
```

Syntax Visual Basic:

```
Public Function USBcanServer.ResetCan( _
           ByVal pbChannel_p As Byte) As Byte
```

Usability:

HW_INIT, CAN_INIT, version 3.01 and higher

Description:

Resets a CAN-channel of a device. (see function *UcanResetCan()*). With GW-001 and GW-002 only CAN-channel 0 is available.

Parameter:

bChannel_p:	CAN-channel, to be reset. USBCAN_CHANNEL_CH0 for CAN- channel 0 USBCAN_CHANNEL_CH1 for CAN- channel 1

Return value:

Error code of the function.

USBCAN_SUCCESSFUL
USBCAN_ERR_MAXINSTANCES
USBCAN_ERR_ILLHANDLE
USBCAN_ERR_CANNOTINIT
USBCAN_ERR_BUSY
USBCAN_ERR_IOFAILED
USBCAN_ERRCMD_…

GetHardwareInfo

Syntax C#:

```
public byte USBcanServer.GetHardwareInfo(
     ref tUcanHardwareInfoEx pHwInfo_p,
     ref tUcanChannelInfo pCanInfoCh0_p,
     ref tUcanChannelInfo pCanInfoCh1_p);
```

Syntax Visual Basic:

```
Public Function USBcanServer.GetHardwareInfo( _
     ByRef pHwInfo_p As tUcanHardwareInfoEx, _
     ByRef pCanInfoCh0_p As tUcanChannelInfo, _
     ByRef pCanInfoCh1_p As tUcanChannelInfo) As Byte
```

Usability:

HW_INIT, CAN_INIT, version 3.01 and higher

Description:

Returns the extended hardware information of a device. With Multiport CAN-to-USB 3004006 and USB-CANmodul2 3204002/3204003 the information for both CAN-channels is returned separately.

Parameter:

pHwInfo_p:	Pointer to structure where the extended hardware information is to be stored. (*see function UcanGetHardwareInfoEx2*).
pCanInfoCh0_p:	Pointer to structure where the information of CAN-channel 0 is to be stored.
pCanInfoCh1_p:	Pointer to structure where the information of CAN-channel 1 is to be stored.

Return value:

Error code of the function.

USBCAN_SUCCESSFUL
USBCAN_ERR_MAXINSTANCES
USBCAN_ERR_ILLHANDLE
USBCAN_ERR_ILLPARAM

GetStatus

Syntax C#:

```
public byte USBcanServer.GetStatus(byte  bChannel_p,
     ref tStatusStruct pStatus_p);
```

Syntax Visual Basic:

```
Public Function USBcanServer.GetStatus( _
     ByVal pbChannel_p As Byte, _
     ByRef pStatus_p As tStatusStruct) As Byte
```

Usability:

HW_INIT, CAN_INIT, version 3.01 and higher

Description:

Returns the error status of a specific CAN-channel. Structure *tStatusStruct* is described with function *UcanGetStatus()*.

Parameter:

bChannel_p:	CAN-Channel, to be used. USBCAN_CHANNEL_CH0 for CAN-Channel 0 USBCAN_CHANNEL_CH1 for CAN-Channel 1
pStatus_p:	Error status of the device.

Return value:

Error code of the function.

USBCAN_SUCCESSFUL
USBCAN_ERR_MAXINSTANCES
USBCAN_ERR_ILLHANDLE
USBCAN_ERR_ILLPARAM

SetBaudrate

Syntax C#:

```
public byte USBcanServer.SetBaudrate(
     byte  bChannel_p = USBCAN_CHANNEL_CH0,
     short  wBTR_p = USBCAN_BAUD_1MBit,
     int  dwBaudrate_p = USBCAN_BAUDEX_USE_BTR01);
```

Syntax Visual Basic:

```
Public Function USBcanServer.SetBaudrate( _
     Optional ByVal bChannel_p As Byte = USBCAN_CHANNEL_CH0, _
     Optional ByVal wBTR_p As Short = USBCAN_BAUD_1MBit, _
     Optional ByVal dwBaudrate_p As Integer =
     USBCAN_BAUDEX_USE_BTR01) As Byte
```

Usability:

CAN_INIT, version 3.01 and higher

Description:

This function is used to configure the baud rate of specific CAN-channel of a device.

Parameter:

bChannel_p:	CAN-Channel, which is to be configured. USBCAN_CHANNEL_CH0 for CAN-Channel 0 USBCAN_CHANNEL_CH1 for CAN-Channel 1
wBTR_p:	Baud rate register BTR0 as high byte, Baud rate register BTR1 as low byte
dwBaudrate_p:	Baud rate register of Multiport CAN-to-USB 3004006 or USB-CANmodul1/2

Return value:

Error code of the function.

USBCAN_SUCCESSFUL
USBCAN_ERR_MAXINSTANCES
USBCAN_ERR_ILLHANDLE
USBCAN_ERR_CANNOTINIT
USBCAN_ERR_BUSY
USBCAN_ERR_IOFAILED
USBCAN_ERRCMD_…

SetAcceptance

Syntax C#:

```
public byte USBcanServer.SetAcceptance(
     byte  bChannel_p = USBCAN_CHANNEL_CH0,
     int  dwAMR_p = USBCAN_AMR_ALL,
     int  dwACR_p = USBCAN_ACR_ALL);
```

Syntax Visual Basic:

```
Public Function USBcanServer.SetAcceptance( _
     Optional ByVal bChannel_p As Byte = USBCAN_CHANNEL_CH0, _
     Optional ByVal dwAMR_p As Integer = USBCAN_AMR_ALL, _
     Optional ByVal dwACR_p As Integer = USBCAN_ACR_ALL)
     As Byte
```

Usability:

CAN_INIT, version 3.01 and higher

Description:

This function is used to change the acceptance filter values for a specific CAN-channel on a device.

Parameter:

bChannel_p:	CAN-Channel, which is to be configured. USBCAN_CHANNEL_CH0 for CAN-Channel 0 USBCAN_CHANNEL_CH1 for CAN-Channel 1
dwAMR_p:	Acceptance filter mask *(see section 2.3.5)*
dwACR_p:	Acceptance filter code

Return value:

Error code of the function.

USBCAN_SUCCESSFUL
USBCAN_ERR_MAXINSTANCES
USBCAN_ERR_ILLHANDLE
USBCAN_ERR_CANNOTINIT
USBCAN_ERR_BUSY
USBCAN_ERR_IOFAILED
USBCAN_ERRCMD_…

ReadCanMsg

Syntax C#:

```
public byte USBcanServer.ReadCanMsg(ref byte  pbChannel_p,
     ref tCanMsgStruct[] pCanMsg_p,
     ref int pdwCount_p = 0);
```

Syntax Visual Basic:

```
Public Function USBcanServer.ReadCanMsg(ByRef pbChannel_p As
     Byte,
     ByRef pCanMsgStruct_p() As tCanMsgStruct,
     Optional ByRef dwCount_p As Integer = 0) As Byte
```

Usability:

CAN_INIT, version 3.01 and higher

Description:

Reads one or more CAN-messages from the buffer of the specified CAN-channel.

Parameter:

pbChannel_p:	Pointer to a variable containing the CAN-Channel to read from. USBCAN_CHANNEL_CH0 for CAN-Channel 0 USBCAN_CHANNEL_CH1 for CAN-Channel 1 If USBCAN_CHANNEL_ANY is given, then this function writes the CAN-channel to this variable, where the message was read from.
pCanMsg_p:	Address to a CAN message structure. This address must not be NULL.
pdwCount_p:	Address to a variable that specifies the maximum number of CAN messages to be read. This function writes the actual number of CAN messages that were read from the device to this variable.

Structure *tCanMsgStruct* is described with function *UcanReadCanMsg()*.

Return value:

Error code of the function.

USBCAN_SUCCESSFUL
USBCAN_ERR_MAXINSTANCES
USBCAN_ERR_ILLHANDLE
USBCAN_ERR_CANNOTINIT
USBCAN_ERR_ILLPARAM
USBCAN_WARN_NODATA
USBCAN_WARN_SYS_RXOVERRUN
USBCAN_WARN_DLL_RXOVERRUN

WriteCanMsg

Syntax C#:

```
public byte USBcanServer.WriteCanMsg(byte bChannel_p,
     ref tCanMsgStruct[] pCanMsg_p
     ref int dwCount_p = 0);
```

Syntax Visual Basic:

```
Public Function USBcanServer.WriteCanMsg( _
     ByVal pbChannel_p As Byte, _
     ByRef pCanMsgStruct_p() As tCanMsgStruct,
     Optinal ByRef dwCount_p As Integer) As Byte
```

Usability:

CAN_INIT, version 3.01 and higher

Description:

Transmits one ore more CAN messages through the specified CAN-channel of the device.

Parameter:

bChannel_p:	CAN-Channel, which is to be used. USBCAN_CHANNEL_CH0 for CAN-Channel 0 USBCAN_CHANNEL_CH1 for CAN-Channel 1
pCanMsg_p:	Address to a CAN message structure. This address must not be NULL.
dwCount_p:	After return this variable holds the number of CAN messages which was successfully stored to the transmit buffer. This value may be less than the number of elements within the array of CAN messages if not all CAN messages could be stored to the transmit buffer. In that case the function returns the warning USBCAN_WARN_TX_LIMIT.

Structure *tCanMsgStruct* is described with function *UcanWriteCanMsg()*.

Return value:

Error code of the function.

USBCAN_SUCCESSFUL
USBCAN_ERR_MAXINSTANCES
USBCAN_ERR_ILLHANDLE
USBCAN_ERR_CANNOTINIT
USBCAN_ERR_ILLPARAM
USBCAN_ERR_DLL_TXFULL

Example for Visual Basic .NET:

```
' variable for return value
Dim bRet As Byte = 0
' array of tCanMsgStruct with length 1
Dim canMsgStruct(0) As UcanDotNET.USBcanServer.tCanMsgStruct

' initialize the first element with a new structure instance
canMsgStruct(0) = _
    UcanDotNET.USBcanServer.tCanMsgStruct.CreateInstance(&H123)

' fill message data with some value
canMsgStruct(0).m_bData(0) = &HAB
canMsgStruct(0).m_bData(1) = &HCD
canMsgStruct(0).m_bData(2) = &HEF
canMsgStruct(0).m_bData(3) = &H12
canMsgStruct(0).m_bData(4) = &H34
canMsgStruct(0).m_bData(5) = &H56
canMsgStruct(0).m_bData(6) = &H78
canMsgStruct(0).m_bData(7) = &H90

' send message
bRet = m_USBcan.WriteCanMsg( _
         UcanDotNET.USBcanServer.USBCAN_CHANNEL_CH0, _
         canMsgStruct)
' check return value
' ……
```

GetMsgCountInfo

Syntax C#:

```
public byte USBcanServer.GetMsgCount(
     byte bChannel_p,
     ref short wRecvdMsgCount_p, _
     ref short wSentMsgCount_p);
```

Syntax Visual Basic:

```
Public Function USBcanServer.GetMsgCount( _
     ByVal bChannel_p As Byte, _
     ByRef pwRecvdMsgCount_p As Short, _
     ByRef pwSentMsgCount_p As Short) As Byte
```

Usability:

CAN_INIT, version 3.01 and higher

Description:

Reads the message counters of the specified CAN-channel.

Parameter:

bChannel_p:	CAN-Channel, which is to be used. USBCAN_CHANNEL_CH0 for CAN-Channel 0 USBCAN_CHANNEL_CH1 for CAN-Channel 1
pwRecvdMsgCount_p:	pointer to variable for receive message counter.
pwSentMsgCount_p:	pointer to variable for transmit message counter.

Return value:

Error code of the function.

USBCAN_SUCCESSFUL
USBCAN_ERR_MAXINSTANCES
USBCAN_ERR_ILLHANDLE
USBCAN_ERR_CANNOTINIT
USBCAN_ERR_BUSY
USBCAN_ERR_IOFAILED
USBCAN_ERR_ILLCMD
USBCAN_WARN_NOTEQU

GetCanStatusMessage

Syntax C#:

```
public static String USBcanServer.GetCanStatusMessage(
     short wCanStatus_p);
```

Syntax Visual Basic:

```
Public Shared Function USBcanServer.GetCanStatusMessage( _
     ByVal wCanStatus_p As Short) As String
```

Usability:

CAN_INIT, version 3.01 and higher

Description:

Returns the specified status code as a string message.

Parameter:

wCanStatus_p:	CAN status code (*see USBCAN API function UcanGetStatus()*)

Return value:

String message

GetBaudrateMessage

Syntax C#:

```
public static String USBcanServer.GetBaudrateMessage(
     byte bBTR0_p, byte bBTR1_p);
```

Syntax Visual Basic:

```
Public Shared Function USBcanServer.GetBaudrateMessage( _
     ByVal bBTR0_p As Byte, ByVal bBTR1_p As Byte) As String
```

Usability:

CAN_INIT, version 3.01 and higher

Description:

Returns a string containing the BTR register values.

Parameter:

bBTR0_p:	Baud rate register 0
bBTR1_p:	Baud rate register 1

Return value:

String message

GetBaudrateExMessage

Syntax C#:

```
public static String USBcanServer.GetBaudrateExMessage(
     int dwBTR_p);
```

Syntax Visual Basic:

```
Public Shared Function USBcanServer.GetBaudrateExMessage( _
     ByVal dwBTR_p As Integer) As String
```

Usability:

CAN_INIT, version 3.01 and higher

Description:

Returns a string containing the specified baud rate.

Parameter:

dwBTR_p:	Baud rate as 32-bit value

Return value:

String message

2.4.2 Event of class USBcanServer

Class USBcanServer passes the callback events of the USBCAN32.DLL to the application as .NET events.

CanMsgReceivedEvent

Syntax C#:

```
public event USBcanServer.CanMsgReceivedEvent(
     byte bChannel_p);
```

Syntax Visual Basic:

```
Public Event USBcanServer.CanMsgReceivedEvent( _
     ByVal bChannel_p As Byte)
```

Description:

A new CAN-message was received.

Parameter:

bChannel_p:	CAN-Channel that received the message. USBCAN_CHANNEL_CH0 for CAN-Channel 0 USBCAN_CHANNEL_CH1 for CAN-Channel 1 USBCAN_CHANNEL_ANY for any Channel

InitHwEvent

Syntax C#:

```
public event USBcanServer.InitHwEvent();
```

Syntax Visual Basic:

```
Public Event USBcanServer.InitHwEvent()
```

Description:

Device was initialized.

InitCanEvent

Syntax C#:

```
public event USBcanServer.InitCanEvent(byte bChannel_p);
```

Syntax Visual Basic:

```
Public Event USBcanServer.InitCanEvent( _
     ByVal bChannel_p As Byte)
```

Description:

Specified CAN-channel was initialized.

Parameter:

bChannel_p:	CAN-Channel that was initialized. USBCAN_CHANNEL_CH0 for CAN-Channel 0 USBCAN_CHANNEL_CH1 for CAN-Channel 1

StatusEvent

Syntax C#:

```
public event USBcanServer.StatusEvent(byte bChannel_p);
```

Syntax Visual Basic:

```
Public Event USBcanServer.StatusEvent(ByVal bChannel_p As Byte)
```

Description:

Error status for the specified CAN-Channel has changed.

Parameter:

bChannel_p:	CAN-Channel, which error status has been changed. USBCAN_CHANNEL_CH0 for CAN-Channel 0 USBCAN_CHANNEL_CH1 for CAN-Channel 1

Example of an event handler for the status event in Visual Basic .NET:

```
Private Sub USBcan_Status(ByVal bChannel_p As Byte) _
            Handles m_USBcan.StatusEvent

    ' status of USB-CANmodul changed
    Dim status As UcanDotNET.USBcanServer.tStatusStruct
    Dim bRet As Byte = 0

    bRet = m_USBcan.GetStatus(bChannel_p, status)
    If bRet = UcanDotNET.USBcanServer.USBCAN_SUCCESSFUL Then
        Console.WriteLine("CAN status of channel "
                          + bChannel_p.ToString() + ": " +
                          status.m_wCanStatus.ToString("X4"))
    Else
        Console.WriteLine("Error while reading status: "
                          + bRet.ToString("X2"))
    End If

End Sub
```

DeinitCanEvent

Syntax C#:

```
public event USBcanServer.DeinitCanEvent(byte bChannel_p);
```

Syntax Visual Basic:

```
Public Event USBcanServer.DeinitCanEvent( _
    ByVal bChannel_p As Byte)
```

Description:

The CAN-channel was shut down..

Parameter:

bChannel_p:	CAN-Channel, which status has been changed. USBCAN_CHANNEL_CH0 for CAN-Channel 0 USBCAN_CHANNEL_CH1 for CAN-Channel 1

DeinitHwEvent

Syntax C#:

```
public event USBcanServer.DeinitHwEvent()
```

Syntax Visual Basic:

```
Public Event USBcanServer.DeinitHwEvent()
```

Description:

The device was shut down.

ConnectEvent

Syntax C#:

```
public static event USBcanServer.ConnectEvent();
```

Syntax Visual Basic:

```
Public Shared Event USBcanServer.ConnectEvent()
```

Description:

A new device was connected to the USB-port.

DisconnectEvent

Syntax C#:

```
public static event USBcanServer.DisconnectEvent();
```

Syntax Visual Basic:

```
Public Shared Event USBcanServer.DisconnectEvent()
```

Description:

A previously shut down device was disconnected from the USB-port.

FatalDisconnectEvent

Syntax C#:

```
public event USBcanServer.FatalDisconnectEvent();
```

Syntax Visual Basic:

```
Public Shared Event USBcanServer.FatalDisconnectEvent()
```

Description:

A device was disconnected from the USB-port without prior shutdown.

3 Software support for Linux OS

The software package with order number SO-1068 contains a driver *for the Linux Kernel 2.6 in source code for x86-compatible CPUs, a* library with all API functions as well as a demo application. This manual describes the driver since version 2.00.

3.1 Installation of the driver under Linux

There are two conditions for installing the driver under Linux OS:

1) You need to be logged in as “super user”.
2) Kernel sources need to be installed for compiling the kernel driver.

Unzip the archive in a directory of your choice and run the shell script **initdriver.sh**. You will be asked whether you want to recompile the kernel driver. Answer with “y” for yes because the binary is not a content of the archive. The reason for is that Linux OS is able to run under several different CPUs.

Furthermore, the kernel driver for the USB-CANmodul has only been tested for some Linux Kernel Versions (currently 2.6.13, 2.6.16 and 2.6.18 using SUSE distribution, 2.6.24 using Ubuntu distribution, 2.6.25 at 64 bit processor using SUSE distribution). There is the possibility that the kernel driver cannot be compiled without errors if any changes were made in newer Linux Kernel Versions. In this case, please make your own changes to the sources of the USB-CANmodul device driver.

After successful compiling of the driver you will be asked whether the debug mode of the kernel driver should be activated. Within this mode, the driver will redirect all debug outputs to the function *printk()*. Therefore, you will find the debug outputs in */var/log/messages*. For now, answer this question with “n” for no.

Since driver version V2.02 the loader driver also automatically updates the firmware of sysWORXX USB-CANmoduls.

3.2 API functions under Linux

In driver version V2.00 the API functions under Linux are located in two static libraries, **Host.Lin/library/libusbcanr.a** (for release mode) and **Host.Lin/library/libusbcand.a** (for debug mode) of the archive. These functions correspond to those under Windows (*see section 2.3*). However, not all functions are available under Linux. *Table 19* lists all available functions under Linux.

Since driver version V2.02 two dynamic libraries are delivered instead of static libraries. With shell script **initdriver.sh,** both libraries are copied to the path /usr/local/lib and symbolic links are created. **libusbcan.so** is used for the release mode and **libusbcand.so** for the debug mode.

Functions available	**Functions not available**
UcanGetVersionEx()	*UcanGetVersion() (obsolete)*
UcanInitHardware()	*UcanGetFwVersion()*
UcanInitCan()	*UcanInitHwConnectControl()*
UcanResetCan()	*UcanInitHwConnectControlEx()*
UcanGetStatus()	*UcanDeinitHwConnectControl()*
UcanSetBaudrate()	*UcanInitHardwareEx()*
UcanSetAcceptance()	*UcanGetModuleTime()*
UcanReadCanMsg()	*UcanGetHardwareInfo()*
UcanWriteCanMsg()	*UcanGetHardwareInfoEx2()*
UcanDeinitCan()	*UcanInitCanEx()*
UcanDeinitHardware()	*UcanInitCanEx2()*
	UcanSetBaudrateEx()
	UcanSetAcceptanceEx()
	UcanResetCanEx()
	UcanReadCanMsgEx()
	UcanWriteCanMsgEx()
	UcanGetStatusEx()
	UcanGetMsgCountInfo()
	UcanGetMsgCountInfoEx()
	UcanConfigUserPort()
	UcanWriteUserPort()
	UcanReadUserPort()
	UcanReadUserPortEx()
	UcanWriteCanPort()
	UcanWriteCanPortEx()
	UcanReadCanPort()
	UcanReadCanPortEx()
	UcanDefineCyclicCanMsg()
	UcanReadCyclicCanMsg()
	UcanEnableCyclicCanMsg()
	UcanGetMsgPending()
	UcanGetCanErrorCounter()
	UcanDeinitCanEx()

Table 19: Available functions under Linux OS

The following API functions differ from those under Windows:

```
UCANRET UcanInitHardware (tUcanHandle* pUcanHandle_p,
     BYTE                    bDeviceNr_p,
     void*                   pReserved_p);
```

Function *UcanInitHardware()* does not support a callback function. Thus the third parameter is marked as “reserved”.

```
UCANRET UcanSetDeviceNr (tUcanHandle UcanHandle_p,
     BYTE                    bDeviceNr_p);
```

Function *UcanSetDeviceNr()* can be used to configure a new device number for a USB-CANmodul.

```
UCANRET UcanSetDebugMode (DWORD dwDbgLevel_p,
     void*                   pReserved_p,
     DWORD                   dwReserved_p);
```

Function *UcanSetDebugMode()* changes the debug LOG-Level for debug outputs of the USBCAN-library. 0xE0000B03 is the default value for the library libusbcand.a. Refer to section 3.3 and read the function description in section 2.3.2 for more information.

3.3 Logging debug information

If there are problems with the driver it would be helpful to send us a debug log file with the debug information of the USBCAN-library. Under Linux OS you need to use libusbcand.a instead of libusbcanr.a in your application to activate the debug outputs. These outputs are redirected to the function printf() since library version 2.00. Consequently, the debug information will appear at the standard output. Please copy these outputs to a file and send it to our support address.

With the USBCAN-API function *UcanSetDebugMode()* you can specify which debug information should be printed out.

Sometimes we cannot locate the problem from the debug log file. In this case, it would be helpful if you also send us the debug outputs from kernel driver. Therefore, you should answer the question whether you want to activate the debug outputs of the kernel driver in shell script initdriver.sh with “y” for yes. If you do not use this shell script for loading the kernel drivers you need to set the parameter “debug” to 1:

```
Insmod  usbcanlx.ko  debug=1

Insmod  usbcan.ko  debug=1
```

The debug outputs of the kernel driver are redirected to the function printk() if the parameter debug is set to 1. Therewith, the debug information will appear in file */var/log/messages*.

4 Software support for Windows CE OS

The software package with order number SO-1091 contains a driver for Windows CE, a USBCAN-library as DLL and a demo application in source code for Microsoft eMbedded Visual C++ 4.0. The following Windows CE versions and CPU types are tested with the driver:

Windows CE version	CPU type	Tested at CPU
5.0	ARMV4I	Intel PXA255 Intel PXA270
6.0	X86	Intel Atom

Table 20: tested Windows CE versions and CPU types

On request we can build a driver for other CPU types too. Please contact our support department for this.

4.1 Installation of the driver under Windows CE

After unzipping the archive, you will find the driver in subfolder *Driver\XXX* whereas XXX marks the CPU type. For example, the driver for CPU type ARMV4I is located in subfolder *Driver\ARMV4I*. The name of the driver file in each case is **UsbCanDrv.dll**.

This driver in its current version supports the following USB-CANmoduls:

> GW-002, USB-CANmodul1 (3204000 and 3204001) and all derivates of USB-CANmodul2 (3204001, 3204003, …)

Prior to connecting the USB-CANmodul to your Windows CE device, copy the driver to the subfolder *\Windows*. Now connect the USB-CANmodul with your Windows CE device. If the USB-CANmodul is connected for the first time, a window “Unidentified USB device” appears. Type in the name of the driver: **usbcandrv** (no capitalization rules). Now you can access the USB-CANmodul with an application. You can use our **ConsoleDemo.exe** for example. It sends 100 CAN messages with 1 MBit/sec and prints out received CAN messages in the mean time.

4.2 API functions under Windows CE

UsbCanCE.dll contains the API functions for the USB-CANmodul. This USBCAN-library is located in subfolder *Lib\XXX* whereas XXX marks the CPU type. This DLL contains all functions that are also available in USBCAN32.DLL under Windows 2000/XP/Vista – except for three functions:

UcanInitHwConnectControl()

UcanInitHwConnectControlEx()

UcanDeinitHwConnectControl()

Please refer to *section 2.3* for the description of all available functions.

There are some more differences between UsbCanCE.dll an USBCAN32.dll:

- Under Windows CE up to 9 USB-CANmoduls can only be used simultaneously at one Windows CE device.
- Function *UcanGetVersionEx()* only can be run with the parameters kVerTypeUserDll, kVerTypeUserLib and kVerTypeSysDrv.
- The call of *UcanDeinitCan()* or *UcanDeinitCanEx()* can take up to two seconds until return from this function.

4.3 Logging debug information

If there are problems with the driver it would be helpful if you send us a debug log file with debug information of the USBCAN-library. Using the current version Windows CE, it is momentarily only possible to activate this feature by calling the function *UcanSetDebugMode()* in your application. Please refer to *section 2.3.2* for more information.

5 Index

Document:	**USB-CANmodul**
Document number:	, Edition February 2010

How would you improve this manual?

Did you find any mistakes in this manual? page

Submitted by:

Customer number:	
Name:	
Company:	
Address:	

Return to: SYS TEC electronic GmbH
August-Bebel-Str. 29
D-07973 Greiz
GERMANY
Fax : +49 (0) 36 61 / 62 79 99