SYS TEC ELECTRONIC

USB-CANmodul

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

System Manual

Ausgabe Oktober 2008

Systemhaus für verteilte Automatisierung

Im Buch verwendete Bezeichnungen für Erzeugnisse, die zugleich ein eingetragenes Warenzeichen darstellen, wurden nicht besonders gekennzeichnet. Das Fehlen der © Markierung ist demzufolge nicht gleichbedeutend mit der Tatsache, dass die Bezeichnung als freier Warenname gilt. Ebenso wenig kann anhand der verwendeten Bezeichnung auf eventuell vorliegende Patente oder einen Gebrauchsmusterschutz geschlossen werden.
Die Informationen in diesem Handbuch wurden sorgfältig überprüft und können als zutreffend angenommen werden. Dennoch sei ausdrücklich darauf verwiesen, dass die Firma SYS TEC electronic GmbH weder eine Garantie noch die juristische Verantwortung oder irgendeine Haftung für Folgeschäden übernimmt, die auf den Gebrauch oder den Inhalt dieses Handbuches zurückzuführen sind. Die in diesem Handbuch enthaltenen Angaben können ohne vorherige Ankündigung geändert werden. Die Firma SYS TEC electronic GmbH geht damit keinerlei Verpflichtungen ein.
Ferner sei ausdrücklich darauf verwiesen, dass SYS TEC electronic GmbH weder eine Garantie noch die juristische Verantwortung oder irgendeine Haftung für Folgeschäden übernimmt, die auf falschen Gebrauch oder falschen Einsatz der Hard- bzw. Software zurückzuführen sind. Ebenso können ohne vorherige Ankündigung Layout oder Design der Hardware geändert werden. SYS TEC electronic GmbH geht damit keinerlei Verpflichtungen ein.

Informieren Sie sich:

	EUROPA	NORD AMERIKA
Adresse:	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
Angebots Hotline:	+49 (3661) 6279-0 info@systec-electronic.com	+1 (800) 278-9913 order@phytec.com
Technische Hotline:	+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 Seite:	http://www.systec-electronic.com	http://www.phytec.com

21. Ausgabe Oktober 2008

Bild- und Tabellenverzeichnis

Einleitung

Das USB-CANmodul ist eine Baugruppe zur Kopplung von CAN-Feldbussystemen mit dem PC. Dabei kommt die USB-Schnittstelle (Universal Serial Bus) zum Einsatz. Die USB-Schnittstelle ist mittlerweile in jedem neuen PC vorhanden und wird von den Betriebssystemen MS-Windows 95 (ab OSR 2.1), 98, ME, 2000 und XP und neuere Versionen sowie Linux unterstützt.

USB dient zur Verbindung von verschiedenen peripheren Geräten mit dem PC. Die einfache Handhabung erlaubt es, Geräte während der Laufzeit des Rechners anzuschließen (Hot Plug & Play) und sofort damit zu arbeiten. Das Aufschrauben des PC oder Konfigurationsprobleme am Gerät entfallen. Mit USB ist lediglich das Verbindungskabel in den PC oder in einen Hub zu stecken. Der PC als Host verwaltet und bedient alle angeschlossenen Functions (USB-Geräte). Es können mehrere Functions angeschlossen werden. Die USB-Schnittstelle ermöglicht es, Daten mit einer Rate bis zu 12 MBit/s zu übertragen. Mit einem einheitlichen Steckverbinder für alle Gerätetypen ist das System ausgesprochen benutzerfreundlich.

Stellt man die Verbindung zwischen dem USB-CANmodul und dem PC her, liest der PC die Konfiguration ein und lädt den dazugehörigen Gerätetreiber automatisch. Damit ist die Verbindung vom PC zum CAN-Bus hergestellt. Alle CAN-Nachrichten werden über den USB-Bus transparent getunnelt. Dabei wird die max. CAN-Baudrate von 1MBit/s unterstützt. Die CAN-Nachrichten werden vom aktiven USB-CANmodul zwischengepuffert. Das gilt für gesendete und empfangene CAN-Nachrichten. Es werden CAN-Frames nach der CAN-Spezifikation 2.0A und 2.0B (11- und 29-Bit Identifier) unterstützt. Die CAN-Bus-Anschaltung ist nach dem CiA-Standard DS 102 aufgebaut und optional galvanisch entkoppelt.

Zum USB-CANmodul werden Treiber für MS Windows 2000, XP und Vista sowie Linux bereitgestellt. Ein Konfigurationstool unter Windows ermöglicht die Zuordnung mehrerer Geräte am USB-Bus. Das erfolgt mit Hilfe von Gerätenummern, die der Anwender vergibt. Die Gerätenummer wird in einem EEPROM abgelegt. Mit einem

Windows-Tool kann leicht der Datentransfer von CAN-Nachrichten überprüft werden. Die Funktionen für den Datenaustausch mit einer Applikation stehen über eine DLL (Dynamic Linked Library) zur Verfügung. Das mitgelieferte Demoprogramm zeigt die einfache Handhabung der DLL-API-Funktionen.

Dieses Manual bezieht sich auf folgende USB-CANmodule:

Artickel-nummer	**Besonderheiten**
GW-001	Veraltet, wird jedoch von der Software weiterhin unterstützt. - Galvanische Entkopplung wird über Steck-Jumper konfiguriert. - Keine Software-Unterstützung seit Windows Treiber Version V4.00.
GW-002	Veraltet, wird jedoch von der Software weiterhin unterstützt. - kompakteres Gehäuse: 102x54x30 (LxBxH in mm), Schutzgrad IP40, auch für DIN-Schienenmontage - die galvanische Trennung ist Bestelloption, dadurch entfällt das Öffnen des Gehäuses und die Notwendigkeit der Spannungsversorgung über den CAN-Bus - CAN-Masse (CAN-GND) und CAN-Schirm (CAN-SHLD) sind im USB-CANmodul nicht verbunden. - optional verschiedene CAN-Transceiver verfügbar: lowspeed, single-wire - externe Spannungsversorgung bis zu 30V der CAN-Schnittstelle in Abhängigkeit des CAN-Transceivers möglich - kundenspezifisch erweiterbar über einen freien 8 Bit Port des Microcontrollers - weitere CAN-Transceiver über Steckplatz auf der Platine anpassbar
3004006	Multiport CAN-to-USB mit 16 CAN Kanälen - 19“ Einschub - beinhaltet 8 logische Geräte mit jeweils 2 CAN-Kanälen - intelligenter und schneller 32 Bit Microcontroller - externe Spannungversorgung mit Kaltgerätestecker (230 VAC mit 500mA abgesichert) für das gesamte Gerät - galvanisch Trennung der CAN-Kanäle
3204000, 3204001	USB-CANmodul1 - kleines Gehäuse: 78x45x18 (LxBxH in mm) - 1 CAN-Kanal - schneller 32 Bit Microcontroller - Spannungsversorgung über USB-Kabel, Stromverbrauch max. 110mA - Highspeed CAN-Transceiver 82C251 - galvanisch getrennt (nur bei 3204001)

Artickel-nummer	**Besonderheiten**
3204002, 3204003, 3204007, 3204008, 3204009, 3204011	USB-CANmodul2 - 2 unabhängige CAN-Kanäle - schneller 32 Bit Microcontroller - Spannungsversorgung über USB-Kabel - Highspeed CAN-Transceiver 82C251 oder Lowspeed CAN-Transceiver TJA1054 bzw. Single-Wire CAN-Transceiver AU5790 - galvanisch getrennt (nur bei 3204003) - kundenspezifisch erweiterbar über einen freien 8 Bit Port des Microcontrollers (nur bei 3204007)
3404000	USB-CANmodul8 - mit 8 CAN Kanälen - im Tischgehäuse - beinhaltet 4 logische Geräte mit jeweils 2 CAN-Kanälen - intelligenter und schneller 32 Bit Microcontroller - externe Spannungversorgung mit Kaltgerätestecker (230 VAC mit 500mA abgesichert) für das gesamte Gerät - galvanisch Trennung der CAN-Kanäle
3404001	USB-CANmodul16 - mit 16 CAN Kanälen - im Tischgehäuse - beinhaltet 8 logische Geräte mit jeweils 2 CAN-Kanälen - intelligenter und schneller 32 Bit Microcontroller - externe Spannungversorgung mit Kaltgerätestecker (230 VAC mit 500mA abgesichert) für das gesamte Gerät - galvanisch Trennung der CAN-Kanäle
	-

Hinweise auf Hard- und Software-Änderungen

In diesem Kapitel finden Sie Hinweise auf neue Funktionen in Hard- bzw. Software des USB-CANmoduls.

Ab der Version 2.15 wird im PCANView die Auswahl beliebiger Baudraten unterstützt. Die Funktionen *UcanWriteCanPort()* und *UcanReadCanPort()* zur Kontrolle der lowspeed CAN-Transceiver wurden eingebaut.

Software Version 2.16:
- Bei Neuinstallation erscheint das USB-CANmodul im Gerätemanager nicht mehr unter dem Eintrag **„USB Controller**“, sondern unter dem Eintrag **„USB-CAN-Hardware**“.
- Auslesen der Seriennummer und des Modus des CAN-Controllers über die Funktion *UcanGetHardwarInfo()*
- Funktion *UcanGetVersionEx()* mit erweiterter Versionsabfrage
- Funktion *UcanInitCanEx()* zur Erweiterung der Konfiguration des SJA1000, z.B. listen only - Modus
- *UcanConfigUserPort()*, *UcanWriteUserPort()* und *UcanReadUserPort()* zur Bedienung des 8-bit User Ports

Software Version 2.17:
- Es wird nur noch die Installation für Windows2000/XP unterstützt.
- Die Installation und der Betrieb unter Windows98/ME ist möglich, es wird jedoch keine Garantie übernommen.
- Die Meldungen „bus off“, „error passiv“ und „warning limit“ wurden bisher nicht an die Applikation gemeldet.
- Der CAN Status wird von *UcanGetStatus()* nicht mehr automatisch nach dem Lesen, sondern erst nach Aufruf von *UcanResetCan()* gelöscht.

Software Version 2.18:
- Der Gerätetreiber USBCAN.SYS unterstützt auch das Power Management. Wird der Rechner aus dem Stand-By Modus wieder aktiviert, dann wird auch der Gerätetreiber neu geladen.
- Die Genauigkeit des Zeitstempels für Empfangsnachrichten aus der Struktur *tCanMsgStruct* wurde verbessert.

- Die CAN Statusmeldungen USBCAN_CANERR_BUSLIGHT und USBCAN_CANERR_BUSHEAVY werden automatisch gelöscht, sobald der Fehlerzähler im CAN Controller den entsprechenden Wert unterschreitet.
- Das Tool UCAN Config wurde durch das Tool USB-CANmodul Control in der Systemsteuerung ersetzt.
- Das Tool PCANView beantwortete in früheren Versionen die RTR-Frames von 29-Bit CAN Nachrichten nicht automatisch. Dieser Fehler ist in der Version 2.0.4 Build 043 des PCANView behoben.
- Funktion *UcanGetFwVersion()* zur Versionsabfrage der Modulsoftware neu implementiert
- neues Demoprojekt für Microsoft Visual Studio C/C++ 6.0
- Softwareunterstützung für Borland Delphi als Bibliothek und Demoprojekt

Software Version 2.19:
- Ein Fehler wurde aus der USBCAN32.DLL beseitigt. Die ConnectControl-Callback-Funktion wurde in Version 2.18 nicht gerufen, sobald ein USB-CANmodul vom PC getrennt oder angesteckt wurde.
- neu: Softwareunterstützung für LabView inklusive eines Demos

Software Version 2.20:
- Ein Fehler wurde in der USBCAN32.DLL beseitigt. Nachdem die Funktion *UcanDeinitHwConnectControl()* gerufen wurde, konnte mit *UcanInitHwConnectControll()* keine neue Callback-Funktion angemeldet werden.
- Wenn die Funktion *UcanResetCan()* gerufen wurde, während sich noch CAN Nachrichten im Empfangspuffer befanden, konnte es vorkommen, dass danach ältere CAN Nachrichten noch mal empfangen wurden.
- Die Datei USBCAN32.DLD wurde entfernt. Um Debug Informationen zu erzeugen gibt es in „USB-CANmodul Control“ in der Systemsteuerung das Menü „Debug“.

Software Version 3.00:
- Die Hardware Connect Control Callback Funktion wurde unter Umständen mehrmals gerufen, obwohl nur ein USB-CANmodul mit dem PC verbunden oder abgezogen wurde.
- Neue API Funktionen wurden in die USBCAN32.DLL integriert, die die Arbeit mit dem Multiport CAN-to-USB 3004006 (mit 2 CAN Kanälen) unterstützt.
- Erweiterung der Treiber für den Anschluss von bis zu 64 USB-CANmodulen an einen PC.

Software Version 3.01:
- Fehler beseitigt: Der Funktionsaufruf *UcanDeinitCan()* konnte unter Umständen eine Zugriffsverletzung auslösen.
- Fehler beseitigt: Das USB-CANmodul GW-002 sendet nach einen Busoff nicht korrekt CAN-Nachrichten auf den CAN-Bus, obwohl die Funktion *UcanResetCan()* gerufen wurde.
- Fehler beseitigt: Bei Verwendung von mehreren USB-CANmodulen an einem PC über mehrere Applikationsinstanzen konnte es vorkommen, dass eine der Applikationen den Fehlercode 0x06 (illegal handle) meldete, obwohl die Initialisierung erfolgreich war.
- Fehler beseitigt: Wurde mit dem Multiport CAN-to-USB eine CAN2.0A Nachricht empfangen, dann wurde bei der nächsten CAN2.0B Nachricht die CAN-ID falsch an den PC übergeben.
- Bei GW-002 und Multiport CAN-to-USB werden gesendete CAN-Nachrichten als Empfangsnachricht an den PC zurückgegeben. Diese Nachricht ist als Echo gekennzeichnet (Sendeecho).
- Das Multiport CAN-to-USB unterstützt auch 10kBit/s.
- Das Multiport CAN-to-USB aktiviert die Traffic-LED nur dann, wenn der jeweilige CAN-Kanal initialisiert wurde.
- Funktion *UcanGetModuleTime()* in die USBCAN32.DLL implementiert.

Software Version 3.02
- Fehler beseitigt: Erstinstallation unter Windows 2000 funktionierte bei der Version 3.01 nicht.
- Fehler beseitigt: Die Funktion *UcanReadCanMsgEx()* gabt den Fehlercode USBCAN_ERR_CANNOTINIT beim Multiport CAN-to-USB 3004006 zurück, wenn nur der zweite CAN-Kanal

initialisiert wurde und diese Funktion mit dem Parameter *bChannel_p = USBCAN_CHANNEL_ANY* aufgerufen wurde.
- Softwareänderung: Wenn die Funktion *UcanInitCanEx()* bzw. *UcanInitCanEx2()* mit dem CAN-Modus kUcanModeNormal aufgerufen wurde, dann gab die Funktion *UcanReadCanMsg()* bzw. *UcanReadCanMsgEx()* den Returncode USBCAN_WARN_NODATA zurück, obwohl noch CAN-Nachrichten im Puffer enthalten waren. Ursache waren die Sendeechos, die innerhalb der DLL immer verarbeitet werden. Die USBCAN32.DLL wurde so geändert, dass sie die Sendeechos beim CAN-Modus kUcanModeNormal überliest und die nächste empfangene CAN-Nachricht zurückgibt.

Software Version 3.03
- Softwareänderung: Die USBCAN32.DLL hat zwei neue Empfangspuffer bekommen. Jeweils einen pro CAN-Kanal. Dadurch ist es möglich mit der Funktion *UcanReadCanMsgEx()* genau einen CAN-Kanal zu lesen, ohne dass CAN-Nachrichten des anderen CAN-Kanals diese blockieren.
- Softwareänderung: Der Funktion *UcanResetCanEx()* kann als Parameter übergeben werden, was zurückgesetzt werden soll, und was nicht zurückgesetzt werden soll.

Software Version 3.04
- Fehler beseitigt: Einige CAN-Nachrichten wurden vom Multiport CAN-to-USB zweifach gesendet, wenn die Senderate zu groß wurde.
- Fehler beseitigt: Es wurden einige zu sendende CAN-Nachrichten im Multiport CAN-to-USB gelöscht, wenn die Funktion *UcanInitCanEx2()* für den zweiten CAN-Kanal gerufen wurde.
- Fehler beseitigt: Das Löschen der Firmware-internen CAN-Nachrichten Puffer durch die Funktion *UcanInitCanEx2()* wird nicht durchgeführt, wenn einer der CAN-Kanäle bereits initialisiert wurde.
- Softwareänderung: Das Forcen eines Firmware-Updates für das Multiport CAN-to-USB (und folgende Hardware) ist möglich.
- Softwareänderung: Nach dem Löschen der Treiber für das USB-CANmodul aus dem Windows Betriebssystem muss das Betriebssystem neu gestartet werden.

Software Version 3.05
- Unterstützung für USB-CANmodul1 3204000/3204001 und USB-CANmodul2 3204002/3204003 hinzugefügt.
- Softwareänderung: Der Hardware-Typ wird über die Struktur tUcanHardwareInfoEx mit zurückgegeben.
- Softwareänderung: Die Anzahl der CAN-Nachrichten im Empfangs- und Sendepuffer der USBCAN32.DLL kann konfiguriert werden. Dazu gibt es in der Struktur tUcanInitCanParam die zwei neuen Parameter *m_wNrOfRxBufferEntries* und *m_wNrOfTxBufferEntries*.
- In *Tabelle 6* war die Pinbelegung der Porterweiterung falsch beschrieben Pin 9 und 10 waren vertauscht.

Software Version 3.06
- Softwareänderung: Unterstützung für das automatische Senden von CAN-Nachrichten hinzugefügt
- Softwareänderung: Funktion *UcanGetMsgPending()* hinzugefügt zum Auslesen der aktuellen Pufferzähler in den einzelnen Software-Schichten.
- Softwareänderung: Funktion *UcanGetCanErrorCounter()* hinzugefügt zum Auslesen der aktuellen Fehlerzähler des CAN-Controllers.
- Softwareänderung: Die Funktion *UcanWriteCanMsgEx()* gibt die Warnung USBCAN_WARN_TXLIMIT zurück, wenn sie gerufen wurde, um mehr als eine CAN-Nachricht zu senden, wobei nicht alle in den Sendepuffer passten.
- Softwareänderung: Die Funktion *UcanGetVersionEx()* kann auch die Datei-Versionen der Windows-Kernel-Treiber, des Tools USB-CANmodul Control und der Firmware-Loader zurückgeben.
- Softwareänderung: Schreibfehler in der Konstanten USBCAN_ERR_DISCONECT beseitigt. Richtige Schreibweise ist USBCAN_ERR_DISCONNECT, wobei die fehlerhafte Konstante aus Kompatibilitätsgründen weiterhin definiert bleibt.
- Softwareänderung: Einige Verbesserungen in der Darstellung im Tool USB-CANmodul Control wurden durchgeführt. Das Tool zeigt auch die Module an, die zurzeit von einer Applikation verwendet werden (Zeilen sind grau dargestellt). Die Debug-

Logdatei kann separat für das Tool USB-CANmodul Control zugeschaltet werden (bei der Erstinstallation ausgeschaltet).
- Fehler beseitigt: Die Funktion *UcanResetCanEx()* konnte nicht mit dem Parameter USBCAN_RESET_ONLY_STATUS gerufen werden, wenn sich die State-Machine im Zustand HW_INIT befand.
- Fehler beseitigt: Die Timer-Auflösung der neuen sysWORXX-Module wurde korrigiert. Beim Aufruf der Funktion *UcanReadCanMsg()* bzw. *UcanReadCanMsgEx()* wich der Zeitstempel etwas vom realen Wert ab.
- Softwareänderung: Bei den API-Funktionen der USBCAN32.DLL, die einen Fehlercode nach *Kapitel 2.3.3* zurückgeben, wurde der Rückgabetyp von BYTE auf UCANRET geändert. Diese Änderung wird jedoch keinen Einfluss auf bereits bestehende Applikationen haben, da der Typ UCANRET ebenso ein „unsigned char“ ist, wie der Typ BYTE.

Software Version 3.07
- Fehler beseitigt: Das automatische zyklische Senden frierte die Firmware in den sysWORXX USB-CANmodulen ein.
- Softwareänderung: Mutex in USBCAN32.DLL einefügt, damit die DLL unter Windows XP von verschiedenen Windows-Benutzern verwendet werden kann.
- Fehler beseitigt: *UcanWriteCanMsgEx()* gab auch den Wert USBCAN_WARN_TYLIMIT zurück, wenn nur eine einzige CAN-Nachricht gesendet werden sollte.

Software Version 3.08
- Softwareänderung: Erste Beta-Release des neuen Netzwerktreibers. Damit ist es möglich ein logisches USB-CANmodul der sysWORXX-Serie von mehreren Applikationen zu verwenden.
- Softwareänderung: Einige Verbesserungen im Tool USB-CANmodul Control: Anzeige der Verwendung des Netzwerktreibers, Update über das Internet, Module werden sortiert dargestellt.
- Fehler beseitigt: *UcanResetCanEx()* löschte auch dann die CAN-Nachrichten aus dem Firmware-Puffer, wenn dies explizit über die Reset-Maske ausgeklammert wurde.

Software Version 3.09
- Fehler beseitigt: Die rote Status-LED an den sysWORXX USB-CANmodulen wurde nach dem Löschen eines Fehlerzustandes nicht korrekt zurückgesetzt.
- Fehler beseitigt: Das Sendeecho wurde bei mehrkanaligen USB-CANmodulen unter Umständen mit UcanReadCanMsgEx() nicht zurückgegeben.
- Fehler beseitigt: Vom Kernel Treiber unter Windows 2000/XP wurde unter Umständen ein Overrun des Empfangspuffers gemeldet, obwohl dies nicht wirklich der Fall war.
- Neu: Kernel Treiber für Windows Vista verfügbar (jedoch nur die Standard-Variante – kein Netzwerktreiber!).

Software Version 3.10
- Fehler beseitigt: Wenn der Windows-Anwender keine Administrator-Rechte hatte, dann wurden keine Änderungen in der Registry angenommen (z.B. Einstellungen des LogLevels).
- Neu: Bei den neuen sysWORXX USB-CANmodulen werden über AMR und ACR auch die Datenbytes 0 und 1 sowie RTR-Frames gefiltert, so wie es auch mit den Modulen GW-001 und GW-002 der Fall war (*siehe Kapitel 2.3.5*).
- Neu: Die Funktion *UcanSetTxTimeout()* setzt bei sysWORXX USB-CANmodulen ein Timeout-Wert. Wenn eine zu sendende CAN-Nachricht nicht gesendet werden kann (z.B. weil keine CAN-Gegenstelle vorhanden ist), dann werden nach diesem Timeout alle nachfolgenden Sendenachrichten automatisch aus dem Sendepuffer gelöscht, so dass das Senden über einen zweiten Kanal nicht blockiert wird. Zusätzlich erhält die Applikation den Fehlerstatus USBCAN_CANERR_TXMSGLOST zurück. Ein Timeout-Wert von 0 schaltet dieses Feature aus (dies ist die Voreinstellung).
- Neu: Im DemoGW006 wird gezeigt, wie die USBCAN32.DLL zur Laufzeit geladen werden kann, indem man die Win32-Funktionen *LoadLibrary()* und *GetProcAddress()* verwendet.

Software Version 3.11
- Neu: Treiber für Windows CE 5.0 verfügbar. Dieser basiert auf den CPU-Typ ARMV4I und wurde auf dem Intel PXA255 und

PXA270 getestet. Andere CPUs können auf Anfrage portiert werden.
- Neu: Linux Treiber Version 2.02 r3 veröffentlicht. Die API Funktionen werden in einer Shared Library angeboten. Der Kernel Treiber unterstützt nun auch einen Firmware-Update für das USB-CANmodul1.

Software Version 4.00
- Fehler beseitigt: Der Aufruf der Funktion *UcanResetCanEx()* mit dem Flag-Parameter 0 hatte beim USB-CANmodul1 die Auswirkung, dass keine CAN-Nachrichten mehr empfangen werden konnten.
- Fehler beseitigt: Wurde die Funktion *UcanResetCanEx()* zum Löschen der Empfangspuffer gerufen, dann wurden für USB-CANmodul1 und GW-002 nicht alle CAN-Nachrichten gelöscht.
- Fehler beseitigt: Wenn die Funktion *UcanWriteCanMsgEx()* einen Fehlercode zurückgab, dann wurde dennoch die Variable auf 0 gesetzt, die über den Parameter pdwCount_p referenziert wurde.
- Neu: Der Wrapper für LabView 8.5 und 8.6 unterstützt nun auch das automatische und zyklische Senden von CAN-Nachrichten. Es werden nun auch die mehrkanaligen USB-CANmodule unterstützt.
- Neu: Mit der neuen API Funktion *UcanSetDebugMode()* kann das Erstellen eines Debug-Logfiles aus der USBCAN-Library heraus eingeschaltet werden.
- Neu: Die Windows Kernel Treiber unterstützen nun auch Windows Vista (32 und 64 Bit). Achtung: GW-001 wird nicht mehr unterstützt.
- Performance-Verbesserungen bei den sysWORXX USB-CANmodulen.
- Änderung: Bei den Baudraten 10kBit/s, 20kBit/s, 50kBit/s und 100kBit/s wurde die "Sync Jump Width" (SJW) bei den sysWORXX Modulen für eine bessere Synchronisation zu anderen CAN-Knoten auf 1 gesetzt.
- Änderung: Der voreingestellte Pfad für die Erstellung einer LOG Datei aus der USBCAN32.DLL heraus wurde auf "Eigene Dateien" gesetzt.

Technische Daten:

- CAN-Schnittstelle:
  - Nach CiA DS 102 / ISO 11898-2/3,
  - Spannungsversorgung galvanisch entkoppelt (optional)
  - Anschluss über SUB-D-9-Stecker
  - CAN-Frameformat nach Spezifikation 2.0A und 2.0B (11- und 29- Bit CAN-Identifier) werden unterstützt
  - Standard ist PCA82C251 als CAN-Transceiver, andere Varianten wie z.B. lowspeed- und single-wire - Treiber möglich
  - unterstützte CAN-Transceiver: 82C251, TJA1054, TJA1041, AU5790
  - Steckplatz zur Adaption weiterer CAN-Transceiver, z.B. B10011S (nur bei GW-002)
  - optional Spannungsversorgung über CAN-Bus, abhängig vom CAN-Transceiver und von der Bestelloption
  - Zwischenpuffer für 768 CAN-Nachrichten pro Richtung im USB-CANmodul
  - Zwischenpuffer für 4096 CAN-Nachrichten pro Richtung auf dem PC (ab Software-Version 3.05 einstellbar)
- USB-Schnittstelle:
  - USB-Buchse Typ B nach USB-Standard
  - Spannungsversorgung über den USB-Bus (max. 200mA im Betriebsmodus) bei GW-001, GW-002, USB-CANmodul1 und USB-CANmodul2
  - Übertragungstyp Bulk, 12MBit/s
- 8-bit Port für individuelle Erweiterungen (nur GW-002 und alle sysWORXX-Module außer USB-CANmodul1)
- power-LED (grün) und state-LED (rot) bei GW-001 und GW-002
- power-LED (gelb), state-LED (rot) und traffic-LED (grün) bei allen sysWORXX-Modulen
- Halter für Hutschiene und Wandhalter optional verfügbar
- Umgebungstemperatur 0°C...+55°C bei GW-001 und GW-002
- Umgebungstemperatur -15°C...+85°C bei allen sysWORXX-Modulen
- CE-konform

Softwareunterstützung:

- Kernel-Mode-Treiber für Windows 2000, XP (32-Bit Edition, ab Treiber V4.00 auch 64-Bit Edition) und Vista (32 und 64 Bit mit der Treiber Version 4.00):
  - USBCANLD.SYS, USBCANL2.SYS, USBCANL3.SYS, USBCANL4.SYS und USBCANL5.SYS für den Download der Firmware in das USB-CANmodul
  - USBCAN.SYS für die Nutzung der Funktionen des USB-CANmoduls
  - UCANNET.SYS (Netzwerktreiber) ermöglicht die Verwendung eines logischen USB-CANmoduls von bis zu 6 Applikationen.
- User-Mode-Treiber für Windows 2000, XP (32-Bit Edition, ab Treiber V4.00 auch 64-Bit Edition) und Vista (32 und 64 Bit mit der Treiber Version 4.00)::
  - USBCAN32.DLL für die einfache Nutzung der Funktionen des USB-CANmoduls
- Es werden bis zu 64 CAN-Kanäle von den Treibern unterstützt.
- Tools für Windows 2000, XP (32-Bit Edition, ab Treiber V4.00 auch 64-Bit Edition) und Vista (32 und 64 Bit mit der Treiber Version 4.00):
  - USB-CANmodul Control (Systemsteuerung) – Verwaltung mehrerer USB-CANmodule durch Vergabe von Gerätenummern
  - PCANView – CAN-Monitorprogramm
- Drei Demoprogramme im Quellcode (Microsoft C/C++ als MFC und Visual Basic als .NET)
- Contributor Interfaces für LabView
- Kernel-Mode-Treiber und Demo für Linux Kernel 2.6
- Treiber und Demo für Windows CE

Lieferumfang:

Fertig aufgebautes und getestetes Gerät, Manual, Software und USB-Standardkabel (Typ A zu Typ B; ca. 1,5m)

Anmerkungen zum EMV-Gesetz für das USB-CANmodul

Das USB-CANmodul ist ein fertig aufgebautes und getestetes Gerät und nur in diesem Zustand zu verwenden.

Im Betrieb dürfen ohne weitere Schutzbeschaltung und Prüfung keine Leitungen von mehr als 3 m Länge an die Verbinder angeschlossen werden. Insbesondere ist geschirmtes CAN-Kabel zu verwenden.
Die CE-Konformität gilt nur für den hier beschriebenen Anwendungsbereich unter Einhaltung der im folgenden Handbuch gegebenen Hinweise zur Inbetriebnahme!
Für weitere Informationen wenden Sie sich bitte an folgende Adresse:

	EUROPE	NORD AMERIKA
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
Web Site:	http://www.systec-electronic.com	http://www.phytec.com
e-mail:	info@systec-electronic.com	Info@phytec.com
Voice:	+(49) 3661-6279-0	+1 (800) 278-9913
Fax:	+(49) 3661-6279-99	+1 (206) 780-9135

1 Inbebriebnahme

1.1 Installation unter Windows

1.1.1 USB-CANmodul auspacken

Packen Sie die einzelnen Komponenten aus und vergewissern Sie sich, dass sie unbeschädigt und vollständig sind. Im Lieferumfang sind enthalten:

- USB-CANmodul
- SYS TEC Produkt CD mit Installationssoftware und Manuals
- USB Anschlusskabel

1.1.2 Treiberinstallation

Achtung:
Die Installation und der Betrieb unter Win98/ME ist möglich, werden jedoch ab Version 2.17 nicht mehr garantiert.

Wenn Sie das USB-CANmodul das erste Mal mit dem PC verbinden, gehen Sie wie folgt vor:

Hinweis:
Führen Sie die Treiberinstallation durch, bevor Sie das USB-CANmodul am PC anschließen. Stellen Sie sicher, dass Sie für die Installation Administratorrechte besitzen (Windows 2000/XP/Vista).
Software-Updates erhältlich unter: http://www.systec-electronic.com

a) Starten Sie Ihren Computer.

b) Legen Sie die SYS TEC Produkt CD in das CD-ROM Laufwerk ein. Installieren Sie nicht von einem Netzwerkserver.

c) Öffnen sie den Windows Explorer.

d) Wechseln Sie in den Pfad „<CD-ROM>:\Products\USB-CANmodul_XXXXXXX\Software\SO-387“ und führen Sie das Setup-Tool “SO-387.exe“ aus.

e) Folgen Sie den Bildschirmanweisungen.

f) Verbinden Sie das USB Kabel mit dem USB-CANmodul und dem Computer.

g) **Windows erkennt automatisch** das USB-CANmodul. Es öffnet sich der Hardware-Assistent. Die Treiberdateien werden automatisch gefunden (*siehe Hinweis*). Bestätigen Sie die Installation der Treiber für das USB-CANmodul.

Hinweis:
Der Treiber ist nicht von Microsoft signiert worden. Aus diesem Grund erscheint auf dem Betriebssystem Windows XP die Meldung, dass der Treiber den Loop-Test nicht bestanden hat. Ignorieren Sie diese Meldung und drücken Sie „Installation fortsetzen“.

Seit Windows Vista in der 64-Bit Edition müssen Kernel-Mode Treiber mit einem Zertifikat versehen werden, was den Hersteller des Treibers identifiziert. Wird der Treiber das erste Mal installiert, dann erscheint bei der Installation ein Fenster, wie in Bild 1 dargestellt. Setzen Sie dort bitte den Haken, dass Software von der Firma SYS TEC electronic GmbH immer vertraut werden soll.

Bild 1: Installation des Treibers unter Windows Vista

Wenn Sie den Treiber des USB-CANmodul aktualisieren wollen, dann gehen Sie wie folgt vor:

a) Starten Sie Ihren Computer.

b) Legen Sie die SYS TEC Produkt CD in das CD-ROM Laufwerk ein.

c) Öffnen Sie den Windows Explorer.

d) Wechseln Sie in den Pfad „<CD-ROM:>\Products\USB-CANmodul_XXXXXXX\Software\SO-387“ und führen Sie das Setup-Tool “SO-387.exe“ aus.

e) Folgen Sie den Bildschirmanweisungen.

1.1.3 Überprüfung der Geräteinstallation

Testen Sie, ob das Modul ordnungsgemäß am Computer angemeldet wurde:

a) Klicken Sie mit der rechten Maustaste auf das Symbol „**Arbeitsplatz**“ im Desktop.

b) Es wird ein Popup-Menü geöffnet. Klicken Sie auf „**Eigenschaften**“ erscheint die Dialogbox „**Eigenschaften von System**“.

c) Wählen Sie dort den „**Geräte-Manager**“. Bei Windows 2000, XP und Vista befindet sich der Geräte-Manager in der Registrierkarte „**Hardware**“.

d) Erscheint unter dem Eintrag „**Universeller serieller Bus Controller**“ bzw. „**USB Controller**“ bzw. „**USB-CAN-Hardware**“ der Eintrag „**Systec USB-CANmodul device driver**“ oder „**Systec USB-CANmodul network driver**“, dann ist das Modul ordnungsgemäß angemeldet.

Hinweis:
Ab der Version 2.16 erscheint bei einer Neuinstallation das USB-CANmodul nicht unter dem Eintrag „USB Controller“, sondern unter dem Eintrag „USB-CAN-Hardware“.

Bild 2: Geräte-Manager mit dem Eintrag USB-CANmodul (Windows 2000)

War die Installation nicht erfolgreich, führen Sie die Installation erneut entsprechend der Anweisung durch. Verläuft die Installation erfolglos, wenden Sie sich bitte an den Hersteller.

1.1.4 Vergabe einer Gerätenummer

Die Vergabe einer Gerätenummer ermöglicht dem Anwender mehrere USB-CANmodule gleichzeitig am PC zu verwenden. Die Gerätenummer identifiziert die einzelnen USB-CANmodule untereinander.

a) Öffnen Sie die Systemsteuerung unter „**Start**“ → „**Einstellungen**“ → „**Systemsteuerung**“. Unter Windows XP und Vista wählen Sie bitte zusätzlich „**Weitere Systemsteuerungsoptionen**“. Bei der 64-Bit Edition von Windows Vista muss "**32 Bit-Systemsteuerungselemente anzeigen**" ausgewählt werden.

b) Klicken Sie auf das Symbol „**USB-CANmodul Control**“.

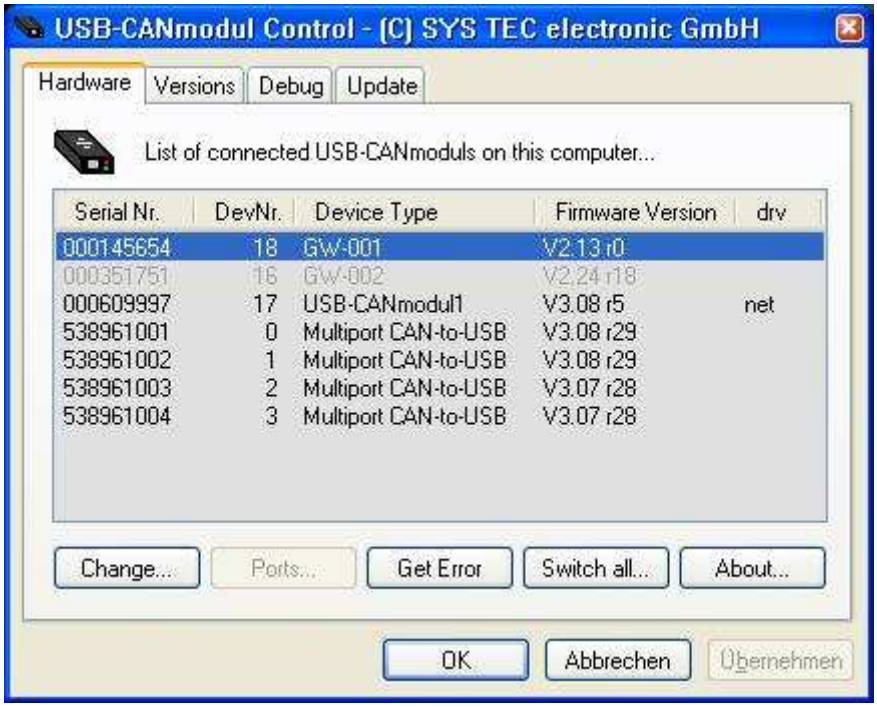


Bild 3: Das Tool USB-CANmodul Control

c) Wählen Sie ein Modul aus der Ihnen angezeigten Liste und klicken Sie auf „**Change...**“.

Hinweis:
Die Gerätenummer von grau dargestellten Modulen können nicht verändert werden, da diese bereits von einer anderen Applikation verwendet werden.

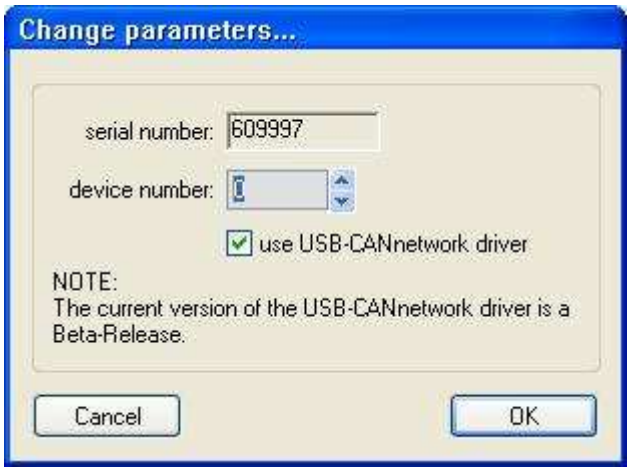


Bild 4: Dialogbox für die Einstellung der Gerätenummer

d) Geben Sie eine neue Gerätenummer ein oder verändern Sie die Gerätenummer durch Klick auf den Up- oder Down-Button.

e) Bestätigen Sie die Änderung mit „**OK**“. Die neue Gerätenummer wird erst mit dem Klick auf die Schaltfläche „**Übernehmen**“ oder „**OK**“ in das USB-CANmodul programmiert.

1.1.5 Anschluss CAN-Knoten

Das USB-CANmodul ist mit einem neunpoligen Sub-D Stecker versehen. Verbinden Sie diesen Stecker über ein entsprechendes Kabel mit einem anderen CAN-Knoten. Die Pinbelegung des Sub-D Steckers ist in *Tabelle 3* beschrieben.

Hinweis:
Achten Sie bitte bei highspeed - CAN-Transceivern (82C251, Standard) darauf, dass das verwendete Kabel mit Abschlusswiderständen zwischen CAN_L und CAN_H (120 Ohm) an beiden Enden des Kabels versehen ist (*siehe auch Kapitel 1.6*). Für lowspeed - CAN-Transceiver (TJA1054 u.a., Option) darf kein Abschlusswiderstand angesteckt werden, dieser ist hier integriert (10kOhm). Bei Leitungslängen >3m ist geschirmtes Kabel zu verwenden.

1.1.6 Starten von PCANView (USBCAN)

Das Tool PCANView ist ein CAN-Busbetrachter für das Betriebssystem Windows.

a) Rufen Sie das Tool aus dem Startmenü **„Start“**, **„Programme“**, **„USB-CANmodul Utilities“**, **„PCANView (USBCAN)“** auf.

b) Es wird ein Fenster **„USB-CANmodul settings“** geöffnet.

Bild 5: Dialogbox mit den Hardwareeinstellungen

c) Tragen Sie die verwendete **Baudrate** (Standardbaudraten nach CiA) und die Gerätenummer des USB-CANmoduls („**any**“ steht für das Modul, das zuerst gefunden wird).

d) Wird im Feld Baudrate „user“ ausgewählt, können die Werte für die Register BTR0 und BTR1 des SJA1000 direkt eingegeben werden. Der SJA1000 wird mit 16MHz getaktet. Die Berechnung der Werte für andere Baudraten ist dem Handbuch zum SJA1000 zu entnehmen. Für das USB-CANmodul der sysWORXX-Reihe geben Sie bitte im Feld „BTR Ext“ die anwenderspezifische

Baudrate ein. Nähere Informationen dazu finden Sie im *Kapitel 2.3.4*.

e) Verwenden Sie das Multiport CAN-to-USB 3004006, das USB-CANmodul2 3204002/3204003, das USB-CANmodul8 3404000 oder das USB-CANmodul16 3404001, dann stellen Sie bitte ein, welcher CAN Kanal verwendet werden soll.

f) Bestätigen Sie die Einstellungen mit „**OK**“.

g) Es erscheint ein neues Fenster namens „**PCANView - Connect to net**“. Tragen Sie dort gegebenenfalls die Message Filterung für den CAN-Controller ein. Bestätigen Sie mit „**OK**“.

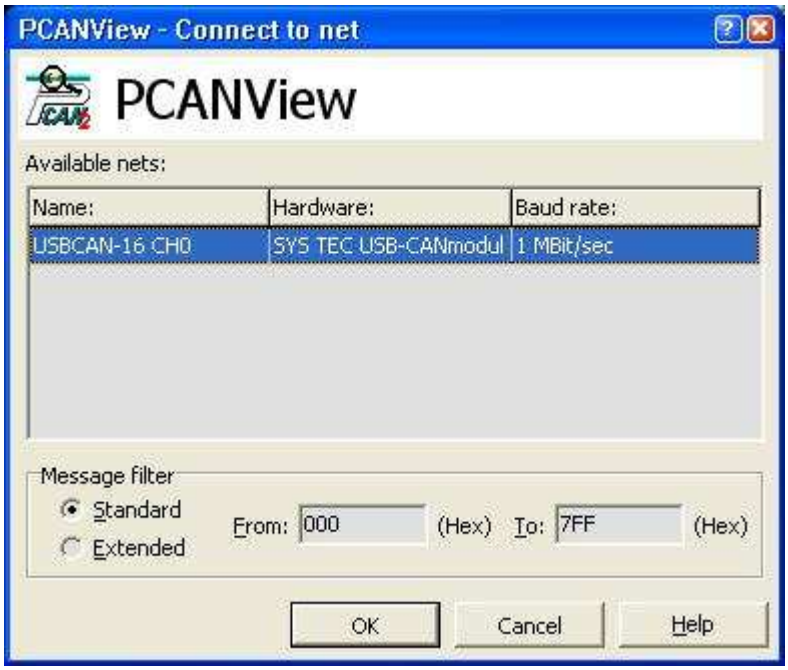


Bild 6: Fenster für die Einstellung der Nachrichtenfilterung

h) Im nächsten Fenster (Hauptfenster) werden im oberen Teil alle empfangenen CAN-Nachrichten angezeigt. Im Menü „**Transmit**“ > „**New...**“ können sie eine neue CAN-Nachricht definieren. Tragen sie im Feld „Period“ eine 0 ein, so wird die CAN-Nachricht nur dann gesendet, wenn sie im unteren Feld des Hauptfensters ausgewählt wurde und die Leertaste gedrückt wird.

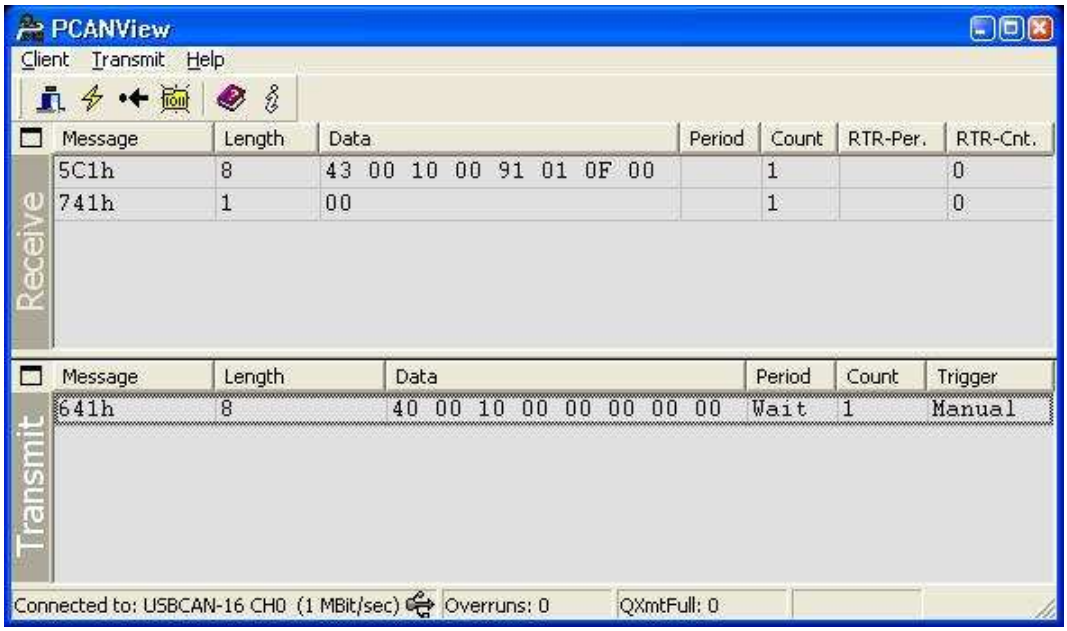


Bild 7: Fenster des Tools PCANView (USBCAN)

1.1.7 Erstellen von Debug-Ausgaben aus der DLL

Sollten Probleme mit der Software auftreten, dann gibt es die Möglichkeit, die Debug-Ausgaben aus der USBCAN32.DLL in eine Log-Datei umzuleiten. Diese Log-Datei sollten Sie uns immer zusenden, um eine schnelle Bearbeitung Ihres Problems zu gewährleisten.

Um dieses Feature zu aktivieren rufen Sie bitte das „**USB-CANmodul Control**“ aus der Systemsteuerung auf. Auf der Seite „**Debug**“ finden Sie dann folgendes Bild:

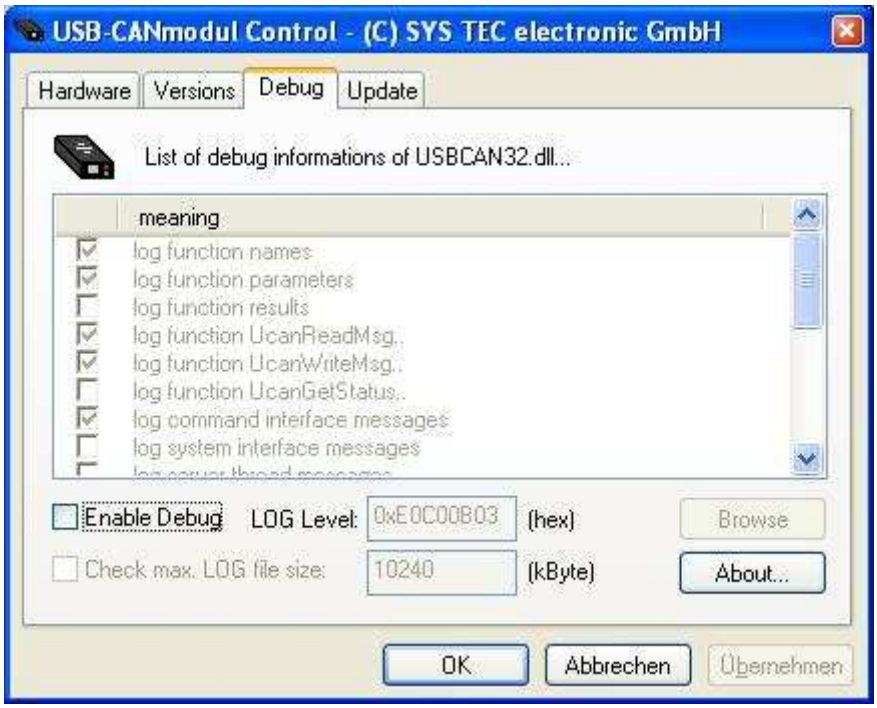


Bild 8: Debug-Einstellungen im USB-CANmodul Control

Setzen Sie einen Haken in das Kästchen „**Enable Debug**“. In der darüberliegenden Liste können Sie dann die einzelnen Informationen aktivieren, die in das Logfile mit aufgenommen werden sollen. Mit der Schaltfläche „**Browse**“ können Sie das Verzeichnis einstellen in welches die Log-Datei gespeichert werden soll. Als Voreinstellung ist C:\ eingestellt.

Übernehmen Sie diese Einstellung und beenden Sie das USB-CANmodul Control wieder. Danach starten Sie Ihre Applikation, die auf das USB-CANmodul zugreift, führen Sie es bis zu Problemstelle und beenden Sie Ihre Applikation wieder.

Danach finden Sie im eingestelltem Verzeichnis eine Datei mit dem Namen USBCAN_XXXXXXXX_YYYYYY_ZZZ.LOG, wobei XXXXXXXX das Erstellungsdatum der Log-Datei im Format JJJJMMTT ist (Jahr Monat Tag), YYYYYY die Erstellungszeit der Log-Datei im Format HHMMSS und ZZZ der Dateiname der aufgerufenen Applikation ist.

Hinweis:
Durch das Aktivieren der Debug-Ausgaben verändert sich die Performance, da dabei mehr Code von den API-Funktionen durchlaufen werden muss, um die Debug-Ausgaben in eine Datei umzuleiten. Das Einschränken von Debug-Informationen über den LOG-Level kann hierbei helfen, die Performance wieder etwas anzuheben. Sie sollten dabei jedoch bedenken, dass dann wichtige Debug-Informationen in der Log-Datei fehlen könnten.

Darüber hinaus können die Log-Dateien recht groß werden. Aktivieren Sie in diesem Fall die Option „Check max. LOG file size“. Dabei überwacht die USBCAN32.DLL die Größe der Log-Datei. Wenn diese überschritten wird, dann werden die vorigen Debug-Ausgaben aus der Log-Datei gelöscht und mit dem Aufzeichnen der Debug-Ausgaben fortgefahren. Voreinstellung für die maximale Größe ist 10240 kByte (d.h. 10 MByte).

Ab der Version V3.11 der USBCAN-Library kann eine Applikation durch Aufruf der Funktion *UcanSetDebugMode()* diesen Modus nachträglich aktivieren. Lesen Sie dazu das *Kapitel 2.3.2.1*.

1.1.8 Aktivierung des Netzwerktreibers

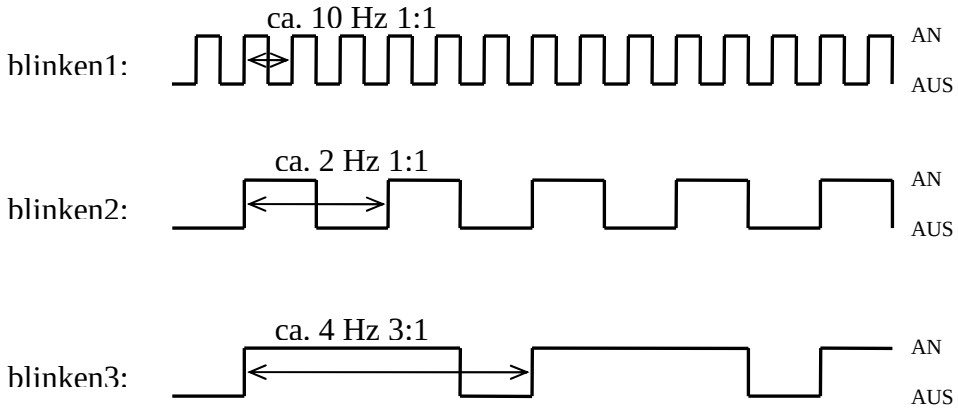
Der Netzwerktreiber UCANNET.SYS wurde entwickelt, damit mehrere Applikationen auf ein physisches USB-CANmodul zurgreifen kann. Dabei baut der Kernel-Mode Treiber ein virtuelles CAN-Netzwerk auf, mit dem sich mehrere Applikationen verbinden können. Alle CAN-Nachrichten, die von einer Applikation gesendet werden, werden dabei nicht nur auf den physischen CAN-Bus gesendet, sondern auch auf die anderen Applikationen verteilt. Beim Empfang von CAN-Nachrichten werden die CAN-Nachrichten an alle Applikationen verteilt.

Der Netzwerktreiber kann nur für sysWORXX USB-CANmodule aktiviert werden, nicht für die älteren Module GW-001 und GW-002.

Um den Netzwerktreiber zu aktivieren starten Sie das USB-CANmodul Control aus der Systemsteuerung. Markieren Sie das Modul, welches Sie für den Einsatz des Netwerktreibers aktivieren möchten. Wählen Sie "**Change…**", dann erscheint der Dialog, wie in Bild 4 dargestellt. Setzen Sie dort den Haken "**use USB-CANnetwork driver**" und bestätigen Sie mit "**OK**". Nach dem Drücken auf "**Übernehmen**" oder "**OK**" im Hauptfenster des USB-CANmodul Control führt das USB-CANmodul eine automatische Neuanmeldung an den PC durch, wobei der Kernel-Mode Treiber ausgetauscht wird. Jetzt können Sie mehrere Applikationen mit diesem USB-CANmodul verwenden.

1.2 Stati der LED-Anzeigen am USB-CANmodul

Die verschiedenen Zustände des USB-CANmoduls werden über 2 bzw. 3 LEDs angezeigt. Damit mehrere Zustände angezeigt werden können, wurden unterschiedliche Blinkrhythmen definiert.



(nicht maßstabsgerecht)

USB-CANmodul angesteckt?	LED grün (power)	LED rot (state)	Beschreibung
nein	aus	aus	Keine Spannung am USB-CANmodul.
ja	an	blinken1	USB-CANmodul meldet sich beim Host an.
ja	an	an	Anmeldung erfolgreich, CAN ist nicht initialisiert, kein Fehler.
ja	an	aus	CAN ist initialisiert, kein Fehler.
ja	an	blinken2	Ein CAN-Fehler ist im USB-CANmodul aufgetreten.

Tabelle 1: Stati der LED-Anzeigen am USB-CANmodul GW-001/GW-002

Beim Multiport CAN-to-USB gibt es eine Status-LED für jeden CAN-Kanal. Zusätzlich zeigt eine grüne „Traffic-LED“ (eine für jeden CAN-Kanal) den Datentransfer auf dem CAN-Bus an, sobald der entsprechende CAN-Kanal initialisiert wurde. Die LEDs des USB-CANmodul2, USB-CANmodul8 und USB-CANmodul16 entsprechen denen des Multiport CAN-to-USB. Beim USB-CANmodul1 ist die Status-LED und die Traffic-LED nur einmal vorhanden, da es dort nur einen CAN-Kanal gibt. Die Blinkrhythmen dieser drei Hardware-Typen sind in *Tabelle 2* aufgelistet.

USB-CANmodul angesteckt?	LED gelb (power)	LED rot (state)	Beschreibung
nein	aus	aus	Keine Spannung an der Hardware.
nein	an	blinken1	USB Kabel nicht verbunden
ja	an	blinken1	Hardware meldet sich beim Host an.
ja	an	an	Anmeldung erfolgreich, CAN-Kanal ist nicht initialisiert, kein Fehler.
ja	an	aus	CAN-Kanal ist initialisiert, kein Fehler.
ja	an	blinken2	Ein CAN-Fehler ist aufgetreten.
ja	an	blinken3	Update der Firmware. In diesem Zustand darf die Betriebsspannung nicht abgeschaltet werden!

Tabelle 2: Stati der LED-Anzeigen an den sysWORXX-Modulen

1.3 CAN-Spannungsversorgung

Für die Standardversion GW-002 und die Versionen GW-002-xx0 ist keine externe CAN-Spannungsversorgung notwendig. Die lowspeed - Versionen GW-002-xx1 und GW-002-xx2 benötigen eine externe Spannungsversorgung für den CAN-Transceiver. Dabei sind die Einschränkungen für die CAN-Transceiver zu beachten.

Der CAN Anschluss ist ein Sub-D-9 Stecker mit folgender Belegung:

Pin	Belegung Sub-D-9 Stecker	
	mit 82C251, 82C252, TJA1041, TJA1054 (differencial)	mit AU5790 (single wire)
1	nicht belegt	nicht belegt
2	CAN-L	nicht belegt
3	GND	GND
4	nicht belegt	nicht belegt
5	CAN-SHLD	CAN-SHLD
6	GND	GND
7	CAN-H	CAN-H
8	nicht belegt	nicht belegt
9	Vcc (+7 bis +30 VDC)	Vcc (+5,3 bis +13 VDC)

Tabelle 3: Pinbelegung des Sub-D-9 Steckers

Anmerkung:
Vcc an Pin 9 ist abhängig vom verwendeten alternativen CAN-Transceiver.

Im Normalfall kann für den Spannungsbereich die Version 30V gewählt werden. Die Nominalspannung beträgt 24V +-25%. Kurzzeitig sind bis 35V zulässig, das Modul arbeitet ab 8V. Durch eine interne Schutzbeschaltung kann bei 12V-Anschluss (+-20%) die Versorgungsspannung VBAT direkt am CAN-Transceiver auf unter 8V sinken, wodurch die Erkennung des standby-Modus unter Umständen nicht funktioniert. Um das zu vermeiden, ist die Version mit externer 12V-Versorgung vorgesehen. Die 12V-Version kann auch bei 24V +20% betrieben werden. Der Einsatz in 24V-Systemen ist möglich, aber nicht zu empfehlen. Auf eine Regelung der Versorgungsspannung wurde verzichtet, um keinen zusätzlichen

Strombedarf zu erzeugen. Es fließt nur der Strom für den CAN-Transceiver.

1.4 CAN-Port für lowspeed - CAN Transceiver

Als Standard wird der highspeed - CAN-Transceiver 82C251 eingesetzt. Alternativ können andere CAN-Transceiver eingesetzt werden, wobei sich lediglich das Verhalten auf dem Bus ändert, nicht aber das Verhalten zur Software. Mit einer Software (z.B. dem mitgelieferten PCANView) lassen sich alle CAN-Transceiver einsetzen.
Die optional einsetzbaren lowspeed - CAN-Transceiver TJA1054 bzw. der single-wire CAN-Transceiver AU5790 besitzen mehrere Signale, um den Betriebsmodus des CAN-Transceivers einzustellen und den Betriebszustand anzuzeigen. Folgende Signale werden unterstützt:

Signal	Name	Bedeutung	Typ	Standardwert
EN	Enable	Einschaltsignal	high-aktiv	high-Pegel
/STB	Standby	Abschaltsignal	low-aktiv	high-Pegel
/ERR	Error	Error-Signal	low-aktiv	high-Pegel
TRM	Termination Resistor	Abschluss-widerstand	high-aktiv	low-Pegel

Tabelle 4: Verfügbare Signale am CAN-Port

Hinweis:
Der Abschlusswiderstand kann Software-seitig nur mit dem USB-CANmodul2 zurückgelesen werden (*siehe Kapitel 1.6*).

Die Standard-Pegel sind so eingestellt, dass die CAN-Transceiver im normalen Betriebsmodus arbeiten. Damit ist der Betrieb mit dem PCANView sofort möglich. Das Error-Signal wird nicht ausgewertet. Funktionen zum Setzen der Betriebsmodi und Lesen des Error-Signals werden durch die USBCAN-Library unterstützt und sind im Abschnitt Softwareunterstützung beschrieben.
Zum Setzen der Betriebsmodi ist das Datenblatt zum jeweiligen CAN-Transceiver zu beachten. Der AU5790 besitzt keinen Error-Ausgang.

Für die Adaption weiterer CAN-Transceiver wie z.B. dem B10011S ist ein Steckplatz für eine Stift- oder Sockelleiste 2 x 7polig im Rastermaß 2,54 mm vorhanden, der folgende Belegung besitzt:

Beschreibung	Pin	Pin	Beschreibung
/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

Tabelle 5: Pin-Belegung des CAN-Port für CAN-Transceiver beim GW-002

Die Widerstände RTL bzw. RTH für den TJA1054 sind mit 1kOhm bestückt. Für den AU5790 ist Rt mit 9,1kOhm und Cul mit 220pF bestückt.

1.5 Port-Erweiterung

Das USB-CANmodul besitzt einen freien 8-bit Port zur Erweiterung der Funktionalität. Einsatzmöglichkeiten wären z.B. der Einbau digitaler Eingänge (im einfachsten Fall Taster) und digitaler Ausgänge (im einfachsten Fall LEDs).
Es ist ein Steckplatz für eine Stift- oder Sockelleiste 2 x 5polig im Rastermaß 2,54 mm vorhanden, der folgende Belegung besitzt:

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

Tabelle 6: Pin-Belegung der Porterweiterung beim GW-002 und USB-CANmodul2

Es sind direkt die Portpins des Microcontrollers angeschlossen (Belastbarkeit der Portpins beachten!). Diese können als Eingang oder Ausgang eingestellt werden. Die 5V-Spannung DC5V wird erst bei

Initialisierung der CAN-Schnittstelle im USB-CANmodul zugeschaltet (nach Aufruf der Funktion *UcanInitCan()* bzw. *UcanInitCanEx()*). Die Portpins sollten nicht mit mehr als 2mA belastet werden, um den Microconroller nicht zu zerstören.
Für eine Adaption Ihrer Schaltung und Software geben wir gern Unterstützung.

Im folgenden Bild ist die Lage der Anschlüsse und Steckplätze dargestellt. Eine detaillierte Zeichnung ist auf Anfrage erhältlich.

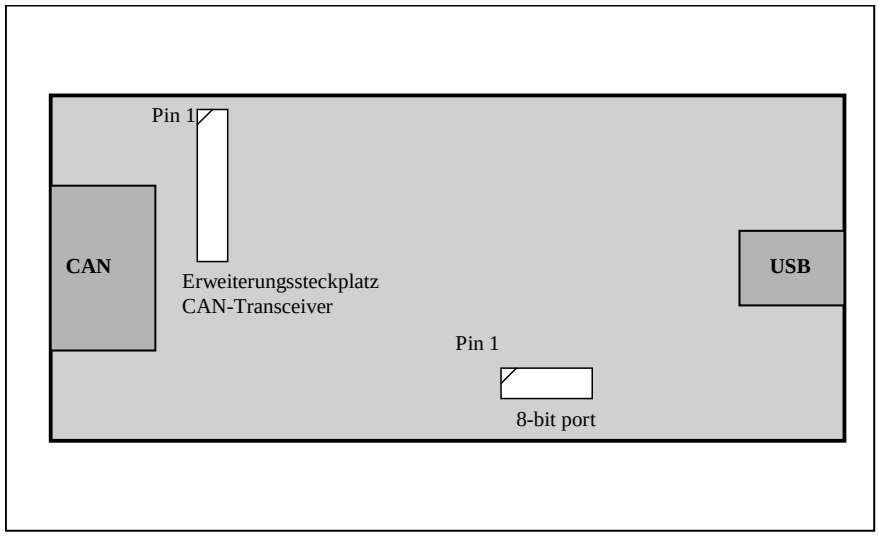


Bild 9: Lage der Erweiterungssteckplätze beim GW-002

Symbol	Parameter	Bedingung	min.	max.	Einheit
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 Capacitance			10	pF
V_{CC}	Supply Voltage		4.75	5.25	V

Tabelle 7: Eigenschaften der Porterweiterung beim GW-002

Funktionen für den Zugriff auf diesen Port werden im *Kapitel 2.3.2.4* beschrieben.

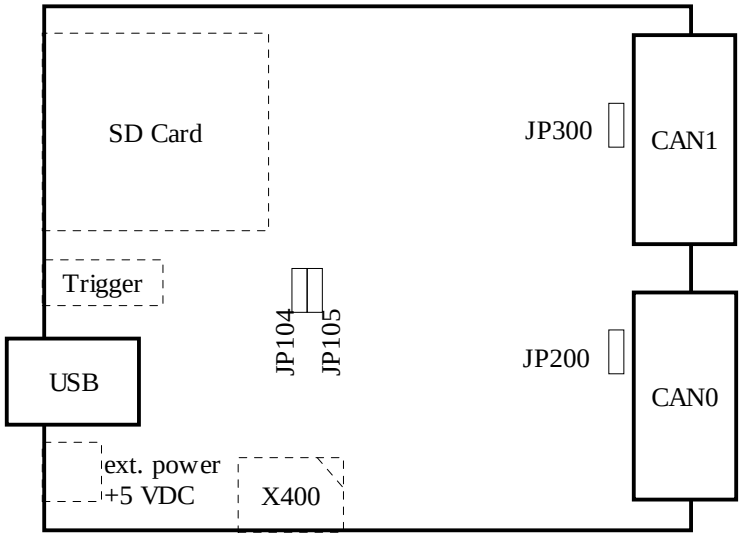


Bild 10: Lage der Erweiterungssteckplätze beim USB-CANmodul2

Die Pinbelegung der Porterweiterung X400 des USB-CANmodul2 entspricht der *Tabelle 6*. Bitte beachten Sie jedoch, dass sich das Pin 1 an der rechten oberen Kante von X400 befindet. Dieser Steckverbinder ist nicht auf jeder Platine bestückt!

Symbol	Parameter	Bedingung	min.	max.	Einheit
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

Tabelle 8: Eigenschaften der Porterweiterung beim USB-CANmodul2

Eine Beschaltung der Porterweiterung hängt immer davon ab, bis zu welcher Ebene man die Hardware des USB-CANmoduls vor Zerstörung schützen muss. Eine einfache Beschaltung ohne Schutzbeschaltung finden Sie im nachfolgenden Bild.

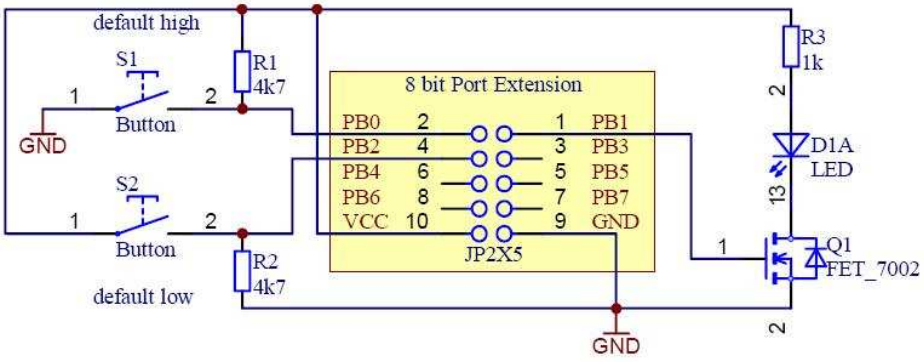
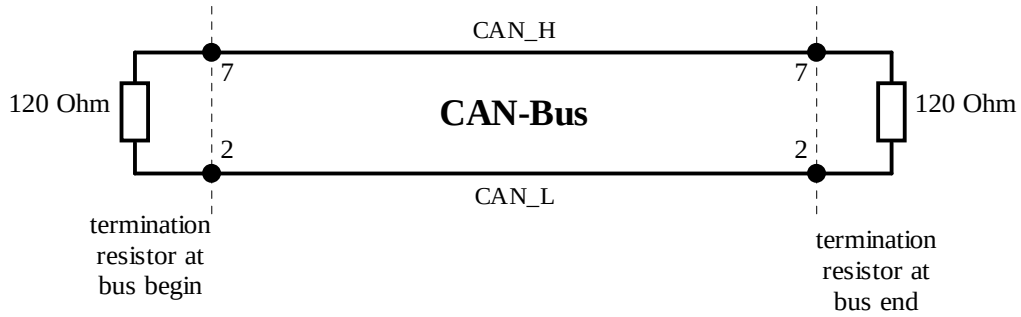


Bild 11: einfache Beispielbeschaltung der Porterweiterung

Beachten Sie bitte auch, dass bei Verwendung von Vcc als Spannungsversorgung der externen Beschaltung die Gesamtstromaufnahme von USB-Geräten nicht 500mA überschreiten darf (beim Anstecken dürfen es sogar nur maximal 100mA sein). Werden Bus-powered USB-Hubs eingesetzt, dann können bereits Probleme unterhalb von 500mA auftreten. Manche USB-Hubs teilen auch den maximal verfügbaren Strom durch die Anzahl der Ports am USB-Hub. Hier können auch Probleme unterhalb von 500mA auftreten, wenn an den anderen Ports weitere USB-Geräte angeschlossen sind. Es wird daher empfohlen, eine galvanisch entkoppelte Variante der externen Beschaltung mit einer eigenen Spannungsversorgung einzusetzen.

1.6 Abschlusswiderstand für den CAN-Bus

Bitte beachten Sie, dass an einem CAN-Bus immer 2 Abschlusswiderstände mit jeweils 120 Ohm angeschlossen werden müssen, wenn Sie ein USB-CANmodul mit einem Highspeed CAN-Transceiver haben. Diese müssen jeweils an den beiden Enden des CAN-Buses erscheinen.



Hinweis:
Für lowspeed - CAN-Transceiver (TJA1054 u.a., Option) darf kein Abschlusswiderstand angesteckt werden, dieser ist bereits integriert.

Für USB-CANmodul2, USB-CANmodul8, USB-CANmodul16 und Multiport CAN-to-USB ist für jeden CAN-Kanal Hardware-seitig ein 120 Ohm Abschlusswiderstand auf der Platine vorgesehen. Dieser kann per Jumper (USB-CANmodul2) oder per Schalter in der Frontblende (USB-CANmodul8, USB-CANmodul16 und Multiport CAN-to-USB) zu oder abgeschaltet werden. Der Standardzustand der Abschlusswiderstände ist: abgeschaltet.

Soll der Abschlusswiderstand für einen CAN-Kanal zugeschaltet werden, dann schalten Sie den entsprechenden Schalter auf ON bzw. schließen Sie den entsprechenden Jumper (*siehe Bild 10* – JP200 für CAN-Kanal 0 und JP300 für CAN-Kanal 1).

Der aktuelle Schaltzustand des Abschlusswiderstandes kann Software-seitig nur beim USB-CANmodul2 indirekt zurückgelesen werden (Verwendung der Funktion *UcanReadCanPort()* oder Anzeige im Tool USB-CANmodul Control /TRM – *siehe auch Bild 13*). Es muss darauf geachtet werden, dass der Jumper JP104 den

gleichen Zustand wie JP200 haben muss (CAN-Kanal 0) und dass der Jumper JP105 den gleichen Zustand wie JP300 haben muss (CAN-Kanal 1). Anderenfalls wird nicht der richtige Zustand des Abschlusswiderstandes zurückgelesen. Der Grund für diese Lösung liegt in der galvanischen Entkopplung der CAN-Busse.

1.7 Bestelloptionen

Das USB-CANmodul ist in folgenden Varianten lieferbar:

Bestellnummer	***Option***
GW-002	Standardmodul, highspeed CAN-Transceiver (Philips 82C251) ohne galvanischer Entkopplung
GW-002-x0x	highspeed CAN-Transceiver (Philips 82C251)
GW-002-x1x	lowspeed CAN-Transceiver (Philips TJA1054)
GW-002-x2x	lowspeed, single-wire CAN-Transceiver (Philips AU5790) *)
GW-002-x3x	highspeed CAN-Transceiver (Philips TJA1041)
GW-002-x4x	lowspeed, high-level CAN-Transceiver (Atmel B10011S)
GW-002-x5x	highspeed CAN-Transceiver (Philips TJA1050)
GW-002-0xx	ohne galvanische Entkopplung
GW-002-1xx	mit galvanischer Entkopplung
GW-002-xx0	interne Versorgung für CAN
GW-002-xx1	externe Versorgung für CAN 7 - 27V *)
GW-002-xx2	externe Versorgung für CAN 12 - 30V *)
GW-002-KSMxx	kundenspezifisch angepasste Version

Alle sysWORXX-Varianten:

3004006	Multiport CAN-to-USB mit 16 CAN-Kanälen, highspeed CAN-Transceiver (Philips 82C251), galvanischer Entkopplung als 8 logische USB-CANmodule je 2 CAN-Kanäle im 19 Zoll Einschub
3204000	USB-CANmodul1 als Ablösung des GW-002-0xx (ohne galvanische Entkopplung), CAN-Transceiver (Philips 82C251)
3204001	USB-CANmodul1 als Ablösung des GW-002-1xx (mit galvanische Entkopplung), CAN-Transceiver (Philips 82C251)
3204002	USB-CANmodul2 mit 2 CAN-Kanälen, ohne galvanische Entkopplung, CAN-Transceiver (Philips 82C251)

3204003	USB-CANmodul2 mit 2 CAN-Kanälen, mit galvanischer Entkopplung, CAN-Transceiver (Philips 82C251)
3204007	USB-CANmodul2 wie 3204000 jedoch mit eine 8 Bit Porterweiterung.
3404000	USB-CANmodul8 mit 8 CAN-Kanälen, CAN-Transceiver (Philips 82C251), galvanischer Entkopplung als 4 logische USB-CANmodule je 2 CAN-Kanäle im Tischgehäuse
3404001	USB-CANmodul16 mit 16 CAN-Kanälen, CAN-Transceiver (Philips 82C251), galvanischer Entkopplung als 8 logische USB-CANmodule je 2 CAN-Kanäle im Tischgehäuse

**) externe Versorgung des CAN-Transceivers nicht für 82C251, AU5790 muss extern versorgt werden*

Zurzeit verfügbare Varianten:

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, 3404000, 3404001

weiteres Zubehör:

WK054	ungeschirmtes CAN–Bus-Kabel für max. 5 Knoten, mit steckbaren Abschlusswiderständen 120Ohm und für Spannungseinspeisung vorbereitet
WK-004	geschirmtes CAN-Kabel für Direktverbindung von 2 Knoten mit integrierten Abschlusswiderständen 120Ohm
GW-002-Z01	Wandhalterung
GW-002-Z02	Adapter DB9 auf 5-pol. Combicon, Belegung wie *DeviceNet*
GW-002-Z03	USB-Kabel 3 m (A-B)
GW-002-Z04	USB-Kabel 4,5 m (A-B)
GW-002-Z05	Halterung für Hutschiene

1.8 Die neuen sysWORXX-Module

1.8.1 Das Multiport CAN-to-USB

Das Multiport CAN-to-USB 3004006 ist ein fertiges Gerät für ein 19“ Einschub. Es beinhaltet 8 logische USB-CANmodule mit je 2 CAN-Kanälen. Die CAN-Kanäle sind galvanisch entkoppelt und sind mit einem highspeed CAN-Transceiver ausgerüstet. Die 8 logischen USB-CANmodule werden über 2 USB-Hubs zu 2 USB-Ports zusammengefasst.

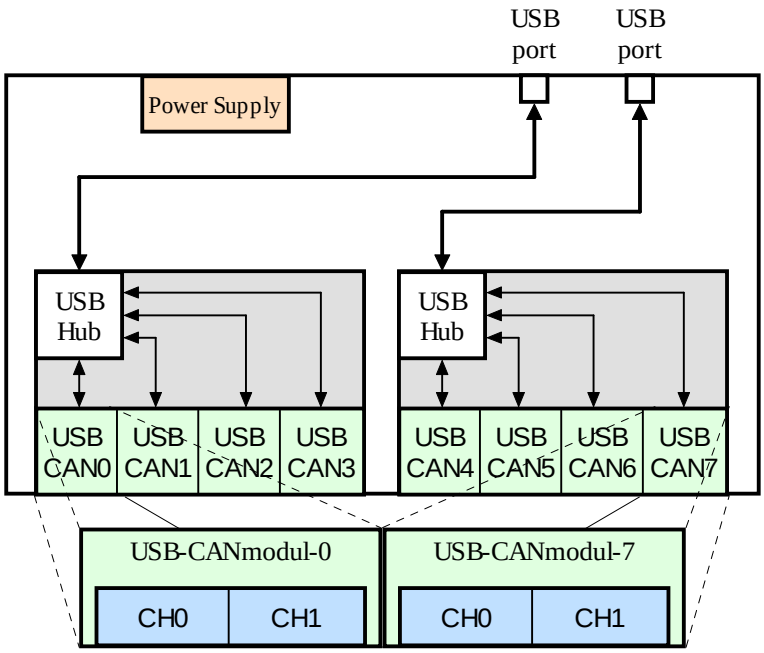


Bild 12: Schematischer Aufbau des Multiport CAN-to-USB

Es gibt keine gesonderten Treiber für das Multiport CAN-to-USB. Es werden die bestehenden Treiber für das USB-CANmodul verwendet. Für das Multiport CAN-to-USB sind erweiterte Funktionen in die USBCAN-Library implementiert worden. Diese erweiterten DLL-API-Funktionen sind auch für das USB-CANmodul GW-002 verwendbar. Bis zu einem bestimmten Level sind die bisherigen DLL-API-Funktionen (aus Software-Version kleiner 3.00) auch für das Multiport CAN-to-USB verwendbar. Die Unterschiede liegen in der

Anzahl der CAN-Kanäle, Baudrateneinstellung und Akzeptanzfilterung. Lesen Sie dazu auch die entsprechenden Kapitel *2.3.4, 2.3.5 und 2.3.6*.

Die Gerätenummern der 8 logischen USB-CANmodule sind nach der Produktion fortlaufend vorprogrammiert. Das heißt das erste logische USB-CANmodul (von links gezählt) trägt die Gerätenummer 0, das zweite die Gerätenummer 1 usw. Diese Gerätenummern könne wie auch beim USB-CANmodul GW-002 umprogrammiert werden. Dazu verwenden Sie das Symbol „USB-CANmodul Control“ aus der Systemsteuerung.

1.8.2 Das USB-CANmodull1

Das USB-CANmodul1 3204000 / 3204001 ist eine kostenoptimierte Variante der sysWORXX-Module mit nur einem CAN-Kanal. Die CAN-Schnittstelle ist optional galvanisch entkoppelt oder nicht galvanisch entkoppelt erhältlich. Beide Bestellvarianten sind mit einem highspeed CAN-Transceiver ausgerüstet. Es gibt KEINE Porterweiterung zur Steuerung digitaler I/Os.

1.8.3 Das USB-CANmodull2

Das USB-CANmodul2 3204002 / 3204003 ist ein erweitertes sysWORXX-Modul mit zwei CAN-Kanälen je Modul. Beide CAN-Schnittstellen sind optional galvanisch entkoppelt oder nicht galvanisch entkoppelt erhältlich. Beide Bestellvarianten sind mit einem highspeed CAN-Transceiver ausgerüstet. Auf Anfrage kann auch ein lowspeed CAN-Transceiver eingesetzt werden. Mit der Atikelnummer 3204007 erhalten Sie ein Modul, mit Zugang auf die 8 Bit Porterweiterung, wie sie in *Kapitel 1.5* beschrieben ist.

1.8.4 Das USB-CANmodull8 und 16

Das USB-CANmodul8 3404000 und USB-CAnmodul16 3404001 sind identisch mit dem Multiport CAN-to-USB, werden jedoch mit einem Tischgehäuse ausgeliefert. Im USB-CANmodul16 sind 2 Platinen vom selben Typ untergebracht, wie in USB-CANmodul8 eingebaut ist.

2 Softwareunterstützung unter Windows

2.1 Verzeichnisstruktur

Wenn bei der Installation der USB-CANmodul Utilities kein anderer Zielpfad angegeben wird, dann werden alle Dateien in das Verzeichnis „C:\Programme\SYSTEC-electronic\ USB-CANmodul Utility Disk“ installiert. Der Inhalt dieses Verzeichnisses ist in *Tabelle 9* angegeben. Einige Unterverzeichnisse werden nur dann angelegt, wenn sie bei der Installation ausgewählt wurden.

Unterverzeichnis	Inhalt
Bin	Programmdateien (PCANView)
Contrib	beigesteuerte Files anderer Firmen
Borland Delphi	Delphi Klasse mit Demo im Source (wurde entfernt)
LabView	LabView Unterstützung mit Demo
Demo.api	MFC Demo im Source für MS Visual Studio 6.0 oder höher
DemoGW006	MFC Demo im Source für MS Visual Studio 6.0 oder höher und einem USB-CANmodul mit 2 CAN Kanälen
DemoCyclicMsg	MFC Demo im Source für MS Visual Studio 6.0 oder höher für das automatische Senden zyklischen CAN-Nachrichten.
Docu	Handbücher
Include	C – Header Dateien für die USBCAN32.DLL. Die Demoapplikationen für MS Visual Studio 6.0 referenzieren diese Header-Dateien.
Lib	Allgemeine USBCAN32.DLL und Importlibrary für MS Visual Studio. Die Demoapplikationen für MS Visual Studio 6.0 referenzieren diese Importlibrary.
UcanDotNET	Wrapper-DLL im Source für die Verwendung der USBCAN32.DLL unter MS .NET Projekte
USBcanDemoNET	MS Visual Basic .NET Demo im Source unter Verwendung der Wrapper-DLL UcanDotNET.dll

Tabelle 9: Verzeichnisstruktur der installierten Software

2.2 Tools für das USB-CANmodul

2.2.1 USB-CANmodul Control

Das Tool USB-CANmodul Control ersetzt das Tool UCAN Config ab der Version 2.18. Dieses Tool kann wahlweise aus der Systemsteuerung oder aus der Programmgruppe „USB-CANmodul Utilities“ gestartet werden. Das *Bild 3* zeigt das Tool nach dem Start.

Mit diesem Tool kann die Gerätenummer der USB-CANmodule geändert werden (*siehe dazu auch Kapitel 1.1.4*).

Zusätzlich kann mit diesem Tool die 8 Bit Port-Erweiterung (*siehe Kapitel 1.5*) und das CAN-Port für lowspeed CAN-Transceiver (*siehe Kapitel 1.4*) manipuliert werden. Dazu muss aus der Liste das entsprechende USB-CANmodul ausgewählt und auf die Schaltfläche „**Ports...**“ geklickt werden. Das *Bild 13* zeigt die Dialogbox, die dabei geöffnet wird.

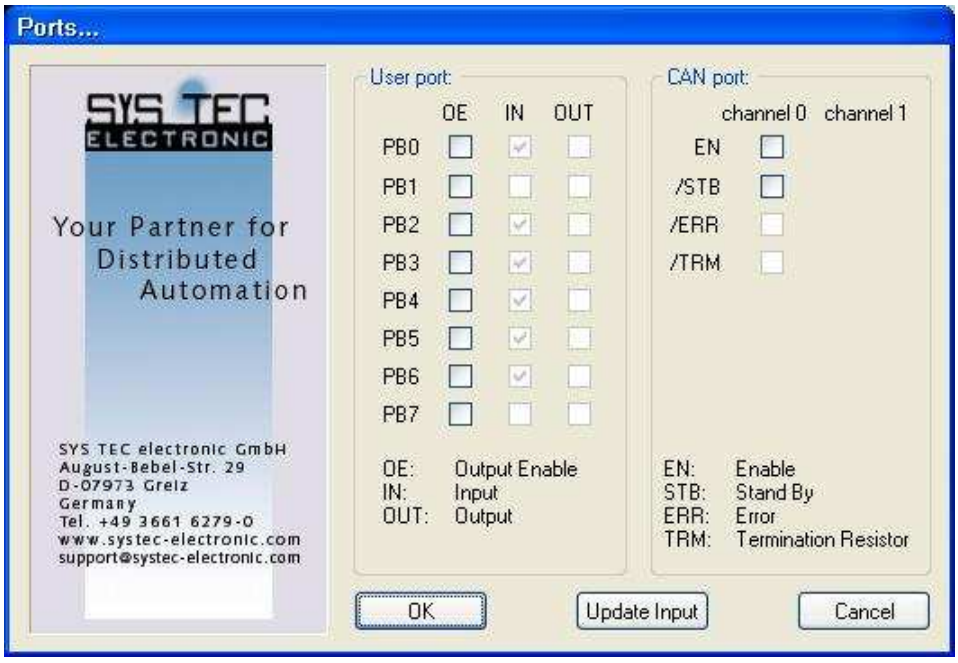


Bild 13: Dialogbox zur Manipulation der Port-Erweiterung und des CAN-Ports

Am Anfang sind alle 8 Leitungen als Eingang konfiguriert. Mit der Spalte OE wird die entsprechende Leitung als Ausgang geschalten. Damit wird das Kästchen für den Ausgangswert in der Spalte OUT aktiviert. Wird in dieser Spalte eine Leitung auf logisch 1 gesetzt, dann wird die entsprechende Leitung an der Port-Erweiterung auf High gesetzt. Bei jeder Änderung wird der aktuelle Zustand der Port-Erweiterung wieder eingelesen und in der Spalte IN für die Eingänge dargestellt. Um die aktuellen Zustände der Eingänge einzulesen, ohne dass ein Ausgang geändert werden muss, klicken Sie auf die Schaltfläche „**Update Input**“

Auf der rechten Seite des Fensters ist der aktuelle Zustand des CAN-Ports für lowspeed CAN-Transceiver dargestellt. Die Leitungen EN und /STB sind Ausgänge und die Leitung /ERR ist ein Eingang. Für nähere Informationen lesen Sie das *Kapitel 1.4*.

2.2.2 PCANView (USBCAN) für Windows

Das Windows-Tool PCANView (USBCAN) dient zur Visualisierung von CAN-Nachrichten, die über den CAN-Bus gesendet werden.

Nach dem Start des Tools wird eine Dialogbox zur Einstellung der Hardware geöffnet (*siehe Bild 5).* Die USB-Gerätenummer ist die Nummer des USB-CANmoduls, wie sie mit dem Tool USB-CANmodul Control (*siehe Kapitel 1.1.4*) eingestellt worden ist. Mit der Baudrate stellt man die Bitrate für den CAN-Bus ein. Mit der Option „listen only“ wird der CAN-Controler so konfiguriert, dass er nur CAN-Nachrichten empfangen kann. Dabei wird kein Acknowledge zum sendenden CAN-Knoten zurückgesendet. Für zweikanalige logische USB-CANmodule muss auch der zu verwendende CAN Kanal ausgewählt werden.

Nach Bestätigung mit „**OK**“ wird ein Fenster nach *Bild 6* geöffnet. Hier können Sie eine CAN-Nachrichtenfilterung eingeben. Die Filterung richtet sich nach dem Format der CAN-Nachricht; ob Sie CAN-Nachrichten mit 11 Bit Identifier (Standard = CAN Spec. 2.0A) oder 29 Bit Identifier (Extended = CAN Spec. 2.0B) filtern möchten. Bitte wählen Sie vorher das richtige Format aus. Geben Sie in den Feldern „From“ und „To“ die CAN-Identifier der CAN-Nachrichten

ein, die Sie mit dem Tool empfangen möchten. Ändern Sie diese Werte nicht, werden alle CAN-Nachrichten angezeigt. Wählen Sie "**OK**" um alle Einstellungen zu bestätigen.

Es wird das Hauptfenster geöffnet *(Bild 7)*. Im oberen Teil des Fensters werden die empfangenen CAN-Nachrichten aufgelistet. Im unteren Teil werden zu sendende CAN-Nachrichten aufgelistet.

Der hexadezimale CAN-Identifier ("**Message**") kann auch 29 Bit groß sein (nach der CAN-Spezifikation 2.0B). Dabei wird der Identifier 8-stellig angezeigt, wobei der Identifier einer 11 Bit CAN-Nachricht (nach der CAN-Spezifikation 2.0A) 3-stellig angezeigt wird. Hat die CAN-Nachricht weniger als 8 Datenbytes, so werden die Datenbytes, die nicht mehr zur CAN-Nachricht gehören mit „--" gekennzeichnet. Die Spalte „**Count**" gibt die Anzahl wieder, wie oft diese CAN-Nachricht mit diesem Identifier empfangen oder gesendet wurden ist. Die Intervallzeit „**Period**" gibt die Zeit zwischen jeder Sendung bzw. zwischen jedem Empfang dieser CAN-Nachricht an.

Im Menü „**Transmit**" > „**New...**" können neue CAN-Nachrichten erzeugt werden, die gesendet werden sollen. Geben Sie den CAN-Identifier, die Datenlänge, die Daten sowie das Frame Format und die Sendeperiode ein. Eine Sendeperiode von 0 heißt, dass diese CAN-Nachricht nur manuell gesendet werden kann. Wählen Sie dazu diese CAN-Nachricht im unteren Teil des Hauptfensters aus und drücken Sie die Leertaste.

Hinweis:
Diese beiden Listen sind nach CAN-Identifier sortiert. Es wird also keine zeitliche Abfolge der CAN-Nachrichten dargestellt.

2.3 Beschreibung der USBCAN-Library

Die USBCAN-Library ist eine Funktionslibrary für Anwendungsprogramme. Unter Windows 2000/XP oder höher handelt es sich hierbei um eine Dynamic Linked Library (DLL – mit dem Dateiname USBCAN32.DLL) . Sie dient als Schnittstelle zwischen der System-Treiberschicht und einem Anwendungsprogramm. Die USBCAN-Library für das USB-CANmodul soll die Verwendung des USB-CAN-Systemtreibers vereinfachen. Sie sorgt für die Verwaltung der geöffneten USB-CANmodule und für die Übersetzung der USB-Daten in CAN-Nachrichten.

Zur Einbindung der DLL in ein eigenes Microsoft Visual C/C++ Projekt, kann die USBCAN32.LIB mit zum Projekt hinzugefügt werden. Dabei wird die USBCAN32.DLL automatisch geladen, wenn das Anwendungsprogramm gestartet wird. Wird die LIB nicht zum Projekt hinzugefügt, oder es handelt sich um eine andere Entwicklungsumgebung (z.B. Borland C++ Builder), so muss die DLL mit der Windows-Funktion *LoadLibrary()* nachgeladen und die Libraryfunktionen mit der Funktion *GetProcAdress()* hinzugefügt werden (*siehe auch Demo-Applikation „DemoGW006“*). Für Microsoft .NET Anwendungen wurde eine .NET Wrapper DLL implementiert, die in *Kapitel 2.4* beschrieben ist.

Die PUBLIC-Direktive der Aufruffunktionen der DLL sorgt für eine standardisiere Aufrufschnittstelle zum Anwender. So ist sichergestellt, dass auch Anwender einer anderen Programmiersprache (Pascal, ...) diese Funktionen nutzen können.

In diesem Handbuch wird die DLL als USBCAN-Library bezeichnet, da die Funktions-API (auch als USBCAN-API bezeichnet) auch für andere Plattformen gültig ist, wobei die Library einen anderen Dateinamen trägt (z.B. unter Windows CE) bzw. gar keine DLL ist (z.B. unter Linux).

Unter dem Pfad <SETUP_DIR>\DEMO.API, <SETUP_DIR>\DEMOGW006 und <SETUP_DIR>\DEMOCYCLICMSG befinden sich Demoprogramme, welche mit Microsoft Visual C/C++ 6.0 und 7.0 für MFC erstellt wurden. In diesen Projekten wird die Handhabung der Funktionen dieser DLL gezeigt.

2.3.1 Das Konzept der USBCAN-Library

Mit der USBCAN-Library können bis zu 64 USB-CANmodule gleichzeitig verwendet werden (unter Windows CE sind es nur 9). Dabei spielt es keine Rolle, ob diese 64 Module von einem oder mehreren Anwendungsprogrammen verwendet werden. Es ist jedoch nur unter Windows 2000/XP/Vista möglich, ein USB-CANmodul von mehreren Anwendungsprogrammen zu verwenden, wenn der USB-CANnetwork Treiber für ein sysWORXX USB-CANmodul aktiviert ist.

Bei der Verwendung dieser DLL entstehen für jedes USB-CANmodul drei Zustände für die Software. Nachdem das Anwendungsprogramm gestartet und die DLL geladen wurde, befindet sich die Software im Zustand DLL_INIT. Dabei wurden alle notwendigen Ressourcen für die DLL angelegt. Wird die Library-Funktion *UcanInitHardware()* bzw. *UcanInitHardwareEx()* gerufen, so wechselt die Software in den Zustand HW_INIT. Hier sind alle Ressourcen angelegt, die für die Kommunikation mit dem USB-CANmodul notwendig sind. Es können jedoch noch keine CAN-Nachrichten gesendet oder empfangen werden. Erst wenn aus dem Zustand HW_INIT die Library-Funktion *UcanInitCan(), UcanInitCanEx()* bzw. *UcanInitCanEx2()* gerufen wurde, können im Zustand CAN_INIT auch CAN-Nachrichten gesendet oder empfangen werden. Mit der Library-Funktion *UcanDeinitCan()* gelangt man vom Zustand CAN_INIT in den Zustand HW_INIT zurück und von dort mit *UcanDeinitHardware()* in den Zustand DLL_INIT. Erst jetzt darf das Anwendungsprogramm beendet werden.

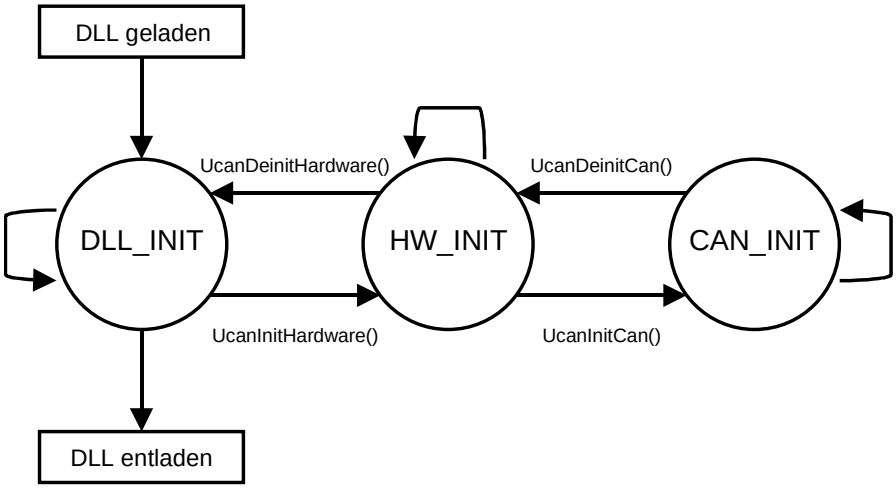


Bild 14: Zustände der Software

Der Funktionsumfang in den Zuständen der Software ist unterschiedlich. So führt zum Beispiel die Funktion *UcanWriteCanMsg()* im Zustand DLL_INIT zu einem Fehler. *Tabelle 10*: zeigt den Funktionsumfang in jedem Zustand.

Werden mehrere USB-CANmodule in einer Anwendung verwendet, so sind die Zustände für jedes USB-CANmodul zu verstehen. Während das erste USB-CANmodul bereits im Zustand CAN_INIT ist, kann sich das zweite noch im Zustand DLL_INIT befinden.

Zustand	**Funktionsumfang**	**veraltet**		**sysWORXX**	
		GW-001	**GW-002**	**3004006 3204002 3204003 3204007 3404000 3404001**	**3204000 3204001**
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

Zustand	Funktionsumfang	veraltet		sysWORXX	
		GW-001	**GW-002**	**3004006 3204002 3204003 3204007 3404000 3404001**	**3204000 3204001**
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

Tabelle 10: Funktionsumfang der Softwarezustände

Bedeutung der Angaben in *Tabelle 10*:

"-" = Funktion wird nicht unterstützt.
"X" = Funktion wird ohne Einschränkungen unterstützt.
"CH0" = Funktion wird für jedes einkanalige Modul bzw. nur für CAN-Kanal 0 eines zweikanaligen logischen Moduls unterstützt, da der Parameter für den CAN-Kanal fehlt.
"XH0" = Funktion wird nur mit dem Parameter für CAN-Kanal 0 eines logischen Moduls unterstützt, da die Hardware nur einen CAN-Kanal besitzt.

2.3.2 Funktionen der USBCAN-Library

Dieses Kapitel beschreibt die Funktionen, die mit der USBCAN-Library zur Verfügung gestellt werden. Der größte Teil der Funktionen geben einen Wert vom Typ UCANRET zurück. Es enthält einen Fehlercode, dessen Bedeutung für jede Funktion dieselbe ist. Neben dem Syntax, der Bedeutung und den Parametern werden auch die möglichen Fehlercodes jeder Funktion angegeben.

Einige der erweiterten Funktionen haben einen weiteren Parameter erhalten, der den CAN-Kanal bestimmt, auf dem die Funktion im Multiport CAN-to-USB 3004006 bzw. USB-CANmodul2 3204002/3204003 ausgeführt werden soll. Wird die „Vorgänger-Funktion“ aufgerufen, dann wirkt sie immer auf den ersten CAN-Kanal (als Kanal 0 bezeichnet). Die erweiterten Funktionen können auch auf die USB-CANmodule GW-001 und GW-002 angewendet werden. Wird dabei jedoch der zweite CAN-Kanal adressiert (als Kanal 1 bezeichnet), dann wird der Fehlercode USBCAN_ERR_ILLCHANNEL zurückgegeben (*siehe Kapitel 2.3.3*).

2.3.2.1 Allgemeine Funktionen

UcanSetDebugMode

Syntax:

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

Verwendbarkeit:
DLL_INIT, HW_INIT, CAN_INIT, ab Version 3.11

Bedeutung:
Diese Funktion schaltet die Erstellung einer Debug Log-Datei aus der USBCAN-Library ein. Sollte dieses Feature bereits von über das USB-CANmodul Control aktiviert worden sein, dann wir der Inhalt dieser automatisch generierten Log-Datei in die neue Datei kopiert und an dieser Stelle fortgefahren.

Parameter:

dwDbgLevel_p: Bit-Maske mit der die einzelnen Debug-Informationen aktiviert werden können, die in die Log-Datei aufgenommen werden sollen. Diese Bit-Maske hat die gleiche Bedeutung wie das „LOG-Level“ aus dem USB-CANmodul Control und soll daher hier nicht näher erläutert werden.

pszFilePathName_p: Pfad auf eine Text-basierende Datei, die mit den Debug-Informationen geschrieben wird. Wird hier ein NULL-Pointer übergeben, dann wird nur der Wert des Parameters *dwDbgLevel_p* übernommen.

dwFlags_p: Zusätzliches Flag. Der Wert 0 erstellt eine neue Datei. Sollte die Datei, die mit dem Parameter *pszFilePathName_p* angegeben wurde, bereits bestehen, dann wird deren Inhalt nach dem Öffnen gelöscht. Der Wert 1 hängt jedoch bei einer bereits existierenden Datei die neuen

Debug-Informationen hinten an (Append-Modus).

Rückgabewert: Gibt diese Funktion FALSE zurück, dann konnte die Log-Datei nicht angelegt werden. Eine mögliche Ursache kann sein, dass der angegebene Pfad nicht existiert.

Beispiel :

```
// 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);
```

Verwendbarkeit:
DLL_INIT, HW_INIT, CAN_INIT

Bedeutung:
Gibt die Softwareversionsnummer der USBCAN-Library zurück. Diese Funktion ist veraltet und sollte nicht mehr verwendet werden. Verwenden Sie stattdessen die Funktion *UcanGetVersionEx()*.

Parameter:

Rückgabewert: Softwareversionsnummer als DWORD mit folgendem Format:

Bit 0 bis 7:	niederwertige Versionsnummer im Binärformat
Bit 8 bis 15:	höherwertige Versionsnummer im Binärformat
Bit 16 bis 30:	reserviert
Bit 31:	1 = kundenspezifische Version

Beispiel:

```
DWORD dwVersion;
_TCHAR szVersion[8];
...
// Versionsnummer holen
dwVersion = UcanGetVersion ();

// in einen String umwandeln
_stprintf (szVersion, _T(„V%d.%2d“), (dwVersion & 0xff00) >> 8,
     dwVersion & 0xff);
...
```

UcanGetVersionEx

Syntax:

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

Verwendbarkeit:
DLL_INIT, HW_INIT, CAN_INIT ab Version 2.16

Bedeutung:
Gibt die Versionsnummer der einzelnen Softwaremodule zurück.

Parameter:

VerType_p: Typ der Versionsinformation gibt an, von welchem Softwaremodul die Version zurückgegeben werden soll. Die *Tabelle 11* gibt alle möglichen Werte für diesen Parameter an. Das Format der Versionsinformation unterscheidet sich von dem der Funktion *UcanGetVersion()*.

VerType_p	Wert	Bedeutung
kVerTypeUserDll kVerTypeUserLib	0x0001	Gibt die Version der USBCAN-Library zurück.
kVerTypeSysDrv	0x0002	Gibt die Version der USBCAN.SYS zurück.
kVerTypeNetDrv	0x0004	Gibt die Version der UCANNET.SYS zurück (Netzwerktreiber).
kVerTypeSysLd	0x0005	Gibt die Version des Loaders USBCANLD.SYS für das GW-001 zurück.
kVerTypeSysL2	0x0006	Gibt die Version des Loaders USBCANL2.SYS für das GW-002 zurück.
kVerTypeSysL3	0x0007	Gibt die Version des Loaders USBCANL3.SYS für das Multiport CAN-to-USB zurück.
kVerTypeSysL4	0x0008	Gibt die Version des Loaders USBCANL4.SYS für das USB-CANmodul1 zurück.

VerType_p	Wert	Bedeutung
kVerTypeSysL5	0x0009	Gibt die Version des Loaders USBCANL5.SYS für das USB-CANmodul2 zurück.
kVerTypeCpl	0x000A	Gibt die Version des USB-CANmodul Control zurück.

Tabelle 11: Konstanten für die Funktion UcanGetVersionEx()

Rückgabewert: Softwareversionsnummer als DWORD mit folgendem Format:

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

Beispiel:

```
DWORD dwVersion;
_TCHAR szVersion[16];
...
// Versionsnummer der USBCAN-Library holen
dwVersion = UcanGetVersionEx (kVerTypeUserDll);

// in einen String umwandeln
_stprintf (szVersion, _T(„V%d.%02d.%d“),
    USBCAN_MAJOR_VER(dwVersion),
    USBCAN_MINOR_VER(dwVersion),
    USBCAN_RELEASE_VER(dwVersion));
...
```

UcanGetFwVersion

Syntax:

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

Verwendbarkeit:
HW_INIT, CAN_INIT ab Version 2.18

Bedeutung:
Gibt die Versionsnummer der Software im USB-CANmodul zurück.

Parameter:

UcanHandle_p:	USB-CAN-Handle, das mit der Funktion *UcanInitHardware()* bzw. *UcanInitHardwareEx()* erhalten wurde.

Rückgabewert: Softwareversionsnummer als DWORD mit folgendem Format:

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

Das Format der Versionsnummer gleicht dem Format aus der Funktion *UcanGetVersionEx()*.

UcanInitHwConnectControl

Syntax:

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

Verwendbarkeit:
DLL_INIT, HW_INIT, CAN_INIT

Bedeutung:
Initialisiert die Überwachung für neu angesteckte USB-CANmodule. Wird ein neues Modul an den Rechner angesteckt, so wird die als Parameter übergebene Callback Funktion gerufen. Auch wenn ein bereits angestecktes Modul vom Rechner getrennt wird, wird diese Callback Funktion gerufen.

Parameter:

fpConnectControlFkt_p: Adresse auf die Callback Funktion, die gerufen werden soll, wenn ein neues USB-CANmodul angesteckt oder abgezogen wird. Diese Adresse darf nicht NULL sein!

Die Callback Funktion muss folgendes Format aufweisen *(siehe Kapitel 2.3.7)*:

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

Rückgabewert: Fehlercode der Funktion.
USBCAN_SUCCESSFUL
USBCAN_WARN_NULL_PTR

UcanInitHwConnectControlEx

Syntax:

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

Verwendbarkeit:
DLL_INIT, HW_INIT, CAN_INIT ab Version 3.00

Bedeutung:
Initialisiert die Überwachung für neu angesteckte USB-CANmodule. Wird ein neues Modul an den Rechner angesteckt, so wird die als Parameter übergebene Callback Funktion gerufen. Im Gegensatz zur Funktion *UcanInitHwConnectControll()* wird hier ein weiteres anwenderspezifisches Argument angegeben, das beim Aufruf der Callback Funktion mit übergeben wird. Hier kann der Anwender zum Beispiel Informationen über die verwendete Instanz hinterlegen.

Achtung:
Diese Funktion wird als Alternative zur Funktion *UcanInitHwConnectControl()* verwendet. Es dürfen nicht beide Funktionen gleichzeitig innerhalb einer Applikation verwendet werden.

Parameter:

fpConnectControlFktEx_p:	Adresse auf die Callback Funktion, die gerufen werden soll, wenn ein neues USB-CANmodul angesteckt oder abgezogen wird. Diese Adresse darf nicht NULL sein!
pCallbackArg_p:	Anwenderspezifisches Argument, das beim Aufruf der Callback Funktion mit übergeben wird.

Die Callback Funktion muss folgendes Format aufweisen *(siehe Kapitel 2.3.7)*:

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

Rückgabewert: Fehlercode der Funktion.
USBCAN_SUCCESSFUL
USBCAN_WARN_NULL_PTR

UcanDeinitHwConnectControl

Syntax:

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

Verwendbarkeit:
DLL_INIT, HW_INIT, CAN_INIT

Bedeutung:
Diese Funktion beendet die Überwachung der neu angesteckten und abgezogenen USB-CANmodule. Sie muss gerufen werden, wenn in einer Anwendung die Funktion *UcanInitHwConnectControl()* oder *UcanInitHwConnectControlEx()* gerufen wurde und die Anwendung beendet wird.

Rückgabewert: Fehlercode der Funktion.
USBCAN_SUCCESSFUL

UcanInitHardware

Syntax:

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

Verwendbarkeit:
DLL_INIT

Bedeutung:
Initialisiert ein USB-CANmodul. Dabei gelangt die Software in den Zustand HW_INIT. Hier können dann die dafür erlaubten Funktionen aus *Tabelle 10* gerufen werden. Wurde die Funktion erfolgreich ausgeführt, dann übergibt die Funktion an die Variable **pUcabHandle_p* ein USB-CAN-Handle, mit dem dann andere Funktionen aufgerufen werden müssen.

Parameter:

pUcanHandle_p:	Zeiger auf die Variable für das USB-CAN-Handle. Dieser Zeiger darf nicht NULL sein!
bDeviceNr_p:	Gerätenummer des USB-CANmoduls (0 – 254). Der Wert *USBCAN_ANY_MODULE* (= 255) sorgt dafür, dass das USB-CANmodul verwendet wird, welches zuerst gefunden wurde.
fpCallbackFkt_p:	Adresse auf die Callback Funktion dieses USB-CANmoduls. Dieser Wert kann NULL sein. Dabei wird keine Callback Funktion gerufen, wenn entsprechend Ereignisse auftreten. Diese Adresse kann aber auch dieselbe sein, die bereits von anderen USB-CANmodulen verwendet wird, da die Callback Funktion das dazugehörige USB-CAN-Handle mitliefert.

Die Callback Funktion muss folgendes Format aufweisen *(siehe Kapitel 2.3.7)*:

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

Rückgabewert: Fehlercode der Funktion.

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_...

Beispiel:

```
UCANRET bRet;
tUcanHandle UcanHandle;

...
// ein USB-CANmodul ohne Callback Funktion initialisieren
bRet = UcanInitHardware (&UcanHandle, USBCAN_ANY_MODULE, NULL);
...
```

UcanInitHardwareEx

Syntax:

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

Verwendbarkeit:
DLL_INIT ab Version 3.00

Bedeutung:
Initialisiert ein USB-CANmodul als Alternative zur Funktion *UcanInitHardware()*. Im Gegensatz zur Funktion *UcanInitHardware()* wird hier ein weiterer anwenderspezifischer Parameter angegeben, der beim Aufruf der Callback Funktion mit übergeben wird.

Parameter:

pUcanHandle_p:	Zeiger auf die Variable für das USB-CAN-Handle. Dieser Zeiger darf nicht NULL sein!
bDeviceNr_p:	Gerätenummer des USB-CANmoduls (0 – 254). Der Wert *USBCAN_ANY_MODULE* (= 255) sorgt dafür, dass das USB-CANmodul verwendet wird, welches zuerst gefunden wurde.
fpCallbackFktEx_p:	Adresse auf die Callback Funktion dieses USB-CANmoduls. Dieser Wert kann NULL sein. Dabei wird keine Callback Funktion gerufen, wenn entsprechend Ereignisse auftreten. Diese Adresse kann aber auch dieselbe sein, die bereits von anderen USB-CANmodulen verwendet wird, da die Callback Funktion das dazugehörige USB-CAN-Handle mitliefert.
pCallbackArg_p:	Anwenderspezifischer Parameter, der beim Aufruf der Callback Funktion mit übergeben wird.

Die Callback Funktion muss folgendes Format aufweisen *(siehe Kapitel 2.3.7)*:

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

Rückgabewert: Fehlercode der Funktion.

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);
```

Verwendbarkeit:
HW_INIT, CAN_INIT

Bedeutung:
Deinitialisiert ein USB-CANmodul, das zuvor mit einer der beiden Funktionen *UcanInitHardware()* oder *UcanInitHardwareEx()* initialisiert wurde. Dabei gelangt die Software wieder in den Zustand DLL_INIT. Das USB-CAN-Handle ist nach dem Funktionsaufruf ungültig, so dass die für die States HW_INIT und CAN_INIT gültigen Funktionen (*siehe Tabelle 10*) nicht mehr ausgeführt werden können.

Parameter:
UcanHandle_p: USB-CAN-Handle, das mit der Funktion *UcanInitHardware()* bzw. *UcanInitHardwareEx()* erhalten wurde.

Rückgabewert: Fehlercode der Funktion.
USBCAN_SUCCESSFUL
USBCAN_ERR_MAXINSTANCES
USBCAN_ERR_ILLHANDLE
USBCAN_ERR_ILLHW

Hinweis:
Wird diese Funktion vor der Beendigung der Anwendung nicht gerufen, können andere Anwendungen später nicht auf dieses USB-CANmodul zugreifen.

UcanGetModuleTime

Syntax:

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

Verwendbarkeit:
HW_INIT, CAN_INIT ab Version 3.01

Bedeutung:
Diese Funktion liest den aktuellen Zeitstempel aus dem USB-CANmodul.

Parameter:

UcanHandle_p: USB-CAN-Handle, das mit der Funktion *UcanInitHardware()* bzw. *UcanInitHardwareEx()* erhalten wurde.

pdwTime_p: Pointer auf eine Variable, in die die Funktion den Zeitstempel ablegt.

Rückgabewert: Fehlercode der Funktion.

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_...

Hinweis:
Die Übertragung des Zeitstempels benötigt etwas Zeit. Das heißt nach Rückkehr der Funktion ist der Zeitstempel unter Umständen bereits veraltet. Wie viel Zeit zwischendurch vergangen ist hängt von vielen Kriterien ab. Deshalb können hier keine konkreten Aussagen getroffen werden.

UcanInitCan

Syntax:

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

Verwendbarkeit:
HW_INIT

Parameter:

UcanHandle_p:	USB-CAN-Handle, das mit der Funktion *UcanInitHardware()* bzw. *UcanInitHardwareEx()* erhalten wurde.
bBTR0_p:	Baudratenregister 0 (*siehe Kapitel 2.3.4*)
bBTR1_p:	Baudratenregister 1 (*siehe Kapitel 2.3.4*)
dwAMR_p:	Akzeptanzfiltermaske (*siehe Kapitel 2.3.5*)
dwACR_p:	Akzeptanzfiltercode (*siehe Kapitel 2.3.5*)

Bedeutung:
Initialisiert die CAN-Schnittstelle eines USB-CANmoduls. Die Software wechselt dabei in den Zustand CAN_INIT. Danach ist es möglich CAN-Nachrichten zu senden und zu empfangen. *Tabelle 10* zeigt die Funktionen, die in diesem Zustand möglich sind.

Rückgabewert: Fehlercode der Funktion.

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

UcanInitCanEx

Syntax:

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

Verwendbarkeit:
HW_INIT ab Version 2.16

Parameter:

UcanHandle_p: USB-CAN-Handle, das mit der Funktion *UcanInitHardware()* bzw. *UcanInitHardwareEx()* erhalten wurde.

pInitCanParam_p: Pointer auf eine Initialisierungsstruktur

```
typedef struct
{
  DWORD  m_dwSize;     // Größe dieser Struktur in Bytes
  BYTE   m_bMode;      // Modus der CAN-Übertragung
                       // (siehe folgende Tabelle)
  BYTE   m_bBTR0;      // Baudratenregister 0 des SJA1000
  BYTE   m_bBTR1;      // Baudratenregister 1 des SJA1000
  BYTE   m_bOCR;       // Output Control Register des SJA1000
                       // (sollte immer 0x1A sein)
  DWORD  m_dwAMR;      // Akzeptanzfiltermaske des SJA1000
  DWORD  m_dwACR;      // Akzeptanzfiltercode des SJA1000
  DWORD  m_dwBaudrate; // Baudratenregister für Multiport,
                       // USB-CANmodul1 und USB-CANmodul2
  WORD   m_wNrOfRxBufferEntries; // Anzahl der Einträge im
                          // Empfangspuffer der USBCAN-Library
  WORD   m_wNrOfTxBufferEntries; // Anzahl der Einträge im
                          // Sendepuffer der USBCAN-Library
} tUcanInitCanParam;
```

Hinweis:
Die Baudrateneinstellungen zwischen GW-001 bzw. GW-002 und den neuen sysWORXX-Modulen unterscheiden sich sehr stark. Für Standardwerte *(siehe Kapitel 2.3.4)* kann für die sysWORXX-Module auch BTR0 und BTR1 eingestellt werden. Dabei muss jedoch der Wert *m_dwBaudrate* auf den Wert USBCAN_BAUDEX_USE_BTR01 gesetzt werden.

Der Modus der CAN-Übertragung wird über eine 8 Bit Bitmaske eingestellt. Folgende Tabelle listet mögliche Konstanten für den Modus auf.

Konstante	Wert	Bedeutung
kUcanModeNormal	0x00	normaler Sende- und Empfangsmodus
kUcanModeListenOnly	0x01	nur Empfangsmodus; zu sendende CAN-Nachrichten erscheinen nicht auf dem CAN-Bus. CAN-Nachrichten von einem entfernten CAN-Knoten werden nicht mit einem Acknowledge bestätigt.
kUcanModeTxEcho	0x02	*UcanReadCanMsg()* bzw. *UcanReadCanMsgEx()* liefert auch die gesendeten CAN-Nachrichten als Sendeecho zurück (siehe Funktion *UcanReadCanMsg()*; nicht für GW-001 verfügbar)

Tabelle 12: Konstanten für den Modus der CAN-Übertragung

Bedeutung:
Initialisiert die CAN-Schnittstelle eines USB-CANmoduls mit erweiterten Parametern. Diese Funktion arbeitet wie die Funktion *UcanInitCan()*. Sie sollte jedoch nicht gleichzeitig mit *UcanInitCan()* gerufen werden, sondern nur als Alternative.

Rückgabewert: Fehlercode der Funktion.

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

UcanInitCanEx2

Syntax:

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

Verwendbarkeit:
HW_INIT ab Version 3.00

Parameter:

UcanHandle_p: USB-CAN-Handle, das mit der Funktion *UcanInitHardware()* bzw. *UcanInitHardwareEx()* erhalten wurde.

bChannel_p: CAN-Kanal, der initialisiert werden soll.
USBCAN_CHANNEL_CH0 für CAN-Kanal 0
USBCAN_CHANNEL_CH1 für CAN-Kanal 1

pInitCanParam_p: Pointer auf eine Initialisierungsstruktur

Definition der Struktur *tUcanInitCanParam* siehe Funktion *UcanInitCanEx().*

Bedeutung:
Initialisiert einen CAN-Kanal des USB-CANmoduls. Für GW-001 und GW-002 kann nur der CAN-Kanal 0 initialisiert werden. Diese Funktion wird als Alternative zur *UcanInitCanEx()* verwendet.

Rückgabewert: Fehlercode der Funktion.

USBCAN_SUCCESSFUL
USBCAN_ERR_MAXINSTANCES
USBCAN_ERR_ILLHANDLE
USBCAN_ERR_RESOURCE
USBCAN_ERR_BUSY
USBCAN_ERR_IOFAILED
USBCAN_ERRCMD_…
USBCAN_ERR_ILLPARAM
USBCAN_ERR_ILLHW
USBCAN_ERR_DATA

USBCAN_ERR_ABORT
USBCAN_ERR_DISCONNECT
USBCAN_ERR_TIMEOUT

UcanSetTxTimeout

Syntax:

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

Verwendbarkeit:
CAN_INIT ab Version 3.10, nur für mehrkanalige Module

Bedeutung:
Wird diese Funktion mit einem Timeout größer als 0 Millisekunden gerufen, dann wird in der Firmware eine Timeout-Überwachung für jede zu sendende CAN-Nachricht gestartet. Konnte eine CAN-Nachricht innerhalb dieses Timeouts nicht gesendet werden, dann versetzt sich die Firmware in einen speziellen Status, bei dem alle folgenden CAN-Nachrichten für diesen Kanal automatisch gelöscht werden. Bei jeder gelöschten CAN-Nachricht wird der neue CAN-Treiber Status USBCAN_CANERR_TXMSGLOST neu gesetzt. Konnte die CAN-Nachricht später doch gesendet werden, dann verlässt die Firmware diesen speziellen Status wieder.

Dieses Feature soll verhindern, dass die zu sendeden CAN-Nachrichten eines CAN-Kanals nicht die des anderen blockieren, weil dort unter Umständen kein weiterer CAN-Teilnehmer am CAN-Bus angeschlossen ist oder vielleicht ein Problem mit der physischen Anschaltung des CAN-Busses vorliegt.

Parameter:

UcanHandle_p:	USB-CAN-Handle, das mit der Funktion *UcanInitHardware()* bzw. *UcanInitHardwareEx()* erhalten wurde.
bChannel_p:	CAN Kanal, für den das Timeout gelten soll. USBCAN_CHANNEL_CH0 für CAN Kanal 0 USBCAN_CHANNEL_CH1 für CAN Kanal 1

dwTxTimeout_p: Sende-Timeout in Millisekunden. Der Wert 0 schaltet die Timeout-Überwachung wieder aus.

Rückgabewert: Fehlercode der Funktion.

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

UcanResetCan

Syntax:

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

Verwendbarkeit:
HW_INIT, CAN_INIT

Bedeutung:
Setzt den CAN-Controller im USB-CANmodul zurück und löscht den Zwischenpuffer für die CAN-Nachrichten. Diese Funktion muss gerufen werden, wenn ein BUSOFF aufgetreten ist.
Ab Version 2.17 wird ein Fehler im CAN Status (über *UcanGetStatus()* lesbar) ebenfalls gelöscht.

Parameter:

UcanHandle_p: USB-CAN-Handle, das mit der Funktion *UcanInitHardware()* bzw. *UcanInitHardwareEx()* erhalten wurde.

Rückgabewert: Fehlercode der Funktion.

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

UcanResetCanEx

Syntax:

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

Verwendbarkeit:
HW_INIT, CAN_INIT ab Version 3.00

Bedeutung:
Setzt parametrierte Features global oder für einen CAN-Kanal im USB-CANmodul zurück (siehe auch Funktion *UcanResetCan()*). Für das GW-001 werden immer alle Features der Firmware zurückgesetzt. Für das GW-001 und das GW-002 sowie das USB-CANmodul1 können nur die Features für den CAN-Kanal 0 zurückgesetzt werden.

Parameter:

UcanHandle_p: USB-CAN-Handle, das mit der Funktion *UcanInitHardware()* bzw. *UcanInitHardwareEx()* erhalten wurde.

bChannel_p: CAN-Kanal, der zurückgesetzt werden soll.
USBCAN_CHANNEL_CH0 für CAN-Kanal 0
USBCAN_CHANNEL_CH1 für CAN-Kanal 1

dwResetFlags_p: Diese Flags geben an, was zurückgesetzt werden soll, und was nicht zurückgesetzt werden soll (*siehe folgende Liste*). Diese Flags können miteinander kombiniert werden.

USBCAN_RESET_ALL 0x00000000:
Es wird alles zurückgesetzt. Die Firmware wird jedoch nicht komplett zurückgesetzt.

USBCAN_RESET_NO_STATUS 0x00000001:
Der CAN Status wird nicht zurückgesetzt. (wird nicht von den veralteten Modulen GW-001 und GW-002 unterstützt)

USBCAN_RESET_NO_CANCTRL.....0x00000002:
Der CAN-Controller wird nicht zurückgesetzt.

USBCAN_RESET_NO_TXCOUNTER 0x00000004:
Der Zähler der gesendeten CAN-Nachrichten wird nicht zurückgesetzt.

USBCAN_RESET_NO_RXCOUNTER 0x00000008:
Der Zähler der empfangenen CAN-Nachrichten wird nicht zurückgesetzt.

USBCAN_RESET_NO_TXBUFFER_CH 0x00000010:
Der Sendepuffer für einen bestimmten CAN-Kanal (angegeben im Parameter bChannel_p) wird nicht zurückgesetzt.

USBCAN_RESET_NO_TXBUFFER_DLL 0x00000020:
Der DLL-interne Sendepuffer für beide CAN-Kanäle wird nicht zurückgesetzt.

USBCAN_RESET_NO_TXBUFFER_FW 0x00000080:
Der Firmware-interne Sendepuffer für beide CAN-Kanäle wird nicht zurückgesetzt.

USBCAN_RESET_NO_RXBUFFER_CH 0x00000100:
Der Empfangspuffer für einen bestimmten CAN-Kanal (angegeben im Parameter bChannel_p) wird nicht zurückgesetzt.

USBCAN_RESET_NO_RXBUFFER_DLL 0x00000200:
Der DLL-interne Empfangspuffer für beide CAN-Kanäle wird nicht zurückgesetzt.

USBCAN_RESET_NO_RXBUFFER_SYS 0x00000400:
Der Kernel-Mode-Treiber-interne Empfangspuffer für beide CAN-Kanäle wird nicht zurückgesetzt.

USBCAN_RESET_NO_RXBUFFER_FW.....0x00000800:
Der Firmware-interne Empfangspuffer für beide CAN-Kanäle wird nicht zurückgesetzt.

USBCAN_RESET_FIRMWARE.....0xFFFFFFFF:
Die Firmware in der Hardware wird komplett zurückgesetzt.

Folgende Kombinationen sind bereits vordefiniert:

USBCAN_RESET_ONLY_STATUS.....0x0000FFFE:
Löscht nur den CAN Status.

USBCAN_RESET_ONLY_CANCTRL 0x0000FFFD:
Setzt nur den CAN-Controller im USB-CANmodul zurück. Dies muss immer nach einem Bus-off erfolgen, da der CAN-Controller nicht selbständig aus dem Bus-off Zustand zurückschalten kann.

USBCAN_RESET_ONLY_RXBUFFER_FW 0x0000F7FF:
Löscht nur den Empfangspuffer innerhalb der Firmware des USB-CANmoduls.

USBCAN_RESET_ONLY_TXBUFFER_FW..... 0x0000FF7F:
Löscht nur den Sendepuffer innerhalb der Firmware des USB-CANmoduls.

USBCAN_RESET_ONLY_RXCHANNEL_BUFF 0x0000FEFF:
Löscht nur den Empfangspuffer für einen bestimmten CAN Kanal.

USBCAN_RESET_ONLY_TXCHANNEL_BUFF 0x0000FFEF:
Löscht nur den Sendepuffer für einen bestimmten CAN Kanal.

USBCAN_RESET_ONLY_RX_BUFF 0x0000F0F7:
Löscht nur die Empfangspuffer in allen Software-Schichten und setzt den Empfangszähler zurück.

USBCAN_RESET_ONLY_TX_BUFF.....0x0000FF0B:
Löscht nur die Sendepuffer in allen Software-Schichten und setzt den Sendezähler zurück.

USBCAN_RESET_ONLY_ALL_BUFF 0x0000F003:
Löscht nur alle Puffer in allen Software-Schichten und setzt den Sendezähler und den Empfangszähler zurück.

USBCAN_RESET_ONLY_ALL_COUNTER 0x0000FFF3:
Löscht nur die Sendezähler und Empfangszähler.

ACHTUNG:
Wenn die Konstanten USBCAN_RESET_NO_... miteinander kombiniert werden sollen, dann muss eine logische ODER-Verknüpfung verwendet werden.

Beispiel:
```
dwFalgs =  USBCAN_RESET_NO_COUNTER_ALL  |
           USBCAN_RESET_NO_BUFFER_ALL;
```

Wenn jedoch die Konstanten USBCAN_RESET_ONLY_... miteinander kombiniert werden sollen, dann muss eine logische UND-Verknüpfung verwendet werden.

Beispiel:
```
dwFalgs =  USBCAN_RESET_ONLY_RX_BUFF  &
           USBCAN_RESET_ONLY_STATUS;
```

Für das GW-002 muss die Konstante USBCAN_RESET_ONLY_RX_BUFF_GW002 statt USBCAN_RESET_ONLY_RX_BUFF verwendet werden. Dabei wird in der Firmware auch der Sendepuffer gelöscht.

Rückgabewert: Fehlercode der Funktion.

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);
```

Verwendbarkeit:
CAN_INIT

Bedeutung:
Deinitialisiert die CAN-Schnittstelle eines USB-CANmoduls. Bei GW-001 und GW-002 wird dabei die Betriebsspannung des CAN-Controllers auf 0 V geschaltet. Alle CAN-Nachrichten, die nach diesem Funktionsaufruf vom CAN-Bus empfangen werden, werden nicht mehr zum PC übertragen.

Parameter:
UcanHandle_p: USB-CAN-Handle, das mit der Funktion *UcanInitHardware()* bzw. *UcanInitHardwareEx()* erhalten wurde.

Rückgabewert: Fehlercode der Funktion.
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

UcanDeinitCanEx

Syntax:

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

Verwendbarkeit:
CAN_INIT

Bedeutung:
Deinitialisiert die CAN-Schnittstelle eines USB-CANmoduls. Bei GW-001 und GW-002 wird dabei die Betriebsspannung des CAN-Controllers auf 0 V geschaltet. Alle CAN-Nachrichten, die nach diesem Funktionsaufruf vom CAN-Bus empfangen werden, werden nicht mehr zum PC übertragen.

Parameter:

UcanHandle_p:	USB-CAN-Handle, das mit der Funktion *UcanInitHardware()* bzw. *UcanInitHardwareEx()* erhalten wurde.
bChannel_p:	CAN-Kanal, der deinitialisiert werden soll. USBCAN_CHANNEL_CH0 für CAN-Kanal 0 USBCAN_CHANNEL_CH1 für CAN-Kanal 1

Rückgabewert: Fehlercode der Funktion.

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

UcanGetHardwareInfo

Syntax:

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

Verwendbarkeit:
HW_INIT, CAN_INIT

Bedeutung:
Gibt die Hardwareinformation eines USB-CANmoduls zurück. Diese Funktion ist sehr nützlich, wenn ein USB-CANmodul mit der Gerätenummer *USBCAN_ANY_MODULE* initialisiert wurde. Die Hardwareinformation enthält danach die Gerätenummer des initialisierten USB-CANmoduls.

Parameter:

UcanHandle_p: USB-CAN-Handle, das mit der Funktion *UcanInitHardware()* bzw. *UcanInitHardwareEx()* erhalten wurde.

pHwInfo_p: Adresse auf die Hardwareinformationsstruktur. (siehe folgende Seite)

Rückgabewert: Fehlercode der Funktion.

USBCAN_SUCCESSFUL
USBCAN_ERR_MAXINSTANCES
USBCAN_ERR_ILLHANDLE
USBCAN_ERR_ILLPARAM
USBCAN_ERR_ILLHW

```
typedef struct
{
    BYTE         m_bDeviceNr;  // Gerätenummer
    tUcanHandle  m_UcanHandle; // USB-CAN-Handle
    DWORD        m_dwReserved; // reserviert
    BYTE         m_bBTR0;      // Baudratenregister 0
    BYTE         m_bBTR1;      // Baudratenregister 1
    BYTE         m_bOCR;       // Output-Control-Register
    DWORD        m_dwAMR;      // Akzeptanzfiltermaske
    DWORD        m_dwACR;      // Akzeptanzfiltercode
    BYTE         m_bMode;      // Modus des CAN Controllers
                               // (siehe tUcanMode)
    DWORD        m_dwSerialNr; // Seriennummer des
                               // USB-CANmoduls
} tUcanHardwareInfo;
```

Hinweis:
Die Parameter *m_bMode* und *m_dwSerialNr* sind erst ab der Softwareversion 2.16 verfügbar.

Beispiel:

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

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

// kein Fehler?
if (bRet == USBCAN_SUCCESSFUL)
{
    // Hardwareinformation holen
    UcanGetHardwareInfo (UcanHandle, &HwInfo);

    // Gerätenummer in einen String wandeln
    _stprintf (szDeviceNr, _T(„Gerätenummer = %d“),
        HwInfo.m_bDeviceNr);
...
}
```


UcanGetHardwareInfoEx2

Syntax:

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

Verwendbarkeit:
HW_INIT, CAN_INIT ab Version 3.00

Bedeutung:
Gibt die erweiterten Hardwareinformationen eines USB-CANmoduls zurück. Für das Multiport CAN-to-USB 3004006 und das USB-CANmodul2 3204002/3204003 stehen zwei CAN-Kanäle je logisches Gerät zur Verfügung, wobei die Informationen jedes CAN-Kanals separat zurückgegeben werden.

Parameter:

UcanHandle_p: USB-CAN-Handle, das mit der Funktion *UcanInitHardware()* bzw. *UcanInitHardwareEx()* erhalten wurde.

pHwInfoEx_p: Adresse auf die erweiterte Hardware-informationsstruktur (siehe folgende Seite).

pCanInfoCh0_p: Adresse auf die Informationsstruktur für CAN-Kanal 0. Dieser Parameter kann NULL sein.

pCanInfoCh1_p: Adresse auf die Informationsstruktur für CAN-Kanal 1. Dieser Parameter kann NULL sein.

Rückgabewert: Fehlercode der Funktion.

USBCAN_SUCCESSFUL
USBCAN_ERR_MAXINSTANCES
USBCAN_ERR_ILLHANDLE
USBCAN_ERR_ILLPARAM
USBCAN_ERR_ILLHW

```
typedef struct
{
    DWORD       m_dwSize;           // Anzahl der Bytes dieser
                                    // Struktur
    tUcanHandle m_UcanHandle;       // USB-CAN-Handle
    BYTE        m_bDeviceNr;        // Gerätenummer
    DWORD       m_dwSerialNr;       // Seriennummer
    DWORD       m_dwFwVersionEx;    // Firmware Version
    DWORD       m_dwProductCode;    // Hardware-Typ

} tUcanHardwareInfoEx;

typedef struct
{
  DWORD    m_dwSize;        // Anzahl der Bytes dieser
                            // Struktur
  BYTE     m_bMode;         // Modus der CAN-Übertragung
  BYTE     m_bBTR0;         // Bus Timing Register 0
  BYTE     m_bBTR1;         // Bus Timing Register 1
  BYTE     m_bOCR;          // Output Controll Register
  DWORD    m_dwAMR;         // Acceptance Mask Register
  DWORD    m_dwACR;         // Acceptance Code Register
  DWORD    m_dwBaudrate;    // Baudrate Register für Multiport,
                            // USB-CANmodul1 und USB-CANmodul2
  BOOL     m_fCanIsInit;    // ist TRUE wenn CAN-Kanal
                            // initialisiert wurde
  WORD     m_wCanStatus;    // letzter CAN Status
                            // (siehe UcanGetStatus())
} tUcanChannelInfo;
```

Verwenden Sie gegebenenfalls die folgenden Makros, um die Unterstützung von verschiedenen Features abzufragen:

USBCAN_CHECK_SUPPORT_CYCLIC_MSG(pHwIndoEx)
: Dieses Makro prüft, ob das USB-CANmodul das automatische Senden von CAN-Nachrichten unterstützt.

USBCAN_CHECK_SUPPORT_TWO_CHANNEL(pHwIndoEx)
: Dieses Makro prüft, ob das USB-CANmodul zwei CAN-Kanäle je logisches Modul unterstützt.

USBCAN_CHECK_SUPPORT_TERM_RESISTOR(pHwIndoEx)
: Dieses Makro prüft, ob das USB-CANmodul einen zurücklesbaren Abschlusswiderstand unterstützt.

USBCAN_CHECK_SUPPORT_USER_PORT(pHwIndoEx)

Dieses Makro prüft, ob das USB-CANmodul eine programmierbare Port Ereiterung unterstützt (*siehe Kapitel 1.5*).

USBCAN_CHECK_SUPPORT_RBUSER_PORT(pHwIndoEx)

Dieses Makro prüft, ob das USB-CANmodul eine programmierbare Port Erweiterung unterstützt, bei dem der zuletzt geschriebene Wert der Konfiguration und der Ausgänge nach dem nächsten PowerOn wieder automatisch eingenommen wird.

USBCAN_CHECK_SUPPORT_RBCAN_PORT(pHwIndoEx)

Dieses Makro prüft, ob das USB-CANmodul einen programmierbaren CAN-Port (für lowspeed CAN-Transceiver) unterstützt, bei dem der zuletzt geschriebene Wert der Konfiguration nach dem nächsten PowerOn wieder automatisch eingenommen wird.

Beispiel:

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

...
// USB-CANmodul initialisieren
bRet = UcanInitHardware (&UcanHandle, USBCAN_ANY_MODULE, NULL);
if (bRet == USBCAN_SUCCESSFUL)
{
    // erweiterte Hardwareinformationen holen
    bRet = UcanGetHardwareInfoEx2 (UcanHandle, &HwInfoEx,
        NULL, NULL);
    if (bRet == USBCAN_SUCCESSFUL)
    {
        // prüfen, ob zwei CAN-Kanäle zur Verfügung stehen
        if (USBCAN_CHECK_SUPPORT_TWO_CHANNEL (&HwInfoEx))
        {
            ...
        }
        ...
    }
    ...
}
```

UcanGetMsgCountInfo

Syntax:

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

Verwendbarkeit:
CAN_INIT ab Version 3.00

Bedeutung:
Liest die Zählerstände für gesendete bzw. empfangene CAN-Nachrichten aus.

Parameter:

UcanHandle_p: USB-CAN-Handle, das mit der Funktion *UcanInitHardware()* bzw. *UcanInitHardwareEx()* erhalten wurde.

pMsgCountInfo_p: Pointer auf eine Struktur *tUcanMsgCountInfo* mit den Zählerständen

Rückgabewert: Fehlercode der Funktion.

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; // Zähler der gesendeten CAN-Nachrichten
  WORD m_wRecvdMsgCount;// Zähler der empfangenen CAN-Nachrichten

} tUcanMsgCountInfo;
```

UcanGetMsgCountInfoEx

Syntax:

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

Verwendbarkeit:
CAN_INIT ab Version 2.16

Bedeutung:
Liest die Zählerstände für gesendete bzw. empfangene CAN-Nachrichten eines bestimmten CAN-Kanals aus.

Parameter:

UcanHandle_p:	USB-CAN-Handle, das mit der Funktion *UcanInitHardware()* bzw. *UcanInitHardwareEx()* erhalten wurde.
bChannel_p:	CAN-Kanal, dessen Zähler ausgelesen werden sollen. USBCAN_CHANNEL_CH0 für CAN-Kanal 0 USBCAN_CHANNEL_CH1 für CAN-Kanal 1
pMsgCountInfo_p:	Pointer auf eine Struktur *tUcanMsgCountInfo* mit den Zählerständen

Rückgabewert: Fehlercode der Funktion.

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

Die Struktur *tUcanMsgCountInfo* ist bei der Funktion *UcanGetMsgCountInfo()* beschrieben.

UcanGetStatus

Syntax:

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

Verwendbarkeit:
HW_INIT, CAN_INIT

Bedeutung:
Gibt den Fehlerstatus aus dem USB-CANmodul zurück. Tritt im USB-CANmodul ein Fehler auf, so beginnt die rote LED zu blinken und eine Statusinformation wird zum PC gesendet. Wurde der Funktion *UcanInitHardware()* bzw. *UcanInitHardwareEx()* eine Callback Funktion übergeben, so wird zusätzlich diese Callback Funktion mit dem Ereignis *USBCAN_EVENT_STATUS* gerufen. Nach dem Aufruf der Funktion *UcanGetStatus()* wird der Fehlerzustand im USB-CANmodul gelöscht, und die rote LED hört auf zu blinken.
Ab Version 2.17 muss ein Fehler im CAN Status durch Aufruf von *UcanResetCan()* gelöscht werden.

Parameter:

UcanHandle_p:	USB-CAN-Handle, das mit der Funktion *UcanInitHardware()* bzw. *UcanInitHardwareEx()* erhalten wurde.
pStatus_p:	Fehlerstatus des USB-CANmoduls.

```
typedef struct
{
    WORD    m_wCanStatus;    // aktueller CAN-Status
    WORD    m_wUsbStatus;    // aktueller USB-Status

} tStatusStruct;
```

Rückgabewert: Fehlercode der Funktion.

USBCAN_SUCCESSFUL
USBCAN_ERR_MAXINSTANCES
USBCAN_ERR_ILLHANDLE
USBCAN_ERR_ILLPARAM
USBCAN_ERR_ILLHW

Das WORD *m_wCanStatus* in der Struktur *tStatusStruct* kann folgende Werte annehmen:

USBCAN_CANERR_OK = 0x0000
kein Fehler

USBCAN_CANERR_XMTFULL = 0x0001
Sendepuffer im CAN-Controller übergelaufen

USBCAN_CANERR_OVERRUN = 0x0002
Empfangspuffer im CAN-Controller übergelaufen

USBCAN_CANERR_BUSLIGHT = 0x0004
Fehlergrenze 1 im CAN-Controller überschritten Der CAN-Controller befindet sich im „Warning Limit“ Status.

USBCAN_CANERR_BUSHEAVY = 0x0008
Fehlergrenze 2 im CAN-Controller überschritten Der CAN-Controller befindet sich im „Error Passive“ Status.

USBCAN_CANERR_BUSOFF = 0x0010
CAN-Controller ist im BUSOFF-Status

USBCAN_CANERR_QOVERRUN = 0x0040
Empfangspuffer im Modul übergelaufen

USBCAN_CANERR_QXMTFULL = 0x0080
Sendepuffer im Modul übergelaufen

USBCAN_CANERR_REGTEST = 0x0100
CAN-Controller nicht gefunden (Hardwarefehler)

USBCAN_CANERR_TXMSGLOST = 0x0400
Eine zu sendende CAN-Nachricht wurde auf Grund eines Sende-Timeouts (*siehe auch Funktion UcanSetTxTimeout()*) von der Firmware gelöscht.

Dieses WORD ist Bit-orientiert. Es können also mehrere Fehler gleichzeitig gemeldet werden.

Das WORD *m_wUsbStatus* ist veraltet und ist aus Kompatibilitätsgründen noch enthalten. Es erhält immer den Wert 0.

UcanGetStatusEx

Syntax:

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

Verwendbarkeit:
HW_INIT, CAN_INIT ab Version 3.00

Bedeutung:
Gibt den Fehlerstatus eines bestimmten CAN-Kanals aus dem USB-CANmodul zurück. Diese Funktion wird als Alternative zur Funktion *UcanGetStatus()* verwendet.

Die Definition der Struktur *tStatusStruct* wird bei der Funktion *UcanGetStatus()* angegeben.

Parameter:

UcanHandle_p:	USB-CAN-Handle, das mit der Funktion *UcanInitHardware()* bzw. *UcanInitHardwareEx()* erhalten wurde.
bChannel_p:	CAN-Kanal, dessen Status gelesen werden soll. USBCAN_CHANNEL_CH0 für CAN-Kanal 0 USBCAN_CHANNEL_CH1 für CAN-Kanal 1
pStatus_p:	Fehlerstatus des USB-CANmoduls.

Rückgabewert: Fehlercode der Funktion.

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);
```

Verwendbarkeit:
CAN_INIT

Bedeutung:
Ändert nachträglich die Baudrateneinstellung des USB-CANmoduls.

Parameter:

UcanHandle_p: USB-CAN-Handle, das mit der Funktion *UcanInitHardware()* bzw. *UcanInitHardwareEx()* erhalten wurde.

bBTR0_p: Baudratenregister 0 (*siehe Kapitel 2.3.4*)

bBTR1_p: Baudratenregister 1 (*siehe Kapitel 2.3.4*)

Rückgabewert: Fehlercode der Funktion.

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

UcanSetBaudrateEx

Syntax:

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

Verwendbarkeit:
CAN_INIT ab Version 3.00

Bedeutung:
Ändert nachträglich die Baudrateneinstellung eines bestimmten CAN-Kanals des USB-CANmoduls. Diese Funktion wird als Alternative zur Funktion *UcanSetBaudrate()* verwendet.

Parameter:

UcanHandle_p:	USB-CAN-Handle, das mit der Funktion *UcanInitHardware()* bzw. *UcanInitHardwareEx()* erhalten wurde.
bChannel_p:	CAN-Kanal, dessen Baudrate gesetzt werden soll. USBCAN_CHANNEL_CH0 für CAN-Kanal 0 USBCAN_CHANNEL_CH1 für CAN-Kanal 1
bBTR0_p:	Baudratenregister BTR0 (*siehe Kapitel 2.3.4*)
bBTR1_p:	Baudratenregister BTR1 (*siehe Kapitel 2.3.4*)
dwBaudrate_p:	Baudraten Register für alle sysWORXX-Module (*siehe Kapitel 2.3.4*)

Hinweis:
Die Baudrateneinstellungen zwischen GW-001 bzw. GW-002 und den neuen sysWORXX-Modulen unterscheiden sich sehr stark. Für Standardwerte *(siehe Kapitel 2.3.4)* kann für die sysWORXX-Module auch BTR0 und BTR1 eingestellt werden. Dabei muss jedoch der Wert *m_dwBaudrate* auf den Wert USBCAN_BAUDEX_USE_BTR01 gesetzt werden.

Rückgabewert: Fehlercode der Funktion.

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

UcanSetAcceptance

Syntax:

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

Verwendbarkeit:
CAN_INIT

Bedeutung:
Ändert nachträglich die Akzeptanzfilterwerte des USB-CANmoduls.

Parameter:

UcanHandle_p: USB-CAN-Handle, das mit der Funktion *UcanInitHardware()* bzw. *UcanInitHardwareEx()* erhalten wurde.

dwAMR_p: Akzeptanzfiltermaske (*siehe Kapitel 2.3.5*)

dwACR_p: Akzeptanzfiltercode (*siehe Kapitel 2.3.5*)

Rückgabewert: Fehlercode der Funktion.

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

UcanSetAcceptanceEx

Syntax:

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

Verwendbarkeit:
CAN_INIT ab Version 3.00

Bedeutung:
Ändert nachträglich die Akzeptanzfilterwerte für einen bestimmten CAN-Kanal des USB-CANmoduls. Diese Funktion wird als Alternative zur Funktion *UcanSetAcceptance()* verwendet.

Parameter:

UcanHandle_p:	USB-CAN-Handle, das mit der Funktion *UcanInitHardware()* bzw. *UcanInitHardwareEx()* erhalten wurde.
bChannel_p:	CAN-Kanal, dessen Filterwerte gesetzt werden soll. USBCAN_CHANNEL_CH0 für CAN-Kanal 0 USBCAN_CHANNEL_CH1 für CAN-Kanal 1
dwAMR_p:	Akzeptanzfiltermaske (*siehe Kapitel 2.3.5*)
dwACR_p:	Akzeptanzfiltercode (*siehe Kapitel 2.3.5*)

Rückgabewert: Fehlercode der Funktion.

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

UcanReadCanMsg

Syntax:

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

Verwendbarkeit:
CAN_INIT

Bedeutung:
Liest eine CAN-Nachricht aus dem Zwischenpuffer. Diese Funktion gibt eine Warnung zurück, wenn keine CAN-Nachrichten im Zwischenpuffer sind. Ist der Zwischenpuffer übergelaufen, gibt diese Funktion eine gültige CAN-Nachricht und eine Warnung zurück.

Parameter:

UcanHandle_p:	USB-CAN-Handle, das mit der Funktion *UcanInitHardware()* bzw. *UcanInitHardwareEx()* erhalten wurde.
pCanMsg_p:	Adresse auf eine CAN-Nachrichtenstruktur. Diese Adresse darf nicht NULL sein.

```
typedef struct
{
    DWORD   m_dwID;     // CAN-Identifier
    BYTE    m_bFF;      // CAN-Frameformat
    BYTE    m_bDLC;     // CAN-Datenlängencode
    BYTE    m_bData[8]; // CAN-Daten
    DWORD   m_dwTime;   // Empfangszeit in ms

} tCanMsgStruct;
```

Das CAN-Frameformat ist eine Bitmaske, welches das Format der CAN-Nachricht angibt. Die folgende Tabelle listet mögliche Werte der Bitmaske auf.

Konstante	Wert	Bedeutung
USBCAN_MSG_FF_STD	0x00	CAN2.0A Nachricht mit 11 Bit CAN-ID
USBCAN_MSG_FF_ECHO	0x20	Sendeecho; Wird nur empfangen, wenn bei der Initialisierung der Modus kUcanModeTxEcho eingeschaltet wurde.
USBCAN_MSG_FF_RTR	0x40	Remote Frame zur Abfrage von Daten eines entfernten CAN-Knotens
USBCAN_MSG_FF_EXT	0x80	CAN2.0B Nachricht mit 29 Bit CAN-ID

Tabelle 13: Konstanten für das CAN-Frameformat

Rückgabewert: Fehlercode der Funktion.

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

Beispiel:

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

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

    // kein Fehler? dann CAN-Nachricht ausgeben
    if (USBCAN_CHECK_VALID_RXCANMSG (bRet))
    {
        PrintCanMsg (&CanMsg);
        if (USBCAN_CHECK_WARNING (bRet))
        {
            PrintWarning (bRet);
        }
    }

    // keine Warnung? dann Fehler ausgeben
    else if (USBCAN_CHECK_ERROR (bRet))
    {
        PrintError (bRet);
        break;
    }
    else
    {
        break;
    }
}
...
```

Hinweis:
Es wird empfohlen, die Funktion *UcanReadCanMsg()* in einer Schleife gerufen wird, so lange sie eine gültige CAN-Nachricht zurückgibt (bzw. ein Fehler auftritt). Dadurch wird verhindert, dass CAN-Nachrichten längere Zeit im Empfangspuffer verweilen und unter Umständen ein Empfangspufferüberlauf auftritt.

Wird eine Warnung zurückgegeben (außer USBCAN_WARN_NODATA), dann ist dennoch eine gültige CAN-Nachricht gelesen worden. Verwenden Sie gegebenenfalls das Makro *USBCAN_CHECK_VALID_RXCANMSG()*, um über den Rückgabewert zu prüfen, ob eine gültige CAN-Nachricht empfangen wurde (siehe Beispiel).

Die Variable *m_dwTime* aus der Struktur *tCanMsgStruct* enthält einen 32 Bit Zeitstempel, bei dem jedoch nur 24 Bit gültig sind. Dies liegt daran, dass dieser Zeitstempel auf der Hardware generiert wird. Aus Gründen der Kompatibilität wurde dies auch bei den neuen sysWORXX USB-CANmodulen so implementiert. Dieser Umstand muss beachtet werden, wenn man aus diesem Zeitstempel eine Zeitdifferenzen berechnen muss. In diesem Fall empfehlen wir die Verwendung des Makros USBCAN_CALC_TIMEDIFF().

Beispiel:

```
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);
```

Verwendbarkeit:
CAN_INIT ab Version 3.00

Bedeutung:
Liest eine oder mehrere CAN-Nachrichten aus dem Zwischenpuffer. Diese Funktion wird als Alternative für die Funktion *UcanReadCanMsg()* verwendet. Es können CAN-Nachrichten von einem bestimmten CAN-Kanal gelesen werden.

Parameter:

UcanHandle_p: USB-CAN-Handle, das mit der Funktion *UcanInitHardware()* bzw. *UcanInitHardwareEx()* erhalten wurde.

pbChannel_p: Adresse auf eine Variable mit dem CAN-Kanal von der die CAN-Nachrichten gelesen werden sollen.
USBCAN_CHANNEL_CH0 für CAN-Kanal 0
USBCAN_CHANNEL_CH1 für CAN-Kanal 1
Wird USBCAN_CHANNEL_ANY angegeben, dann schreibt diese Funktion die Kanalnummer auf diese Variable, von der CAN-Nachrichten empfangen wurden.

pCanMsg_p: Adresse auf ein Array aus CAN-Nachrichtenstrukturen. Diese Adresse darf nicht NULL sein.

pdwCount_p: Adresse auf eine Variable mit der Anzahl von CAN-Nachrichten, die maximal gelesen werden sollen. Die Funktion schreibt nach dem Lesen die Anzahl der CAN-Nachrichten auf diese Variable, die tatsächlich gelesen wurde. Ist dieser Parameter NULL, dann wird genau eine CAN-Nachricht gelesen.

Die Definition der Struktur *tCanMsgStruct* wird bei der Funktion *UcanReagCanMsg()* beschrieben.

Rückgabewert: Fehlercode der Funktion.

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

Beispiel:

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

...
while (1)
{
    // bis zu 16 CAN-Nachrichten lesen
    bChannel = USBCAN_CHANNEL_ANY;
    dwCount  = sizeof (RxCanMsg) / sizeof (tCabMsgStruct);
    bRet = UcanReadCanMsgEx (UcanHandle, &bChannel,
        &RxCanMsg, &dwCount);

    // kein Fehler? dann CAN-Nachricht ausgeben
    if (USBCAN_CHECK_VALID_RXCANMSG (bRet))
    {
        PrintCanMessages (&RxCanMsg[0], dwCount);
        if (USBCAN_CHECK_WARNING (bRet))
        {
            PrintWarning (bRet);
        }
    }
    // keine Warnung? dann Fehler ausgeben
    else if (USBCAN_CHECK_ERROR (bRet))
    {
        PrintError (bRet);
        break;
    }
    else
    {
        break;
    }
}
...
```

Hinweis:
Es wird empfohlen, dass die Funktion *UcanReadCanMsgEx()* in einer Schleife gerufen wird, so lange sie eine (oder mehrere) gültige CAN-Nachricht(en) zurückgibt (bzw. ein Fehler auftritt). Dadurch wird verhindert, dass CAN-Nachrichten längere Zeit im Empfangspuffer verweilen und unter Umständen ein Empfangspufferüberlauf auftritt.

Wird eine Warnung zurückgegeben (außer USBCAN_WARN_NODATA), dann sind dennoch gültige CAN-Nachrichten gelesen worden. Verwenden Sie gegebenenfalls das Makro *USBCAN_CHECK_VALID_RXCANMSG()*, um über den Rückgabewert zu prüfen, ob eine gültige CAN-Nachricht empfangen wurde (siehe Beispiel).

Ab der Softwareversion 3.05 kann die maximale Anzahl der CAN-Nachrichten im Empfangspuffer parametriert werden (*siehe Funktion UcanInitCanEx() und Struktur tUcanInitCanParam*).

Die Variable *m_dwTime* aus der Struktur *tCanMsgStruct* enthält einen 32 Bit Zeitstempel, bei dem jedoch nur 24 Bit gültig sind. Dies liegt daran, dass dieser Zeitstempel auf der Hardware generiert wird. Aus Gründen der Kompatibilität wurde dies auch bei den neuen sysWORXX USB-CANmodulen so implementiert. Dieser Umstand muss beachtet werden, wenn man aus diesem Zeitstempel eine Zeitdifferenzen berechnen muss. In diesem Fall empfehlen wir die Verwendung des Makros USBCAN_CALC_TIMEDIFF().

Beispiel:

```
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);
```

Verwendbarkeit:
CAN_INIT

Bedeutung:
Sendet eine CAN-Nachricht über das USB-CANmodul.

Parameter:

UcanHandle_p: USB-CAN-Handle, das mit der Funktion *UcanInitHardware()* bzw. *UcanInitHardwareEx()* erhalten wurde.

pCanMsg_p: Adresse auf eine CAN-Nachrichtenstruktur. Diese Adresse darf nicht NULL sein.

```
typedef struct
{
    DWORD   m_dwID;     // CAN-Identifier
    BYTE    m_bFF;      // CAN-Frameformat
    BYTE    m_bDLC;     // CAN-Datenlängencode
    BYTE    m_bData[8]; // CAN-Daten
    DWORD   m_dwTime;   // hat hier keine Bedeutung

} tCanMsgStruct;
```

Die Bedeutung des CAN-Frameformates wird bei der Funktion *UcanReagCanMsg()* beschrieben. Für das Senden von CAN-Nachrichten hat das Bit USBCAN_MSG_FF_ECHO keine Bedeutung.

Rückgabewert: Fehlercode der Funktion.

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);
```

Verwendbarkeit:
CAN_INIT ab Version 3.00

Bedeutung:
Sendet eine oder mehrere CAN-Nachrichten über das USB-CANmodul. Diese Funktion wird als Alternative zur Funktion *UcanWriteCanMsg()* gerufen. Sie sendet CAN-Nachrichten über einen bestimmten CAN-Kanal des USB-CANmoduls.

Parameter:

UcanHandle_p:	USB-CAN-Handle, das mit der Funktion *UcanInitHardware()* bzw. *UcanInitHardwareEx()* erhalten wurde.
bChannel_p:	CAN-Kanal, über den gesendet werden soll. USBCAN_CHANNEL_CH0 für CAN-Kanal 0 USBCAN_CHANNEL_CH1 für CAN-Kanal 1
pCanMsg_p:	Adresse auf ein Array von CAN-Nachrichtenstrukturen für den Empfang der CAN-Nachrichten. Diese Adresse darf nicht NULL sein.
pdwCount_p:	Adresse auf eine Variable, die die Anzahl der zu sendenden CAN-Nachrichten enthält. Die Funktion schreibt nach dem Senden die Anzahl der tatsächlich gesendeten CAN-Nachrichten auf diese Variable. Ist dieser Parameter NULL, dann wird nur eine CAN-Nachricht gesendet

Die Definition der Struktur *tCanMsgStruct* wird bei der Funktion *UcanWriteCanMsg()* beschrieben.

Rückgabewert: Fehlercode der Funktion.

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

Hinweis:
Wurde diese Funktion gerufen, um mehr als eine CAN-Nachricht zu senden, dann müssen Sie auch auf die Warnung USBCAN_WARN_TXLIMIT prüfen. Wird dieser Fehlercode zurückgegeben, dann wurde nur ein Teil der zu sendenden CAN-Nachrichten in den Sendepuffer geschrieben. Die Anzahl der CAN-Nachrichten von diesem Teil wird an die durch den Parameter pdwCount_p adressierte Variable zurückgeschrieben. Der nicht gesendete Teil muss erneut versucht werden zu senden, sonst gehen diese CAN-Nachrichten auf Applikationsebene verloren.

Verwenden Sie gegebenenfalls das Makro *USBCAN_CHECK_TX_NOTALL()*, um über den Rückgabewert zu prüfen, ob einige CAN-Nachrichten nicht gesendet werden konnten (siehe Beispiel). Das Makro *USBCAN_CHECK_TX_SUCCESS()* prüft, ob alle CAN-Nachrichten erfolgreich gesendet werden konnten. Mit dem Makro *USBCAN_CHECK_TX_OK()* prüfen Sie, ob mindestens eine CAN-Nachricht erfolgreich gesendet werden konnte.

Beispiel:

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

...
{
    // bis zu 10 CAN-Nachrichten senden
    dwCount  = sizeof (TxCanMsg) / sizeof (tCabMsgStruct);
    bRet = UcanWriteCanMsgEx (UcanHandle, USBCAN_CHANNEL_CH0,
        &TxCanMsg, &dwCount);

    // kein Fehler?
    if (USBCAN_CHECK_TX_OK (bRet))
    {
        // wurden nicht alle CAN-Nachrichten gesendet?
        if (USBCAN_CHECK_TX_NOTALL (bRet))
        {
            ...
        }
        // eine andere Warnung aufgetreten?
        else if (USBCAN_CHECK_WARNING (bRet))
        {
            PrintWarning (bRet);
        }
    }
    // keine Warnung? dann Fehler ausgeben
    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);
```

Verwendbarkeit:
CAN_INIT ab Version 3.06, ab sysWORXX

Bedeutung:
Diese Funktion ermittelt die Anzahl der CAN-Nachrichten, die sich noch in den Puffern der einzelnen Software-Schichten befinden. Mit dem Parameter dwFlags_p wird angegeben, welche Puffer überprüft werden sollen. Sollen mehrere Puffer geprüft werden, dann werden die Anzahl der enthaltenen CAN-Nachrichten miteinander addiert und an die durch pdwCount_p adressierte Variable geschrieben.

Parameter:

UcanHandle_p:	USB-CAN-Handle, das mit der Funktion *UcanInitHardware()* bzw. *UcanInitHardwareEx()* erhalten wurde.
bChannel_p:	CAN-Kanal USBCAN_CHANNEL_CH0 für CAN-Kanal 0 USBCAN_CHANNEL_CH1 für CAN-Kanal 1 USBCAN_CHANNEL_ANY für beide Kanäle
dwFlags_p:	Gibt an, welche Puffer überprüft werden sollen (*siehe Tabelle 14*). Die einzelnen Flags können miteinander kombiniert werden.
pdwCount_p:	Adresse auf eine Variable, die die Anzahl der CAN-Nachrichten im Puffer empfangen soll. Dieser Parameter darf nicht NULL sein.

Konstante **USBCAN_PENDING...**	**Wert**	**Bedeutung**
..._FLAG_RX_DLL	0x00000001	Prüft die Anzahl der Nachrichten im Empfangspuffer der DLL.
..._FLAG_RX_FW	0x00000004	Prüft die Anzahl der Nachrichten im Empfangspuffer der Firmware
..._FLAG_TX_DLL	0x00000010	Prüft die Anzahl der Nachrichten im Sendepuffer der DLL.
..._FLAG_TX_FW	0x00000040	Prüft die Anzahl der Nachrichten im Sendepuffer der Firmware

Tabelle 14: Flags für die Funktion UcanGetMsgPending()

Rückgabewert: Fehlercode der Funktion.

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_…

Hinweis:
Nach dem Aufruf der Funktion *UcanGetMsgPending()* können sich die Zählerstände der Puffer in den einzelnen Softwareschichten bereits geändert haben. Wenn diese Funktion all zu oft gerufen wird, dann kann dies die Performance beeinflussen.

UcanGetCanErrorCounter

Syntax:

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

Verwendbarkeit:
CAN_INIT ab Version 3.06, ab sysWORXX

Bedeutung:
Liest die aktuellen Zählerstände der Fehlerzähler im CAN-Controller aus. Die Werte werden direkt von der Hardware gelesen.

Parameter:

UcanHandle_p: USB-CAN-Handle, das mit der Funktion *UcanInitHardware()* bzw. *UcanInitHardwareEx()* erhalten wurde.

bChannel_p: CAN-Kanal
USBCAN_CHANNEL_CH0 für CAN-Kanal 0
USBCAN_CHANNEL_CH1 für CAN-Kanal 1

pdwTxCount_p: Adresse auf eine DWORD-Variable, die den Fehlerzähler für Sendenachrichten empfangen soll. Dieser Parameter darf nicht NULL sein.

pdwRxCount_p: Adresse auf eine DWORD-Variable, die den Fehlerzähler für Empfangsnachrichten empfangen soll. Dieser Parameter darf nicht NULL sein.

Rückgabewert: Fehlercode der Funktion.

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 Funktionen für das automatische Senden

Die folgenden Funktionen können nur mit den neuen Hardware-Derivaten verwendet werden (nicht für GW-001 und GW-002). Sie dienen zum automatischen Senden von Layer 2 CAN-Nachrichten mit einer genaueren Zykluszeit, wie sie von einer PC-Applikation nicht erreicht werden kann.

Hinweis:
Die Genauigkeit der Zykluszeit hängt auch von der verwendeten CAN-Baudrate ab. Verwenden Sie zum Beispiel 10 kBit/s, dann kann die Zykluszeit um ca. 10 Millisekunden schwanken.

Es können maximal 16 CAN-Nachrichten für das automatische Senden in einem USB-CANmodul definiert werden. Für das Senden stehen zwei Modi zur Verfügung. Der erste Modus wird „paralleler Modus“ bezeichnet, wobei die Zykluszeit aller bis zu 16 CAN-Nachrichten überprüft wird. Ist eine dieser Zykluszeiten abgelaufen, dann die entsprechende CAN-Nachricht gesendet. Die Zykluszeit wird beim „parallelen Modus“ für jede CAN-Nachricht einzeln bezüglich ihrer letzten Sendezeit überwacht (*siehe Bild 15*). Beim zweiten Modus sprechen wir vom „sequentiellen Modus“. Hier werden die maximal 16 CAN-Nachrichten als Liste betrachtet, wobei die Zykluszeit jeder CAN-Nachricht bezüglich der Sendezeit ihres Vorgängers überwacht wird (*siehe Bild 16*). Mit diesem Modus ist es möglich, eine CAN-Nachricht mit einer CAN-ID mehrfach in diese Liste aufzunehmen, jedoch mit unterschiedlichen Daten.

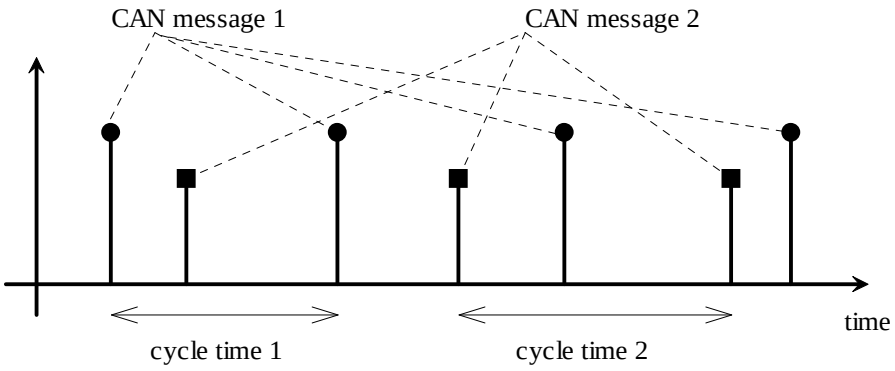


Bild 15: paralleler Modus am Beispiel von 2 CAN-Nachrichten

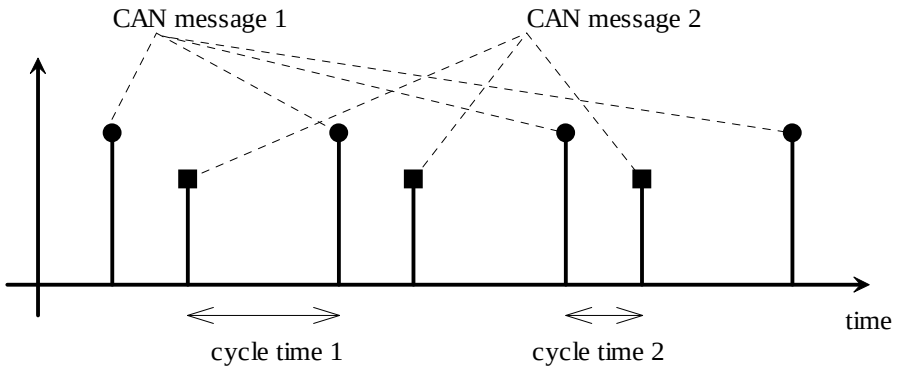


Bild 16: sequentieller Modus am Beispiel von 2 CAN-Nachrichten

Achtung:
Das automatische Senden von zyklischen CAN-Nachrichten kann das Senden über eine PC-Applikation beeinflussen. In dem Fall, bei der die zyklischen CAN-Nachrichten die CAN-Buslast erheblich erhöhen (50% und mehr) werden seltener die CAN-Nachrichten der PC-Applikation bearbeitet. Dies führt dazu, dass die Funktion *UcanWriteCanMsg()* bzw. *UcanWriteCanMsgEx()* keine CAN-Nachrichten mehr senden kann, da der Sendepuffer voll ist.

UcanDefineCyclicCanMsg

Syntax:

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

Verwendbarkeit:
HW_INIT, CAN_INIT ab Version 3.06, ab sysWORXX

Bedeutung:
Definiert eine Liste von bis zu 16 CAN-Nachrichten im USB-CANmodul für das automatische Senden von zyklischen CAN-Nachrichten. Das Senden ist nach Aufruf dieser Funktion noch nicht gestartet. Rufen Sie nachträglich die Funktion *UcanEnableCyclicCanMsg()* um das automatische Senden zu aktivieren. Bitte beachten Sie, dass eine zuvor definierte Liste komplett ersetzt wird.

Parameter:

UcanHandle_p:	USB-CAN-Handle, das mit der Funktion *UcanInitHardware()* bzw. *UcanInitHardwareEx()* erhalten wurde.
bChannel_p:	CAN-Kanal, über den gesendet werden soll. USBCAN_CHANNEL_CH0 für CAN-Kanal 0 USBCAN_CHANNEL_CH1 für CAN-Kanal 1
pCanMsgList_p:	Adresse auf eine Liste von CAN-Nachrichten (als Array vom Typ *tCanMsgStruct*) die zyklisch gesendet werden sollen. Das Element *m_dwTime* der Struktur *tCanMsgStruct* gibt dabei die Zykluszeit in Millisekunden an. Diese Adresse darf nur dann NULL sein, wenn *dwCount_p* 0 ist.
dwCount_p:	Anzahl der CAN-Nachrichten in der Liste. Dieser Parameter kann alle Werte zwischen 0 und 16 annehmen. Wenn der Wert 0 angegeben wird, dann wird nur die alte Liste im Modul gelöscht.

Die Definition der Struktur *tCanMsgStruct* wird bei der Funktion *UcanWriteCanMsg()* beschrieben.

Rückgabewert: Fehlercode der Funktion.
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);
```

Verwendbarkeit:
HW_INIT, CAN_INIT ab Version 3.06, ab sysWORXX

Bedeutung:
Liest die Liste von CAN-Nachrichten aus dem USB-CANmodul zurück, die zuvor für das automatische Senden definiert worden sind.

Parameter:

UcanHandle_p: USB-CAN-Handle, das mit der Funktion *UcanInitHardware()* bzw. *UcanInitHardwareEx()* erhalten wurde.

bChannel_p: CAN-Kanal, über den gesendet werden soll.
USBCAN_CHANNEL_CH0 für CAN-Kanal 0
USBCAN_CHANNEL_CH1 für CAN-Kanal 1

pCanMsgList_p: Adresse auf einen Puffer von CAN-Nachrichten (als Array vom Typ tCanMsgStruct mit max. 16 Einträgen) wohin die Liste der zyklischen CAN-Nachrichten kopiert werden sollen. Diese Adresse darf nicht NULL sein.

pdwCount_p: Adresse auf eine Variable, wohin diese Funktion die Anzahl der zyklischen CAN-Nachrichten innerhalb der Liste schreibt.

Die Definition der Struktur *tCanMsgStruct* wird bei der Funktion *UcanWriteCanMsg()* beschrieben.

Rückgabewert: Fehlercode der Funktion.

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);
```

Verwendbarkeit:
CAN_INIT ab Version 3.06, ab sysWORXX

Bedeutung:
Mit dieser Funktion wird der Modus der Übertragung festgelegt und das Senden der zyklischen CAN-Nachrichten gestartet oder gestoppt. Zusätzlich können einzelne CAN-Nachrichten aus der zuvor definierten Liste gesperrt werden.

Parameter:

UcanHandle_p:	USB-CAN-Handle, das mit der Funktion *UcanInitHardware()* bzw. *UcanInitHardwareEx()* erhalten wurde.
bChannel_p:	CAN-Kanal USBCAN_CHANNEL_CH0 für CAN-Kanal 0 USBCAN_CHANNEL_CH1 für CAN-Kanal 1
dwFlags_p:	Bit-orientierte Flags, mit dem man den Modus festlegen, das Senden starten bzw. stoppen, und einzelne CAN-Nachrichten sperren kann (*siehe Tabelle 15*). Die Flags können miteinander kombiniert werden.

Rückgabewert: Fehlercode der Funktion.

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_…

Konstante **USBCAN_CYCLIC...**	**Wert**	**Bedeutung**
..._FLAG_START	0x80000000	Das automatische Senden wird gestartet wenn dieses Flag gesetzt ist, sonst wird es gestoppt.
..._FLAG_SEQUMODE	0x40000000	Der „sequentielle Modus“ wird verwendet wenn dieses Flag gesetzt ist, sonst wird der „parallele Modus“ verwendet.
..._FLAG_NOECHO	0x00010000	Wenn das Sendeecho eingeschaltet ist, dann werden die zyklischen CAN-Nachrichten nicht als Echo empfangen.
..._FLAG_LOCK_0 bis ..._FLAG_LOCK_15	0x00000001 - 0x00008000	Wenn eines dieser Bits gesetzt ist, dann wird die entsprechende CAN-Nachricht aus der Liste nicht automatisch gesendet.

Tabelle 15: Flags für die Funktion UcanReadCyclicCanMsg()

2.3.2.3 Funktionen für den CAN Port

Die folgenden Funktionen können nur mit dem GW-002-XXX und Multiport CAN-to-USB sowie USB-CANmodul2 verwendet werden (nicht für GW-001). Sie sind eine Erweiterung für die Verwendung des USB-CANmoduls mit einem lowspeed CAN Treiber (z.B.: GW-002-010, GW-002-020, GW-002-030). Das USB-CANmodul1 wird nur mit einem Highspeed CAN-Transceiver angeboten, deshalb werden diese Funktionen für dieses Modul ignoriert (d.h. Fehlercode USBCAN_SUCCESSFUL). Werden diese Funktionen mit dem GW-001 verwendet, dann wird der Fehlercode **USBCAN_ERRCMD_ILLCMD** zurückgegeben. Das Verwenden dieser Funktionen mit dem GW-002 (highspeed 82C251) führt zu keinem Effekt. Es wird aber auch keine Fehlermeldung zurückgegeben. Um diese Funktionen nutzen zu können, muss zusätzlich zu der Header Datei USBCAN32.H die USBCANLS.H als Include eingebunden werden.

UcanWriteCanPort

Syntax:

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

Verwendbarkeit:
HW_INIT, CAN_INIT ab Version 2.15

Bedeutung:
Schreibt einen Wert auf die CAN-Port Schnittstelle. Damit können zusätzliche Signale an einem lowspeed CAN-Transceiver geschaltet werden. Diese Signale sind Standby (STB) und Enable (EN).

Parameter:

UcanHandle_p: USB-CAN-Handle, das mit der Funktion *UcanInitHardware()* bzw. *UcanInitHardwareEx()* erhalten wurde.

bOutValue_p: Neuer Output Wert für die CAN Schnittstelle (*siehe Tabelle 16*).

Rückgabewert: Fehlercode der Funktion.

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

Hinweis:
Nach dem Initialisieren des USB-CANmoduls mit der Funktion *UcanInitCan()* sind diese Signale bereits so eingestellt, dass mit dem USB-CANmodul sofort gearbeitet werden kann.

Ab Software Version 3.00 werden für die sysWORXX Module nach jedem neuen PowerOn des USB-CANmoduls die zuletzt geschriebenen Ausgangswerte eingenommen.

UcanWriteCanPortEx

Syntax:

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

Verwendbarkeit:
HW_INIT, CAN_INIT ab Version 3.00

Bedeutung:
Schreibt einen Wert auf die CAN-Port Schnittstelle eines bestimmten CAN-Kanals. Diese Funktion wird als Alternative zur Funktion *UcanWriteCanPort()* verwendet.

Parameter:

UcanHandle_p:	USB-CAN-Handle, das mit der Funktion *UcanInitHardware()* bzw. *UcanInitHardwareEx()* erhalten wurde.
bChannel_p:	CAN-Kanal, dessen Port geschrieben werden soll. USBCAN_CHANNEL_CH0 für CAN-Kanal 0 USBCAN_CHANNEL_CH1 für CAN-Kanal 1
bOutValue_p:	Neuer Output-Wert für die CAN Schnittstelle (*siehe Tabelle 16*).

Rückgabewert: Fehlercode der Funktion.

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

UcanReadCanPort

Syntax:

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

Verwendbarkeit:
HW_INIT, CAN_INIT ab Version 2.15

Bedeutung:
Liest den aktuellen Input-Wert aus der CAN-Port Schnittstelle. Damit kann das zusätzliche Signal (ERR für Fehler) an einem lowspeed CAN Treiber gelesen werden. Der Abschlusswiderstand kann auch für highspeed CAN-Transceiver zurückgelesen werden (momentan nur für USB-CANmodul2).

Parameter:

UcanHandle_p: USB-CAN-Handle, das mit der Funktion *UcanInitHardware()* bzw. *UcanInitHardwareEx()* erhalten wurde.

pbInValue_p: Adresse auf eine Variable, die nach erfolgreicher Rückkehr dieser Funktion den gelesenen Input-Wert enthält. Diese Variable ist Bit-orientiert mit folgenden Bedeutungen (*siehe auch Kapitel 1.4*):

Konstante	Bit-Wert	Bedeutung
UCAN_CANPORT_TRM	0x10	[IN] Abschlusswiderstand
UCAN_CANPORT_ERR	0x20	[IN] Fehlerstatus
UCAN_CANPORT_STB	0x40	[OUT] Standby
UCAN_CANPORT_EN	0x80	[OUT] Einschaltsignal

Tabelle 16: Konstanten für den lowspeed CAN-Port

Rückgabewert: Fehlercode der Funktion.

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

UcanReadCanPortEx

Syntax:

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

Verwendbarkeit:
HW_INIT, CAN_INIT ab Version 3.00

Bedeutung:
Liest den aktuellen Input-Wert aus der CAN-Port Schnittstelle eines bestimmten CAN-Kanals. Diese Funktion wird als Alternative zur Funktion *UcanReadCanPort()* verwendet.

Parameter:

UcanHandle_p:	USB-CAN-Handle, das mit der Funktion *UcanInitHardware()* bzw. *UcanInitHardwareEx()* erhalten wurde.
bChannel_p:	CAN-Kanal, dessen Port gelesen werden soll. USBCAN_CHANNEL_CH0 für CAN-Kanal 0 USBCAN_CHANNEL_CH1 für CAN-Kanal 1
pbInValue_p:	Adresse auf eine Variable, die nach erfolgreicher Rückkehr dieser Funktion den gelesenen Input-Wert enthält (*siehe Tabelle 16*).
pbLastOut_p:	Adresse auf eine Variable, die nach erfolgreicher Rückkehr dieser Funktion den zuletzt geschriebenen Wert enthält (mit *UcanWriteCanPort()* oder *UcanWriteCanPortEx()* geschriebener Wert). Diesem Parameter kann NULL übergeben werden.

Rückgabewert: Fehlercode der Funktion.

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

2.3.2.4 Funktionen für die Port Erweiterung

Die folgenden Funktionen können nur mit dem GW-002-XXX und Multiport CAN-to-USB 3004006 sowie USB-CANmodul2 3204002/3204003 verwendet werden. Sie sind eine Erweiterung für die Verwendung des USB-CANmoduls mit der Port Erweiterung. Das USB-CANmodul1 3204000/3204001 besitzt keine Port Erweiterung, deshalb werden dort diese Funktionen ignoriert (d.h. Fehlercode USBCAN_SUCCESSFUL). Werden diese Funktionen mit dem GW-001 verwendet, dann wird der Fehlercode **USBCAN_ERRCMD_ILLCMD** zurückgegeben. Um diese Funktionen nutzen zu können, muss zusätzlich zu der Header Datei USBCAN32.H die USBCANUP.H als Include eingebunden werden.

UcanConfigUserPort

Syntax:

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

Verwendbarkeit:
HW_INIT, CAN_INIT ab Version 2.16

Bedeutung:
Konfiguriert die Port Erweiterung *(siehe Kapitel 1.5)*. Jedes einzelne Pin des 8 Bit Port kann wahlweise als Ein- oder Ausgang genutzt werden. Eine logische 0 eines Bits im Parameter *bOutputEnable_p* definiert das entsprechende Pin an der Port Erweiterung als Eingang und eine logische 1 definiert es als Ausgang.

Parameter:

UcanHandle_p:	USB-CAN-Handle, das mit der Funktion *UcanInitHardware()* bzw. *UcanInitHardwareEx()* erhalten wurde.
bOutEn_p:	Konfiguration des 8 Bit Port als Ein- oder Ausgang. Bit X = 0: Pin X = Eingang Bit Y = 1: Pin Y = Ausgang

Rückgabewert: Fehlercode der Funktion.

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

Hinweis:
Nach dem Anstecken des USB-CANmoduls an den PC sind alle Pins der Port Erweiterung als Eingang konfiguriert.

Ab Software Version 3.00 und ab den sysWORXX-Modulen wird nach jedem PowerOn des USB-CANmoduls die zuletzt geschriebene Konfiguration eingenommen.

UcanWriteUserPort

Syntax:

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

Verwendbarkeit:
HW_INIT, CAN_INIT ab Version 2.16

Bedeutung:
Schreibt einen Wert auf die Ausgange der Port Erweiterung. Damit die Ausgänge auch gesetzt werden, müssen die entsprechenden Bits zuvor mit der Funktion *UcanConfigUserPort()* auf Ausgang konfiguriert werden.

Parameter:

UcanHandle_p: USB-CAN-Handle, das mit der Funktion *UcanInitHardware()* bzw. *UcanInitHardwareEx()* erhalten wurde.

bOutValue_p: Neuer Output-Wert für die Ausgänge der Port Erweiterung. Jedes Bit entspricht dem jeweiligen Pin an diesem Port.

Rückgabewert: Fehlercode der Funktion.

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

Hinweis:
Die Spannungsversorgung der Port Erweiterung (Pin 10, *siehe Tabelle 6*) wird beim GW-002 erst nach Aufruf der Funktion *UcanInitCan()* zugeschaltet. Nach dem Anstecken des USB-CANmodul an den PC sind alle Pins der Port Erweiterung als Eingang konfiguriert. Mit dieser Funktion können keine zeitkritischen Schaltvorgänge realisiert werden, da die Reaktionszeit durch mehrere Faktoren beeinflusst wird.

Ab Software Version 3.00 und ab den sysWORXX-Modulen wird nach jedem PowerOn des USB-CANmoduls der zuletzt geschriebene Wert an den konfigurierten Ausgängen eingenommen.

UcanReadUserPort

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

Verwendbarkeit:
HW_INIT, CAN_INIT ab Version 2.16

Bedeutung:
Liest den aktuellen Input-Wert aus der Port Erweiterung.

Parameter:

UcanHandle_p:	USB-CAN-Handle, das mit der Funktion *UcanInitHardware()* bzw. *UcanInitHardwareEx()* erhalten wurde.
pbInValue_p:	Adresse auf eine Variable, die nach erfolgreicher Rückkehr dieser Funktion den gelesenen Input-Wert enthält. Diese Variable enthält dann den Zustand der 8 Bit Port Erweiterung. Jedes Bit entspricht dem jeweiligen Pin an diesem Port.

Rückgabewert: Fehlercode der Funktion.

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

Hinweis:
Nach dem Anstecken des USB-CANmodul an den PC sind alle Pins der Port Erweiterung als Eingang konfiguriert (außer den sysWORXX-Modulen). Mit dieser Funktion können auch die Zustände der Ausgänge zurückgelesen werden.

UcanReadUserPortEx

Syntax:

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

Verwendbarkeit:
HW_INIT, CAN_INIT ab Version 3.00

Bedeutung:
Liest den aktuellen Input-Wert aus der Port Erweiterung. Diese Funktion wird als Alternative zur Funktion *UcanReadUserPort()* verwendet.

Parameter:

UcanHandle_p:	USB-CAN-Handle, das mit der Funktion *UcanInitHardware()* bzw. *UcanInitHardwareEx()* erhalten wurde.
pbInValue_p:	Adresse auf eine Variable, die nach erfolgreicher Rückkehr dieser Funktion den gelesenen Input-Wert enthält. Diese Variable enthält dann den Zustand der 8 Bit Port Erweiterung. Jedes Bit entspricht dem jeweiligen Pin an diesem Port.
pbLastOutEn_p:	Adresse auf eine Variable, die nach erfolgreicher Rückkehr dieser Funktion die letzte Konfiguration enthält (Konfiguration, die mit der Funktion *UcanConfigUserPort()* durchgeführt wurde). Diesem Parameter kann NULL übergeben werden.
pbLastOutVal_p:	Adresse auf eine Variable, die nach erfolgreicher Rückkehr dieser Funktion den zuletzt geschriebenen Output-Wert enthält (Wert, der mit der Funktion *UcanWriteUserPort()* geschrieben wurde). Diesem Parameter kann NULL übergeben werden.

Rückgabewert: Fehlercode der Funktion.

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

2.3.3 Fehlercodes der Funktionen

Die Funktionen der USBCAN-Library geben ein Fehlercode vom Typ UCANRET zurück. Jeder Rückgabewert repräsentiert einen Fehler. Die Funktion *UcanReadCanMsg()* macht dabei jedoch eine Ausnahme. Sie kann auch Warnungen zurückgeben. Die Warnung *USBCAN_WARN_NODATA* zeigt an, dass keine CAN-Nachrichten im Puffer sind. Andere Warnungen zeigen der aufrufenden Funktion, dass ein Ereignis aufgetreten ist, jedoch eine gültige CAN-Nachricht übergeben wurde.

Es folgen alle möglichen Rückgabecodes der Funktionen der USBCAN-Library:

USBCAN_SUCCESSFUL

Wert: 0x00

Bedeutung:
Die Funktion wurde erfolgreich ausgeführt.

USBCAN_ERR_RESOURCE

Wert: 0x01

Bedeutung:
Eine Ressource konnte nicht erzeugt werden. Unter dem Begriff Ressourcen sind Speicher und Handle zusammengefasst, die von Windows vergeben werden.

USBCAN_ERR_MAXMODULES

Wert: 0x02

Bedeutung:
Es wurde versucht mehr als die maximale Anzahl von USB-CANmodule zu öffnen. Die maximale Anzahl beträgt in der Standardversion der USBCAN32.DLL 64 (unter Windows CE nur 9). Dieser Fehler tritt auch auf, wenn mehrere

Anwendungen auf mehr als 64 USB-CANmodule zugreifen wollen (z.B.: Anwendung 1 hat 60 Module geöffnet, Anwendung 2 hat 4 Module geöffnet und Anwendung 3 will ein Modul öffnen. Anwendung 3 erhält diesen Fehler.).

USBCAN_ERR_HWINUSE

Wert: 0x03

Bedeutung:
Es wurde versucht ein USB-CANmodul mit der Gerätenummer x zu initialisieren. Ist dieses Modul bereits von der eigenen oder einer anderen Anwendung initialisiert worden, dann wird dieser Fehler zurückgegeben.

USBCAN_ERR_ILLVERSION

Wert: 0x04

Bedeutung:
Die Softwareversion der Firmware im USB-CANmodul ist nicht kompatibel zu der Version der USBCAN-Library. In diesem Fall muss der USB-CAN-Treiber neu installiert werden.

USBCAN_ERR_ILLHW

Wert: 0x05

Bedeutung:
Es wurde kein USB-CANmodul mit der Gerätenummer x gefunden. Wurde die Funktion *UcanInitHardware()* bzw. *UcanInitHardwareEx()* mit der Gerätenummer *USBCAN_ANY_MODULE* gerufen, dann ist kein Modul am PC angesteckt, oder es werden alle angesteckten Module bereits verwendet.

USBCAN_ERR_ILLHANDLE

Wert: 0x06

Bedeutung:
Einer Funktion wurde ein falsches USB-CAN-Handle übergeben. Die Funktion prüft zuerst, welches USB-CANmodul zu diesem Handle initialisiert wurde. Wurde dafür kein Modul initialisiert, erhalten Sie diesen Fehler.

USBCAN_ERR_ILLPARAM

Wert: 0x07

Bedeutung:
Einer Funktion wurde ein falscher Parameter übergeben. Häufigster Fehler ist zum Beispiel eine übergebene NULL anstatt einer gültigen Adresse auf eine Variable.

USBCAN_ERR_BUSY

Wert: 0x08

Bedeutung:
Verwenden Sie in einer Anwendung mehrere Threads, die auf ein USB-CANmodul zugreifen, kann diese Fehlermeldung auftreten. Nachdem die anderen Threads ihre Arbeit abgeschlossen haben, kann die Funktion noch einmal aufgerufen werden.

USBCAN_ERR_TIMEOUT

Wert: 0x09

Bedeutung:
Die Funktion hat ein Kommando an das USB-CANmodul gesendet und erhält keine Antwort. Um den Fehler zu beheben

muss die Anwendung beendet werden, und das USB-CANmodul kurz abgezogen und neu angesteckt werden.

USBCAN_ERR_IOFAILED

Wert: 0x0a

Bedeutung:
Dieser Fehler tritt auf, wenn die Kommunikation zum USB-CAN-Treiber fehlgeschlagen ist. Dieser Fehler kann auftreten, wenn während einer Funktionsausführung das USB-CANmodul abgezogen wird.

USBCAN_ERR_DLL_TXFULL

Wert: 0x0b

Bedeutung:
Die Funktion *UcanWriteCanMsg() bzw. UcanWriteCanMsgEx()* prüft zuerst, ob im Sendepuffer der USBCAN-Library noch Platz für eine CAN-Nachricht ist. Ist kein Platz mehr, so wird diese Fehlermeldung zurückgegeben. Die an der Funktion *UcanWriteCanMsg()* bzw. *UcanWriteCanMsgEx()* übergebene CAN-Nachricht wurde somit nicht in den Sendepuffer abgelegt, um andere CAN-Nachrichten nicht zu überschreiben. Ab der Treiberversion 3.05 kann die Größe des Sendepuffers eingestellt werden (*siehe Funktion UcanInitCanEx() und Struktur tUcanInitCanParam*).

USBCAN_ERR_MAXINSTANCES

Wert: 0x0c

Bedeutung:
Auf die USBCAN32.DLL können in der aktuellen Version maximal 64 Anwendungen zugreifen (unter Windows CE nur 9). Greift eine weitere Anwendung auf die DLL zu, wird dieser

Fehler zurückgegeben. Es ist damit auch nicht möglich ein USB-CANmodul zu initialisieren.

USBCAN_ERR_CANNOTINIT

Wert: 0x0d

Bedeutung:
Wenn ein USB-CANmodul mit der Funktion *UcanInitHardware()* bzw. *UcanInitHardwareEx()* initialisiert wurde, befindet sich die Software in dem Zustand HW_INIT. Funktionen wie *UcanReadCanMsg()* oder *UcanWriteCanMsg()* liefern in diesem Zustand diesen Fehler zurück. Mit der Funktion *UcanInitCan()* gelangt die Software in den Zustand CAN_INIT. In diesem Zustand können CAN-Nachrichten gelesen und gesendet werden.

USBCAN_ERR_DISCONNECT

Wert: 0x0e

Bedeutung:
Dieser Fehlercode wird zurückgegeben, wenn eine Funktion der USBCAN-Library für ein USB-CANmodul gerufen wurde, das gerade vom PC abgezogen wurde (USB-Kabel).

USBCAN_ERR_NOHWCLASS

Wert: 0x0f

Bedeutung:
Dieser Fehlercode ist veraltet und wird nicht zurückgegeben.

USBCAN_ERR_ILLCHANNEL

Wert: 0x10

Bedeutung:
Dieser Fehlercode wird zurückgegeben, wenn eine erweiterte Funktion der USBCAN-Library mit dem Parameter bChannel_p = USBCAN_CHANNEL_CH1 gerufen wurde, obwohl das USB-CANmodul GW-001, GW-002 bzw. USB-CANmodul1 verwendet wird.

USBCAN_ERR_ILLHWTYPE

Wert: 0x12

Bedeutung:
Dieser Fehlercode wird zurückgegeben, wenn eine erweiterte Funktion der USBCAN-Library für eine Hardware gerufen wurde, die dieses Feature nicht unterstützt.

USBCAN_ERRCMD_NOTEQU

Wert: 0x40

Bedeutung:
Dieser Fehler tritt bei der Kommunikation zwischen dem PC und dem USB-CANmodul auf. Der PC sendet ein Kommando an ein USB-CANmodul. Dieses Modul führt das Kommando aus und sendet eine Antwort an den PC zurück. Gehört die Antwort nicht zum Kommando, so wird dieser Fehler zurückgegeben.

USBCAN_ERRCMD_REGTST

Wert: 0x41

Bedeutung:
Wenn die CAN-Schnittstelle auf dem USB-CANmodul initialisiert wird, wird getestet, ob der CAN-Controller angesprochen werden kann. Dabei werden einige Register des CAN-Controllers überprüft. Tritt bei diesem Registertest ein Fehler auf, so erhalten Sie diese Fehlermeldung.

USBCAN_ERRCMD_ILLCMD

Wert: 0x42

Bedeutung:
Das USB-CANmodul hat ein Kommando empfangen, welches nicht definiert ist. Dieser Fehler deutet auf einen Versionskonflikt zwischen der Firmware im USB-CANmodul und der USBCAN-Library hin.

USBCAN_ERRCMD_EEPROM

Wert: 0x43

Bedeutung:
Das USB-CANmodul hat eine seriellen EEPROM, in dem die Gerätenummer und die Seriennummer abgelegt ist. Tritt beim Auslesen dieser Werte ein Fehler auf, wird dieser Fehler zurückgegeben.

USBCAN_ERRCMD_ILLBDR

Wert: 0x47

Bedeutung:
Das Multiport CAN-to-USB 3004006, USB-CAnmodul1 3204000/3204001 oder USB-CANmodul2 3204002/3204003

wurde mit einer unbekannten Standard-Baudrate (BTR0 und BTR1) initialisiert.

USBCAN_ERRCMD_NOTINIT

Wert: 0x48

Bedeutung:
Es wurde versucht, auf einen CAN-Kanal des Multiport CAN-to-USB 3004006 oder des USB-CANmodul2 3204002/3204003 zuzugreifen, der jedoch zuvor nicht initialisiert wurde.

USBCAN_ERRCMD_ALREADYINIT

Wert: 0x49

Bedeutung:
Es wurde versucht, einen CAN-Kanal des Multiport CAN-to-USB 3004006 oder des USB-CANmodul2 3204002/3204003 zu initialisieren, der jedoch zuvor schon initialisiert wurde.

USBCAN_WARN_NODATA

Wert: 0x80

Bedeutung:
Kehrt die Funktion *UcanReadCanMsg()* mit dieser Warnung zurück, dann befindet sich keine CAN-Nachricht im Empfangspuffer.

USBCAN_WARN_SYS_RXOVERRUN

Wert: 0x81

Bedeutung:
Läuft im USB-CAN-Systemtreiber der Empfangspuffer über, dann wird dies der USBCAN-Library mitgeteilt. Die Funktion *UcanReadCanMsg()* gibt diese Warnung und eine gültige CAN-Nachricht zurück. Diese Warnung zeigt an, dass CAN-Nachrichten verloren gegangen sind. Die Position der verlorenen CAN-Nachrichten wird mit dieser Warnung nicht angezeigt!

USBCAN_WARN_DLL_RXOVERRUN

Wert: 0x82

Bedeutung:
Die CAN-Nachrichten werden von der USBCAN-Library automatisch vom USB-CANmodul abgefordert und in einem Zwischenpuffer abgelegt. Werden mehr CAN-Nachrichten empfangen, als in den Puffer hineinpassen, dann wird diese Fehlermeldung zurückgegeben. Dabei kommt es zum Verlust von CAN-Nachrichten. Die Position der verlorenen CAN-Nachrichten wird mit dieser Warnung nicht angezeigt! Ab der Treiberversion 3.05 kann die Größe des Empfangspuffers eingestellt werden (*siehe Funktion UcanInitCanEx() und Struktur tUcanInitCanParam*).

USBCAN_WARN_FW_TXOVERRUN

Wert: 0x85

Bedeutung:
Diese Warnung wird von der Funktion *UcanWriteCanMsg()* bzw. *UcanWriteCanMsgEx()* zurückgegeben, wenn im CAN-Treiberstatus das Flag USBCAN_CANERR_QXMTFULL gesetzt ist. Die zu sendende CAN-Nachricht ist jedoch in den

DLL-Puffer erfolgreich abgelegt worden. Mit dieser Warnung wird angezeigt, dass mindestens eine zu sendende CAN-Nachricht in der Modul-Firmware verloren gegangen ist. Die Position der fehlenden CAN-Nachricht(en) wird mit dieser Warnung nicht angezeigt!

USBCAN_WARN_FW_RXOVERRUN

Wert: 0x86

Bedeutung:
Diese Warnung wird von der Funktion *UcanReadCanMsg()* bzw. *UcanReadCanMsgEx()* zurückgegeben, wenn im CAN-Treiberstatus die Flags USBCAN_CANERR_QOVERRUN oder USBCAN_CANERR_OVERRUN gesetzt sind. Die Funktion kehrt jedoch mit einer gültigen CAN-Nachricht zurück. Mit dieser Warnung wird angezeigt, dass mindestens eine empfangene CAN-Nachricht in der Modul-Firmware verloren gegangen ist. Die Position der fehlenden CAN-Nachricht(en) wird mit dieser Warnung nicht angezeigt!

USBCAN_WARN_NULL_PTR

Wert: 0x90

Bedeutung:
Diese Warnung wird von den Funktionen *UcanInitHwConnectContro()* und *UcanInitHwConnectControlEx()* zurückgegeben, wenn für die Adresse auf die Callback Funktion NULL übergeben wurde.

USBCAN_WARN_TXLIMIT

Wert: 0x91

Bedeutung:
Diese Warnung wird von der Funktion *UcanWriteCanMsgEx()* zurückgegeben, wenn diese gerufen wurde, mehr als eine CAN-

Nachricht zu senden, aber nur ein Teil dieser CAN-Nachrichten passte in den Sendepuffer der USBCAN-Library. Der Parameter *pdwCount_p* der Funktion *UcanWriteCanMsgEx()* erhält die Anzahl der CAN-Nachichten, die erfolgreich in diesen Puffer geschrieben werden konnten.

2.3.4 Baudrateneinstellung

Die Baudrateneinstellung wird beim USB-CANmodul GW-001 und GW-002 als Parameter *bBTR0_p* und *bBTR1_p* an die Funktion *UcanInitCan(), UcanInitCanEx()* bzw. *UcanInitCanEx2()* übergeben. Die Einstellung kann nach dem Aufruf dieser Funktion auch nachträglich durch die Funktion *UcanSetBaudrate()* bzw. *UcanSetBaudrateEx()* geändert werden. Folgende Werte werden empfohlen:

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

Beispiel:

```
tUcanHandle UcanHandle;
UCANRET bRet;

...
// Hardware initialisieren
bRet = UcanInitHardware (&UcanHandle, 0, NULL);

// CAN-Schnittstelle initialisieren
bRet = UcanInitCan (UcanHandle,
  HIBYTE (USBCAN_BAUD_1MBit),   // BTR0 für 1MBit/s
  LOBYTE (USBCAN_BAUD_1MBit),   // BTR1 für 1MBit/s
  USBCAN_AMR_ALL,               // AMR: alle Nachrichten empfangen
  USBCAN_ACR_ALL);              // ACR

// Fehler? dann Fehler ausgeben
if (bRet != USBCAN_SUCCESSFUL)
    PrintError (bRet);
...
```

Es können auch andere Baudraten eingestellt werden. Nachfolgend wird das Format der beiden Baudraten Register BTR0 und BTR1 erläutert. Für nähere Informationen wird auf das Handbuch des SJA1000 verwiesen.

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

Bild 17: Format des Baudraten Registers BTR0

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

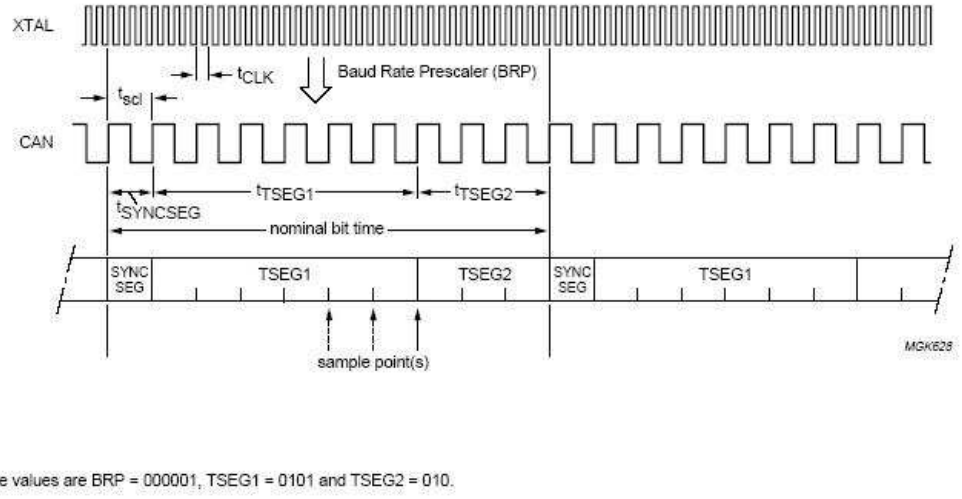
Bild 18: Format des Baudraten Registers BTR1

BPR: *Baudrate Prescaler* bestimmt das Teilerverhältnis zwischen Systemtakt des SJA1000 und dem Takt auf dem CAN-Bus.

SJW: *Synchronization Jump Width* bestimmt die Kompensation der Phasenverschiebung zwischen den Systemtakt und den unterschiedlichen CAN-Controller, die am CAN-Bus angeschlossen sind.

SAM: *Sampling* bestimmt, wie viel *Sample Points* für das Lesen der Bits auf dem CAN-Bus durchgeführt werden. Ist SAM = 1 werden 3 *Sample Points* durchgeführt, sonst nur ein *Sample Point*.

TSEG: *Time Segment* bestimmt die Anzahl der Taktzyklen auf dem CAN-Bus für ein Bit und die Lage der *Sample Points*.



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

Bild 19: generelle Struktur eines Bits auf dem CAN-Bus (Quelle: Handbuch des SJA1000)

Es gelten folgende mathematische Zusammenhänge:

t_{CLK}	= 1 / 16MHz	(Systemtakt)
t_{scl}	= 2 * t_{CLK} * (BRP + 1)	(Takt auf dem CAN-Bus)
$t_{SYNCSEG}$	= 1 * t_{scl}	
t_{TSEG1}	= t_{scl} * (TSEG1 + 1)	
t_{TSEG2}	= t_{scl} * (TSEG2 + 1)	
t_{Bit}	= $t_{SYNCSEG}$ + t_{TSEG1} + t_{TSEG2}	(Zeit für ein Bit auf dem CAN-Bus)

Beispiel für 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/s	

Hinweis:
Die Baudrateneinstellungen zwischen GW-001 bzw. GW-002 und den neuen sysWORXX-Modulen unterscheiden sich sehr stark. Für Standardwerte *(siehe Kapitel 2.3.4)* kann für die sysWORXX-Module auch BTR0 und BTR1 eingestellt werden. Dabei muss jedoch der Wert *m_dwBaudrate* auf den Wert USBCAN_BAUDEX_USE_BTR01 gesetzt werden.

Für sysWORXX-Module gibt es folgende Standardwerte:

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

Es können auch andere Werte als oben definiert vom Anwender eingestellt werden. Nachfolgend wird das Format des erweiterten Baudraten Registers erläutert.

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		

Bild 20: Format des erweiterten Baudraten Registers für die sysWORXX-Module

BPR: *Baudrate Prescaler* bestimmt das Teilerverhältnis zwischen Systemtakt des Microcontroller und dem Takt auf dem CAN-Bus.

SYNC: *Synchronization Jump Width* bestimmt die Kompensation der Phasenverschiebung zwischen dem Systemtakt und den unterschiedlichen CAN-Controller, die am CAN-Bus angeschlossen sind.

SMP: *Sampling Mode* bestimmt, wie viele *Sample Points* für das Lesen der Bits auf dem CAN-Bus durchgeführt werden. Ist SAM = 1 werden 3 *Sample Points* durchgeführt, sonst nur ein *Sample Point*.

PROPAG: *Programming Time Segment* bestimmt die Kompensation der physikalischen Verzögerungszeit auf dem CAN-Bus.

PHASE: *Phase Segment* bestimmt die Anzahl der Taktzyklen auf dem CAN-Bus für ein Bit und die Lage der *Sample Points*.

CLK: *Clock* bestimmt die Frequenz des Microcontroller. Wenn dieses Bit auf 0 gesetzt ist, dann läuft der Microcontroller intern mit 48 MHz, sonst mit 24 MHz. Dadurch verändert sich die Baudrate auf dem CAN-Bus (siehe Systemtakt t_{MCK} im folgenden Beispiel)

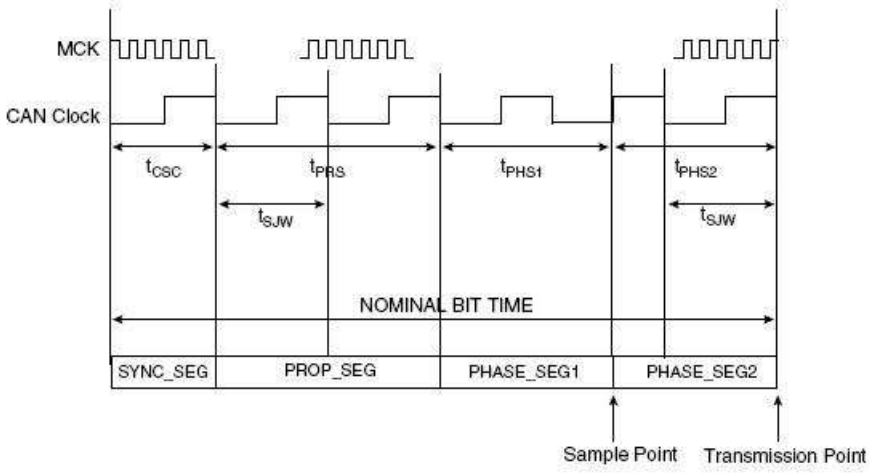


Bild 21: generelle Struktur eines Bits auf dem CAN-Bus (Quelle: Handbuch zum Atmel AT91SAM7A3)

Es gelten folgende mathematische Zusammenhänge:

t_{MCK0}	= 1 / 48MHz	(Systemtakt CLK = 0)
t_{MCK1}	= 1 / 24MHz	(Systemtakt CLK = 1)
t_{CSC}	= t_{MCKx} * (BRP + 1)	(Takt auf CAN-Bus)
$t_{SYNCSEG}$	= 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}	(Zeit für ein Bit auf CAN-Bus)

Beispiel für 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/s	

Hinweis:
Für die Kompatibilität zu bereits bestehenden Applikationen wurde die Konstante USBCAN_BAUDEX_USE_BTR01 definiert. Wird sie für die Baudrateneinstellung die sysWORXX-Module eingestellt, dann sind weiterhin die Definitionen für BTR0 und BTR1 verwendbar. Jedoch sind nur die Baudraten verwendbar, die in diesem Handbuch dokumentiert wurden. Wird versucht eine Anwender-Baudrate einzustellen, dann wird der Fehlercode USBCAN_ERRCMD_ILLBDR zurückgegeben.

Beispiel 1:

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

...

// Initialisierungsparameter ausfüllen
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;

// CAN-Kanal initialisieren
bRet = UcanInitCanEx (UcanHandle, USBCAN_CHANNEL_CH0,
    &InitParam);
...
```

Beispiel 2 (funktioniert jedoch nicht für GW-001 / GW-002):

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

...

// Initialisierungsparameter ausfüllen
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_125kBit;
InitParam.m_wNrOfRxBufferEntries = USBCAN_DEFAULT_BUFFER_ENTRIES;
InitParam.m_wNrOfTxBufferEntries = USBCAN_DEFAULT_BUFFER_ENTRIES;

// CAN-Kanal initialisieren
bRet = UcanInitCanEx (UcanHandle, USBCAN_CHANNEL_CH0,
    &InitParam);
...
```

2.3.5 Filterung von CAN-Nachrichten

Es ist möglich, die zu empfangenden CAN-Nachrichten zu filtern. Die Filterung nimmt der CAN-Controller automatisch vor. Sie entspricht der Filterung des SJA1000 im PeliCAN-Modus (Single-Filter-Modus).

Die Einstellungen des Filters werden als Parameter *dwAMR_p* und *dwACR_p* an die Funktion *UcanInitCan()* übergeben. Diese Werte können nach Aufruf dieser Funktion mit der Funktion *UcanSetAcceptance()* nachträglich geändert werden.

Folgender Mechanismus gilt für die Filterung:

AMR-Bit	ACR-Bit	Bit der CAN-ID
0	0	0
0	1	1
1	0	x
1	1	x

0 Das dazugehörige Bit des CAN-Identifiers muss 0 sein.
1 Das dazugehörige Bit des CAN-Identifiers muss 1 sein.
x Das dazugehörige Bit des CAN-Identifiers kann 0 oder 1 sein.

Die Zugehörigkeit der Bits:

a) Standard-Frame (11-Bit-Identifier) bei GW-001 und GW-002 sowie sysWORXX Module ab 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						**Datenbyte 0**				**Datenbyte 1**			
10	9	...	0	RTR	Unbenutzt	7	6	...	0	7	6	...	0

b) Extended-Frame (29-Bit-Identifier) bei GW-001 und GW-002 sowie sysWORXX Module ab 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	unbenutzt

c) Standard-Frame (11-Bit-Identifier) bei allen sysWORXX-Modulen bis Firmware-Version V3.09:

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	unbenutzt

d) Extended-Frame (29-Bit-Identifier) bei allen sysWORXX-Modulen bis Firmware-Version V3.09:

dwACR_p						
31	30	...	3	2	1	0

⇓

dwAMR_p						
31	30	...	3	2	1	0

⇓

CAN-Identifier				
28	27	...	0	unbenutzt

Es können die Makros *USBCAN_SET_AMR(extended, can_id, rtr)* und *USBCAN_SET_AMR(extended, can_id, rtr)* bzw. *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)* verwendet werden, um die Filterwerte zu berechnen.

Der Parameter *extended* gibt dabei an, ob die Parameter *can_id*, *from_id* und *to_id* 29-Bit-Identifier (TRUE) oder 11-Bit-Identifier (FALSE) angeben. Mit *can_id* wird der Filterwert als CAN-Identifier angegeben, *from_id* und *to_id* geben einen zu filternden Bereich des CAN-Identifier an. Um auch gezielt RTR-Frames filtern zu können gibt es die Parameter *rtr*, *rtr_only* und *rtr_too*. Diese Parameter können die Werte TRUE (=1) und FALSE (=0) annehmen. Ist der Parameter *rtr_only* = TRUE, dann werden nur RTR Frames empfangen und der Parameter *rtr_too* wird ignoriert. Andernfalls werden auch RTR Frames empfangen, wenn *rtr_too* = TRUE ist.

Beispiel 1:

```
tUcanHandle UcanHandle;
UCANRET bRet;

...
// Hardware initialisieren
bRet = UcanInitHardware (&UcanHandle, 0, NULL);

// CAN-Schnittstelle initialisieren
// 11-Bit-CAN-Nachrichten mit ID 0x300 bis 0x3ff filtern,
// RTR-Frames und Datenframes werden empfangen
bRet = UcanInitCan (UcanHandle,
  HIBYTE (USBCAN_BAUD_1MBit),
  LOBYTE (USBCAN_BAUD_1MBit),
  USBCAN_SET_AMR (FALSE, 0x0ff, 1),
  USBCAN_SET_ACR (FALSE, 0x300, 0));

// Fehler? dann Fehler ausgeben
if (bRet != USBCAN_SUCCESSFUL)
    PrintError (bRet);
...
```

Wird nach Beispiel 1 eine CAN-Nachricht mit 29-Bit-Identifier vom CAN-Controller empfangen, dann wird sie mit denselben Einstellungen gefiltert. Es werden jedoch AMR und ACR anders interpretiert. Hier werden auch CAN-Nachrichten mit den 29-Bit-Identifier von 0x0C000000 bis 0FFFFFFF empfangen.

Beispiel 2:

```
tUcanHandle UcanHandle;
UCANRET bRet;

...
// Hardware initialisieren
bRet = UcanInitHardware (&UcanHandle, 0, NULL);

// CAN-Schnittstelle initialisieren
// 11-Bit-CAN-Nachrichten mit ID 0x600 bis 0x67F filtern,
// RTR-Frames werden nicht empfangen
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));

// Fehler? dann Fehler ausgeben
if (bRet != USBCAN_SUCCESSFUL)
    PrintError (bRet);
...
```

2.3.6 Verwendung von mehreren CAN-Kanälen

Nur das Multiport CAN-to-USB 3004006 und das USB-CANmodul2 3204002/3204003/3204007 sowie das USB-CANmodul8 und USB-CANmodul16 besitzen mehrere CAN-Kanäle. Die 16 CAN-Kanäle des Standard Multiport CAN-to-USB sind jedoch auf 8 logische Geräte aufgeteilt. Das heißt, je initialisierte Hardware (Aufruf der Funktion *UcanInitHardware()* bzw. *UcanInitHardwareEx()*) können immer nur bis zu 2 CAN-Kanäle verwendet werden. Sollen alle 16 CAN-Kanäle verwendet werden, dann müssen alle 8 logische Geräte initialisiert werden und von all diesen logischen Geräten müssen beide CAN-Kanäle (Aufruf der Funktion *UcanInitCanEx2()*) initialisiert werden. Äquivalent verhält sich das USB-CANmodul8 mit 4 logischen Modulen, das USB-CANmodul16 mit 8 logischen Modulen und das USB-CANmodul2 mit einem logischen Modul.

Für die Verwendung der CAN-Kanäle gibt es 3 Konstanten:

Konstante	Wert	Bedeutung
USBCAN_CHANNEL_CH0	0	erster CAN-Kanal
USBCAN_CHANNEL_CH1	1	zweiter CAN-Kanal
USBCAN_CHANNEL_ANY	255	unbestimmter CAN-Kanal

Die Konstante USBCAN_CHANNEL_ANY kann nur mit den Funktionen *UcanReadCanMsgEx()* und *UcanGetMsgPending()* verwendet werden. Bei der Funktion *UcanReadCanMsgEx()* gibt sie an, dass die Funktion erst prüfen soll, aus welchem CAN-Kanal die nächste CAN-Nachricht stammt. Kehrt diese Funktion mit dem Rückgabecode USBCAN_SUCCESSFUL zurück, dann übergibt sie der aufrufenden Funktion auch den jeweiligen CAN-Kanal aus dem die gelesene CAN-Nachricht stammt: USBCAN_CHANNEL_CH0 oder USBCAN_CHANNEL_CH1 (*siehe dazu auch Funktionsbeschreibung UcanReadCanMsgEx()*) .

2.3.7 Verwendung der Callback Funktionen

Die USBCAN-Library stellt zwei Typen von Callback Funktionen zur Verfügung. Die Connect Control Callback Funktion meldet Plug&Play Ereignisse für das USB-CANmodul (z.B.: neues USB-CANmodul mit dem PC verbunden; oder abgezogen; …). Der zweite Typ meldet Ereignisse, die während der Arbeit mit dem USB-CANmodul auftreten (z.B.: CAN-Nachricht empfangen; Fehlerstatus hat sich geändert; …).

Ab Software Version 3.00 gibt es für jeden der beiden Typen ein erweitertes Format der Callback Funktion.

Hinweis:
Die Connect Control Callback Funktion hat ein anderes Format als die Callback Funktion für die anderen Ereignisse. Sie müssen in Ihrer Applikation immer darauf achten, dass Sie das richtige Format verwenden. Sie können nicht ein und dieselbe Callback Funktion für beide Typen verwenden!

Auch die erweiterten Formate der Callback Funktionen unterscheiden sich von den Standardformaten. Die erweiterten API-Funktionen müssen immer die Callback Funktion mit dem erweiterten Format angeben.

Achten Sie bitte auch darauf, dass PUBLIC vor den Callback Funktionen angegeben ist. Dies ist unter Microsoft Visual Studio als „__stdcall“ definiert.

Werden diese Hinweise nicht beachtet, dann werden zur Laufzeit Schutzverletzungen auftreten.

UcanConnectControlFkt

Syntax:

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

Bedeutung:
Diese Callback Funktion benachrichtigt das Anwendungsprogramm, wenn ein neues USB-CANmodul an den Rechner angesteckt oder ein schon vorhandenes USB-CANmodul abgezogen wird. Sie wird von der Funktion *UcanInitHwConnectControlEx()* bei der USBCAN-Library angemeldet und kann innerhalb der Applikation einen anderen Namen erhalten.

Parameter:

bEvent_p: Ereignis, welches aufgetreten ist.

USBCAN_EVENT_CONNECT = 6
USBCAN_EVENT_DISCONNECT = 7
USBCAN_EVENT_FATALDISCON = 8

dwParam_p: Zusatzparameter

Ist *bEvent_p* = 8, so gibt dieser Parameter das USB-CAN-Handle des abgezogenen Moduls an. Es werden von diesem Modul keine Nachrichten empfangen und es können keine Nachrichten gesendet werden. Das zugehörige USB-CAN-Handle ist ungültig. In den anderen Fällen ist dieser Parameter 0.

UcanConnectControlFktEx

Syntax:

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

Bedeutung:
Diese Callback Funktion benachrichtigt das Anwendungsprogramm, wenn ein neues USB-CANmodul an den Rechner angesteckt oder ein schon vorhandenes USB-CANmodul abgezogen wird. Sie wird von der Funktion *UcanInitHwConnectControlEx()* bei der USBCAN-Library angemeldet und kann innerhalb der Applikation einen anderen Namen erhalten.

Parameter:

dwEvent_p: Ereignis, welches aufgetreten ist.
- *USBCAN_EVENT_CONNECT = 6*
- *USBCAN_EVENT_DISCONNECT = 7*
- *USBCAN_EVENT_FATALDISCON = 8*

dwParam_p: Zusatzparameter
Ist dw*Event_p* = 8, so gibt dieser Parameter das USB-CAN-Handle des abgezogenen Moduls an. Es werden von diesem Modul keine Nachrichten empfangen und es können keine Nachrichten gesendet werden. Das zugehörige USB-CAN-Handle ist ungültig. In den anderen Fällen ist dieser Parameter 0.

pArg_p: Zusätzlicher Anwenderparameter, der bei der Funktion *UcanInitHwConnectControlEx()* als Parameter *pCallbackArg_p* angegeben wurde.

UcanCallbackFkt

Syntax:

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

Bedeutung:
Diese Callback Funktion benachrichtigt das Anwendungsprogramm, wenn bei einem initialisiertem USB-CANmodul ein Ereignis auftritt. Sie wird von der Funktion *UcanInitHardware()* bei der USBCAN-Library angemeldet und kann innerhalb der Applikation einen anderen Namen erhalten.

Parameter:

UcanHandle_p: USB-CAN-Handle, des USB-CANmoduls, bei dem das Ereignis aufgetreten ist. Dieses Handle wird mit der Funktion *UcanInitHardware()* zurückgegeben.

bEvent_p: Ereignis, welches aufgetreten ist.

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);
```

Bedeutung:
Diese Callback Funktion benachrichtigt das Anwendungsprogramm, wenn bei einem initialisiertem USB-CANmodul ein Ereignis auftritt. Sie wird von der Funktion *UcanInitHardwareEx()* bei der USBCAN-Library angemeldet und kann innerhalb der Applikation einen anderen Namen erhalten.

Parameter:

UcanHandle_p: USB-CAN-Handle, des USB-CANmoduls, bei dem das Ereignis aufgetreten ist. Dieses Handle wird mit der Funktion *UcanInitHardwareEx()* zurückgegeben.

dwEvent_p: Ereignis, welches aufgetreten ist.

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 Kanal, der das Ereignis ausgelöst hat.

USBCAN_CHANNEL_CH0	= 0
USBCAN_CHANNEL_CH1	= 1
USBCAN_CHANNEL_ANY	= 255

pArg_p: Zusätzlicher Anwenderparameter, der bei der Funktion *UcanInitHardwareEx()* als Parameter *pCallbackArg_p* angegeben wurde.

Beide Callback Funktionen müssen vorher initialisiert werden, bevor sie von der USBCAN-Library aufgerufen werden kann.

Alle Callback Funktionen müssen vorher initialisiert werden, bevor sie von der USBCAN-Library aufgerufen werden kann.

Die Ereignisse haben folgende Bedeutung:

USBCAN_EVENT_INITHW: 0x00
Das USB-CANmodul wurde erfolgreich initialisiert.
Der Parameter bChannel_p hat keine Bedeutung.

USBCAN_EVENT_INITCAN:.....0x01
Die CAN-Schnittstelle wurde erfolgreich initialisiert.
Der Parameter bChannel_p gibt den CAN-Kanal zurück, der initialisiert wurde.

USBCAN_EVENT_RECEIVE:.....0x02
Eine CAN-Nachricht wurde empfangen.
Der Parameter bChannel_p gibt den CAN-Kanal zurück, der als letztes von der Hardware empfangen wurde.

USBCAN_EVENT_STATUS: 0x03
Der Fehlerstatus im USB-CANmodul hat sich geändert.
Der Parameter bChannel_p gibt den CAN-Kanal zurück, dessen Fehlerstatus sich geändert hat.

USBCAN_EVENT_DEINITCAN: 0x04
Die CAN-Schnittstelle wurde heruntergefahren.
Der Parameter bChannel_p gibt den CAN-Kanal zurück, der gerade deinitialisiert wird.

USBCAN_EVENT_DEINITHW:.....0x05
Das USB-CANmodul wurde heruntergefahren.
Der Parameter bChannel_p hat keine Bedeutung.

USBCAN_EVENT_CONNECT:.....0x06
Ein neues USB-CANmodul wurde angesteckt.
Der Parameter *dwParam_p* hat keine Bedeutung.

USBCAN_EVENT_DISCONNECT: 0x07

Ein USB-CANmodul wurde abgezogen.
Der Parameter *dwParam_p* hat keine Bedeutung.

USBCAN_EVENT_FATALDISCON 0x08

Ein USB-CANmodul wurde im Zustand HW_INIT oder CAN_INIT vom Rechner abgezogen. Es kann Datenverlust aufgetreten sein.
Der Parameter *dwParam_p* enthält das USB-CAN-Handle des abgezogenen Moduls. Dieses Handle darf nicht mehr verwendet werden.

Hinweis:
Die Callback Funktionen sollten nicht die Funktionen der USBCAN-Library direkt rufen. Dies kann zu unerwünschten Effekten führen. Die beste Methode ist, im Hauptprogramm auf ein Ereignis zu warten (z.B. mit der Win32-Funktion *WaitForMultipleObjects()*) und dort nach Eintreten des Ereignisses die Funktionen der DLL zu rufen. Die Callback Funktionen setzen nur das entsprechende Signal (z.B. mit der Win32-Funktion *SetEvent()*).

Beispiel:

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

void main (void)
{
UCANRET bRet;

  // erste Callback Funktion initialisieren
  bRet = UcanInitHwConnectControl (UcanConnectControlFkt);

  // Fehler?
  if (bRet == USBCAN_SUCCESSFUL)
  {
    // auf Ereignis Warten
    // z.B. mit WaitForMultipleObjects(...)
    // entsprechend auf die Ereignisse reagieren:

    //case INIT:
      // USB-CANmodul mit USBCAN_ANY_MODULE öffnen und
      // zweite Callback Funktion initialisieren
      bRet = UcanInitHardware (&UcanHandle_g, USBCAN_ANY_MODULE,
          UcanCallbackFkt);

      // CAN-Schnittstelle initialisieren
      bRet = UcanInitCan (UcanHandle_g, 0x00, 0x14, 0xFFFFFFFFL,
          0x00000000L);

    //case RECV:
      // CAN-Nachricht lesen
      bRet = UcanReadCanMsg (UcanHandle_g, &CanRxMsg_g);
  }
  ...
}

void PUBLIC UcanConnectControlFkt (BYTE bEvent_p, DWORD
    dwParam_p)
{
UCANRET bRet;

  // welches Ereignis ist aufgetreten?
  switch (bEvent_p)
  {
    // neues USB-CANmodul angesteckt
    case USBCAN_EVENT_CONNECT:
      // Signal an main-Funktion senden, dass USB-CANmodul jetzt
      // jetzt initialisiert werden kann.
      // z.B. mit SetEvent(INIT)
      ...
      break;

    // USB-CANmodul abgezogen
    case USBCAN_EVENT_DISCONNECT:
      ...
      break;
  }
}
```

```
void PUBLIC UcanCallbackFkt (tUcanHandle UcanHandle_p,
  BYTE bEvent_p)
{
  // welches Ereignis ist aufgetreten?
  switch (bEvent_p)
  {
    // CAN-Nachricht empfangen
    case USBCAN_EVENT_RECEIVE:
      // signalisieren, dass CAN-Nachricht gelesen werden kann
      // z.B. mit SetEvent (RECV);
      break;

    // Fehlerstatus geändert
    case USBCAN_EVENT_STATUS:
      // signalisieren, dass CAN-Status gelesen werden kann
      // z.B. mit SetEvent (STATUS);
      break;
    ...
  }
}
```

2.4 Klassenbibliothek für .NET-Programmiersprachen

Um die USBCAN32.DLL auch in .NET-Sprachen wie Visual Basic .NET, Managed C++ und C# zu nutzen, wurde eine Wrapper-Klasse UcanDotNET.USBcanServer in VB .NET entwickelt. Diese Klasse liegt in einer eigenen Dynamik-Link-Library namens UCANDOTNET.DLL. Um die DLL in eigene Projekte einzubinden, sind folgende Schritte unter Visual Studio .NET notwenig: im Menü „Projekte“ den Eintrag „Verweis hinzufügen…“ auswählen. Im Dialog „Verweis hinzufügen“ auf „Durchsuchen…“ klicken und die Datei UCANDOTNET.DLL auswählen. Den Dialog mit „OK“ bestätigen.

Beispiel zur Erstellung eines Objektes der Klasse USBcanServer in Visual Basic .NET:

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

Hinweis:
Die Wrapper-Klasse liegt als Source-Code bei und kann vom Anwender angepasst werden, um neue Features der USBCAN32.DLL nachzurüsten oder andere Verbesserungen durchzuführen.

2.4.1 Methoden der Klasse USBcanServer

GetFwVersion

Syntax C#:
```
public int USBcanServer.GetFwVersion();
```

Syntax Visual Basic:
```
Public USBcanServer.GetFwVersion() as Integer
```

Verwendbarkeit:
HW_INIT ab Version 3.01

Bedeutung:
Gibt die Versionsnummer der Firmware des USB-CANmodul zurück.

Parameter:

Rückgabewert:	Firmwareversionsnummer als Integer mit folgendem 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
```

Verwendbarkeit:
DLL_INIT, HW_INIT, CAN_INIT ab Version 3.01

Bedeutung:
Gibt die Versionsnummer der USBCAN-Library zurück.

Parameter:

Rückgabewert:	Softwareversionsnummer als Integer mit folgendem 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
```

Verwendbarkeit:
DLL_INIT, HW_INIT, CAN_INIT ab Version 3.01

Bedeutung:
Initialisiert ein USB-CANmodul dessen Gerätenummer übereinstimmt.

Parameter:

bDeviceNr_p: Gerätenummer des USB-CANmoduls (0 – 254). Der Wert *USBCAN_ANY_MODULE* (= 255) sorgt dafür, dass das USB-CANmodul verwendet wird, welches zuerst gefunden wurde.

Rückgabewert: Fehlercode der Funktion.

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
```

Verwendbarkeit:
HW_INIT ab Version 3.01

Bedeutung:
Deinitialisiert den CAN-Kanal und ein USB-CANmodul, das zuvor mit den beiden Methoden *InitHardware()* oder *InitCan()* initialisiert wurde. Dabei gelangt die Software wieder in den Zustand DLL_INIT.

Parameter:

Rückgabewert: Fehlercode der Funktion.
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
```

Verwendbarkeit:
HW_INIT ab Version 3.01

Bedeutung:
Initialisiert einen CAN-Kanal des USB-CANmoduls. Für GW-001 und GW-002 kann nur der CAN-Kanal 0 initialisiert werden.

Parameter:

bChannel_p:	CAN-Kanal, der initialisiert werden soll. USBCAN_CHANNEL_CH0 für CAN-Kanal 0 USBCAN_CHANNEL_CH1 für CAN-Kanal 1
wBTR_p:	Baudratenregister BTR0 als high byte, Baudratenregister BTR1 als low byte
dwBaudrate_p:	Baudratenregister für Multiport CAN-to-USB und USB-CANmodul1/2
dwAMR_p:	Akzeptanzfiltermaske *(siehe Kapitel 2.3.5)*
dwACR_p:	Akzeptanzfiltercode
bMode_p:	Arbeitsmodus des CAN-Kanals
bOCR_p:	Output-Control-Register

Rückgabewert: Fehlercode der Funktion.

USBCAN_SUCCESSFUL
USBCAN_ERR_MAXINSTANCES
USBCAN_ERR_ILLHANDLE
USBCAN_ERR_RESOURCE
USBCAN_ERR_BUSY
USBCAN_ERR_IOFAILED
USBCAN_ERRCMD_NOTEQU
USBCAN_ERRCMD_REGTST
USBCAN_ERRCMD_ILLCMD

ResetCan

Syntax C#:
public byte USBcanServer.ResetCan(byte pbChannel_p)

Syntax Visual Basic:

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

Verwendbarkeit:
HW_INIT, CAN_INIT ab Version 3.01

Bedeutung:
Setzt den CAN-Controller eines CAN-Kanals im USB-CANmodul zurück (siehe auch Funktion *UcanResetCan()*). Für GW-001 und GW-002 kann nur CAN-Kanal 0 zurückgesetzt werden.

Parameter:

bChannel_p:	CAN-Kanal, der zurückgesetzt werden soll. USBCAN_CHANNEL_CH0 für CAN-Kanal 0 USBCAN_CHANNEL_CH1 für CAN-Kanal 1

Rückgabewert: Fehlercode der Funktion.

USBCAN_SUCCESSFUL
USBCAN_ERR_MAXINSTANCES
USBCAN_ERR_ILLHANDLE
USBCAN_ERR_CANNOTINIT
USBCAN_ERR_BUSY
USBCAN_ERR_IOFAILED
USBCAN_ERRCMD_NOTEQU
USBCAN_ERRCMD_ILLCMD

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
```

Verwendbarkeit:
HW_INIT, CAN_INIT ab Version 3.01

Bedeutung:
Gibt die erweiterten Hardwareinformationen eines USB-CANmoduls zurück. Für das Multiport CAN-to-USB 3004006 und das USB-CANmodul2 3204002/3204003 stehen zwei CAN-Kanäle je logisches Gerät zur Verfügung, wobei die Informationen jedes CAN-Kanals separat zurückgegeben werden.

Parameter:

pHwInfo_p: Adresse auf die erweiterte Hardware-informationsstruktur (*siehe UcanGetHardwareInfoEx2*).

pCanInfoCh0_p: Adresse auf die Informationsstruktur für CAN-Kanal 0.

pCanInfoCh1_p: Adresse auf die Informationsstruktur für CAN-Kanal 1.

Rückgabewert: Fehlercode der Funktion.

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
```

Verwendbarkeit:
HW_INIT, CAN_INIT ab Version 3.01

Bedeutung:
Gibt den Fehlerstatus eines bestimmten CAN-Kanals aus dem USB-CANmodul zurück.
Die Definition der Struktur *tStatusStruct* wird bei der Funktion *UcanGetStatus()* angegeben.

Parameter:

bChannel_p:	CAN-Kanal, dessen Status gelesen werden soll. USBCAN_CHANNEL_CH0 für CAN-Kanal 0 USBCAN_CHANNEL_CH1 für CAN-Kanal 1
pStatus_p:	Fehlerstatus des USB-CANmoduls.

Rückgabewert: Fehlercode der Funktion.

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
```

Verwendbarkeit:
CAN_INIT ab Version 3.01

Bedeutung:
Ändert nachträglich die Baudrateneinstellung eines bestimmten CAN-Kanals des USB-CANmoduls.

Parameter:

bChannel_p:	CAN-Kanal, dessen Baudrate gesetzt werden soll. USBCAN_CHANNEL_CH0 für CAN-Kanal 0 USBCAN_CHANNEL_CH1 für CAN-Kanal 1
wBTR_p:	Baudratenregister BTR0 als high byte, Baudratenregister BTR1 als low byte
dwBaudrate_p:	Baudraten Register für Multiport CAN-to-USB bzw. USB-CANmodul1/2

Rückgabewert: Fehlercode der Funktion.

USBCAN_SUCCESSFUL
USBCAN_ERR_MAXINSTANCES
USBCAN_ERR_ILLHANDLE
USBCAN_ERR_CANNOTINIT
USBCAN_ERR_BUSY
USBCAN_ERR_IOFAILED
USBCAN_ERRCMD_NOTEQU
USBCAN_ERRCMD_ILLCMD

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
```

Verwendbarkeit:
CAN_INIT ab Version 3.01

Bedeutung:
Ändert nachträglich die Akzeptanzfilterwerte für einen bestimmten CAN-Kanal des USB-CANmoduls.

Parameter:

bChannel_p:	CAN-Kanal, dessen Filterwerte gesetzt werden soll. USBCAN_CHANNEL_CH0 für CAN-Kanal 0 USBCAN_CHANNEL_CH1 für CAN-Kanal 1
dwAMR_p:	Akzeptanzfiltermaske *(siehe Kapitel 2.3.5)*
dwACR_p:	Akzeptanzfiltercode

Rückgabewert: Fehlercode der Funktion.

USBCAN_SUCCESSFUL
USBCAN_ERR_MAXINSTANCES
USBCAN_ERR_ILLHANDLE
USBCAN_ERR_CANNOTINIT
USBCAN_ERR_BUSY
USBCAN_ERR_IOFAILED
USBCAN_ERRCMD_NOTEQU
USBCAN_ERRCMD_ILLCMD

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
```

Verwendbarkeit:
CAN_INIT ab Version 3.01

Bedeutung:
Liest eine oder mehrere CAN-Nachrichten aus dem Zwischenpuffer. Es können CAN-Nachrichten von einem bestimmten CAN-Kanal gelesen werden.

Parameter:

pbChannel_p:	Referenz einer Variable mit dem CAN-Kanal von der die CAN-Nachrichten gelesen werden sollen. USBCAN_CHANNEL_CH0 für CAN-Kanal 0 USBCAN_CHANNEL_CH1 für CAN-Kanal 1 Wird USBCAN_CHANNEL_ANY angegeben, dann schreibt diese Funktion die Kanalnummer auf diese Variable, von der CAN-Nachrichten empfangen wurden.
pCanMsg_p:	ein Array aus CAN-Nachrichtenstrukturen.
pdwCount_p:	Die Funktion schreibt nach dem Lesen die Anzahl der CAN-Nachrichten auf diese Variable, die tatsächlich gelesen wurde.

Die Definition der Struktur *tCanMsgStruct* wird bei der Funktion *UcanReagCanMsg()* beschrieben.

Rückgabewert: Fehlercode der Funktion.

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,
     Optional ByRef dwCount_p As Integer) As Byte
```

Verwendbarkeit:
CAN_INIT ab Version 3.01

Bedeutung:
Sendet eine oder mehrere CAN-Nachrichten über einen bestimmten CAN-Kanal des USB-CANmoduls.

Parameter:

bChannel_p:	CAN-Kanal, über den gesendet werden soll. USBCAN_CHANNEL_CH0 für CAN-Kanal 0 USBCAN_CHANNEL_CH1 für CAN-Kanal 1
pCanMsg_p:	ein Array von CAN-Nachrichtenstrukturen für den Empfang der CAN-Nachrichten.
dwCount_p:	Nach dem Aufruf der Funktion erhält diese Variable die Anzahl der CAN-Nachrichten, die erfolgreich gesendet werden konnten. Der Wert kann kleiner sein als die Anzahl der Elemente im Array, wenn nicht alle CAN-Nachrichten in den Sendepuffer passten. In diesem Fall gibt die Funktion den Warnung USBCAN_WARN_TXLIMIT zurück.

Die Definition der Struktur *tCanMsgStruct* wird bei der Funktion *UcanWriteCanMsg()* beschrieben.

Rückgabewert: Fehlercode der Funktion.

USBCAN_SUCCESSFUL
USBCAN_ERR_MAXINSTANCES
USBCAN_ERR_ILLHANDLE
USBCAN_ERR_CANNOTINIT
USBCAN_ERR_ILLPARAM
USBCAN_ERR_DLL_TXFULL

Beispiel in 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 wRecvdMsgCount_p As Short, _
     ByRef wSentMsgCount_p As Short) As Byte
```

Verwendbarkeit:
CAN_INIT ab Version 3.01

Bedeutung:
Liest die Zählerstände eines bestimmten CAN-Kanals aus.

Parameter:

bChannel_p: CAN-Kanal, dessen Zähler ausgelesen werden sollen.
USBCAN_CHANNEL_CH0 für CAN-Kanal 0
USBCAN_CHANNEL_CH1 für CAN-Kanal 1

wRecvdMsgCount_p: Zähler der empfangenen Nachrichten.

wSentMsgCount_p: Zähler der gesendeten Nachrichten.

Rückgabewert: Fehlercode der Funktion.

USBCAN_SUCCESSFUL
USBCAN_ERR_MAXINSTANCES
USBCAN_ERR_ILLHANDLE
USBCAN_ERR_CANNOTINIT
USBCAN_ERR_BUSY
USBCAN_ERR_IOFAILED
USBCAN_ERR_ILLCMD
USBCAN_WARN_NOTEQU

GetCanMessage

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
```

Verwendbarkeit:
CAN_INIT ab Version 3.01

Bedeutung:
Gibt die Nachricht als String für einen angegebenen CAN-Statuscode zurück.

Parameter:
wCanStatus_p: CAN-Statuscode

Rückgabewert: String mit entsprechender Nachricht

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
```

Verwendbarkeit:
CAN_INIT ab Version 3.01

Bedeutung:
Gibt die Nachricht als String für die angegebenen BTR Registerwerte zurück.

Parameter:

bBTR0_p:	Baudratenregister 0
bBTR1_p:	Baudratenregister 1

Rückgabewert: String mit entsprechender Nachricht

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
```

Verwendbarkeit:
CAN_INIT ab Version 3.01

Bedeutung:
Gibt die Nachricht als String für den angegebenen Baudwert zurück.

Parameter:
dwBTR_p: 32 bit Baudratenregister

Rückgabewert: String mit entsprechender Nachricht

2.4.2 Events der Klasse USBcanServer

Die Callback Ereignisse der USBCAN32.DLL werden mit der USBcanServer Klasse als .NET Events an die Applikation weitergegeben.

CanMsgReceivedEvent

Syntax C#:
```
public event USBcanServer.CanMsgReceivedEvent(
     byte bChannel_p);
```

Syntax Visual Basic:
```
Public Event USBcanServer.CanMsgReceivedEvent( _
     ByVal bChannel_p As Byte)
```

Bedeutung:
Eine neue CAN-Nachricht ist eingetroffen.

Parameter:

bChannel_p:	CAN-Kanal, wo die Nachricht eingetroffen ist. USBCAN_CHANNEL_CH0 für CAN-Kanal 0 USBCAN_CHANNEL_CH1 für CAN-Kanal 1 USBCAN_CHANNEL_ANY für beliebigen Kanal

InitHwEvent

Syntax C#:
```
public event USBcanServer.InitHwEvent();
```

Syntax Visual Basic:
```
Public Event USBcanServer.InitHwEvent()
```

Bedeutung:
USB-CANmodul wurde initialisiert.

InitCanEvent

Syntax C#:
```
public event USBcanServer.InitCanEvent(byte bChannel_p);
```

Syntax Visual Basic:
```
Public Event USBcanServer.InitCanEvent( _
     ByVal bChannel_p As Byte)
```

Bedeutung:
Angegebener CAN-Kanal wurde initialisiert.

Parameter:

bChannel_p:	CAN-Kanal, der initialisiert wurde. USBCAN_CHANNEL_CH0 für CAN-Kanal 0 USBCAN_CHANNEL_CH1 für CAN-Kanal 1

StatusEvent

Syntax C#:
```
public event USBcanServer.StatusEvent(byte bChannel_p);
```

Syntax Visual Basic:
```
Public Event USBcanServer.StatusEvent(ByVal bChannel_p As Byte)
```

Bedeutung:
Fehlerstatus hat sich vom angegebenen CAN-Kanal geändert.

Parameter:

bChannel_p:	CAN-Kanal, dessen Status sich geändert hat. USBCAN_CHANNEL_CH0 für CAN-Kanal 0 USBCAN_CHANNEL_CH1 für CAN-Kanal 1

Beispiel für einen Event-Handler für das 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)
```

Bedeutung:
Der CAN-Kanal wurde heruntergefahren.

Parameter:

bChannel_p: CAN-Kanal, dessen Status sich geändert hat.
USBCAN_CHANNEL_CH0 für CAN-Kanal 0
USBCAN_CHANNEL_CH1 für CAN-Kanal 1

DeinitHwEvent

Syntax C#:
```
public event USBcanServer.DeinitHwEvent()
```

Syntax Visual Basic:
```
Public Event USBcanServer.DeinitHwEvent()
```

Bedeutung:
Das USB-CANmodul wurde heruntergefahren.

ConnectEvent

Syntax C#:
```
public static event USBcanServer.ConnectEvent();
```

Syntax Visual Basic:
```
Public Shared Event USBcanServer.ConnectEvent()
```

Bedeutung:
Ein neues USB-CANmodul wurde an einen USB-Port angeschlossen.

DisconnectEvent

Syntax C#:
```
public static event USBcanServer.DisconnectEvent();
```

Syntax Visual Basic:
```
Public Shared Event USBcanServer.DisconnectEvent()
```

Bedeutung:
Ein schon deinitialisiertes USB-CANmodul wurde vom USB-Port abgezogen.

FatalDisconnectEvent

Syntax C#:

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

Syntax Visual Basic:

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

Bedeutung:
Ein USB-CANmodul wurde ohne Deinitialisierung vom USB-Port abgezogen.

3 Softwareunterstützung unter Linux

Das Softwarepaket SO-1068 enthält einen Treiber für den Linux Kernel 2.6 im Source für eine x86-kompatiblen CPU, eine Library mit den API-Funktionen sowie eine Demoapplikation. Dieses Handbuch beschreibt den Treiber ab Version 2.00.

3.1 Installation des Treibers unter Linux

Es gibt 2 Voraussetzungen, den Treiber unter Linux installieren zu können:

1) Sie müssen als „super user“ angemeldet sein.
2) Die Kernel-Sourcen müssen installiert sein, um den Treiber neu übersetzen zu können.

Entpacken Sie das ausgelieferte Archiv in einem Verzeichnis und führen Sie das Shell-Skript **initdriver.sh** aus. Sie werden gefragt, ob der Treiber neu übersetzt werden soll. Beantworten Sie diese Frage mit „y“ für yes, da der Treiber nicht als Binary ausgeliefert wird. Das liegt daran, dass Linux unter vielen verschiedenen CPUs lauffähig ist.

Zusätzlich ist der Linux-Treiber für das USB-CANmodul nur für einige Linux Kernel Versionen getestet worden (derzeit unter 2.6.13, 2.6.16 und 2.6.18 mit der SUSE Distribution, 2.6.24 mit der Ubuntu Distribution, 2.6.25 auf einem 64 Bit Prozessor mit der SUSE Distribution). Es ist nicht ausgeschlossen, dass sich der Treiber unter neueren Kernel Versionen nicht fehlerfrei übersetzen lässt, wenn Änderungen in neueren Kernel Versionen durchgeführt wurden. In diesen Fällen führen Sie bitte die Änderungen in den Sourcen des Treibers für das USB-CANmodul selbst durch.

Nach dem erfolgereichen Übersetzen des Treibers werden Sie gefragt, ob der Debug-Modus des Kernel Treibers eingeschaltet werden soll. In diesem Modus werden die Debug-Ausgaben aus dem Treiber nach *printk()* umgeleitet, so dass diese Ausgaben auch in */var/log/messages*

sichtbar sind. Sie sollten zunächst diese Frage mit „n“ für no beantworten.

Der Kernel-Treiber unter Linux ab der Version V2.00 besteht aus zwei Dateien. Das sind **usbcanlx.ko** und **usbcan.ko**. Erstere Datei beinhaltet einen Firmware-Loader für die USB-CANmodule. Die zweite Datei stellt die Kommunikation zwischen der USBCAN-Library und dem USB-CANmodul her.

Hinweis:
Nach dem nächsten Booten des Linux-Betriebssystems müssen Sie das Shell-Skript „initdriver.sh“ erneut aufrufen, damit der Treiber geladen wird. Da das automatische Laden des Treibers (ohne Skript) abhängig von der verwendeten Linux-Distribution und Kernel-Version ist, wurde darauf verzichtet, diese Option mit anzubieten. Informieren Sie sich daher bei Ihrer Linux-Distribution, welche Möglichkeiten für ein automatisches Laden von USB-Treibern zur Verfügung stehen.

Ab der Treiberversion V2.02 führt der Loader auch einen Firmware-Update für die sysWORXX USB-CANmodule automatisch durch.

3.2 Funktions-API unter Linux

In der Treiberversion V2.00 befinden sich die API Funktionen unter Linux in zwei statischen Libraries **Host.Lin/library/libusbcanr.a** (für den Release Modus) und **Host.Lin/library/libusbcand.a** (für den Debug-Modus). Diese entsprechen denen unter Windows (*siehe Kapitel 2.3*). Es sind jedoch nicht alle Funktionen verfügbar. *Tabelle 17* listet alle verfügbaren Funktionen unter Linux auf.

Ab Treiberversion V2.02 wird statt der statischen Libraries zwei dynamische Libraries ausgeliefert. Bei der Ausführung des Shell-Skript **initdriver.sh** werden diese Libraries in das Verzeichnis /usr/local/lib kopiert und jeweils ein symbolischer Link angelegt. Für den Release-Modus wird **libusbcan.so** und für Debaug-Ausgaben wird **libusbcand.so** verwendet.

Verfügbare Funktionen	**Nicht verfügbare Funktionen**
UcanGetVersionEx()	*UcanGetVersion() (veraltet)*
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()

Tabelle 17: Verfügbare und nicht-verfügbare Funktionen unter Linux

Folgende API Funktionen unterscheiden sich von denen unter Windows:

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

Die Funktion *UcanInitHardware()* bietet keine Möglichkeit zur Verwendung einer Callback Funktion. Daher ist der dritte Parameter als „reserviert“ gekennzeichnet.

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

Die Funktion *UcanSetDeviceNr()* kann dazu verwendet werden, um in das USB-CANmodul eine neue Gerätenummer zu programmieren.

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

Mit dieser Funktion stellen Sie den LOG-Level für die Debug-Ausgaben ein. Voreinstellung für den LOG-Level ist der Wert 0xE0000803 für die Library libusbcand.a. Lesen Sie hierzu auch das *Kapitel 3.3* sowie die Beschreibung der Funktion *UcanSetDebugMode()* im *Kapitel 2.3.2* für nähere Informationen.

3.3 Erstellung von Debug-Informationen

Sollten Probleme mit dem Treiber auftreten, dann ist es oft sehr hilfreich, wenn Sie uns eine Log-Datei mit den Debug-Informationen der USBCAN-Library zusenden. Unter Linux müssen Sie statt libusbcanr.a die Library libusbcand.a (bzw. libusbcand.so) einbinden, um die Debug-Ausgaben zu aktivieren. Diese Debug-Ausgaben werden ab der Version 2.00 der Library auf printf() umgeleitet und werden damit auf der Standard-Ausgabe erscheinen. Kopieren Sie daher diese Ausgaben in eine Datei und senden Sie uns diese zu.

Mit der Funktion *UcanSetDebugMode()* geben Sie an, welche Debug-Informationen ausgegeben werden sollen.

Oftmals können wir das Problem aus diesen Debug-Informationen der USBCAN-Library nicht erkennen. In diesem Fall sollten Sie uns auch die Debug-Ausgaben des Kernel-Treibers zusenden. Beantworten Sie daher die Frage nach dem Debug Modus im Skript initdriver.sh mit „y“ für yes. Sollten Sie nicht mit diesem Skript arbeiten, um den Kernel-Treiber zu laden, dann müssen Sie beim Laden des Treibers den Parameter „debug“ auf 1 setzen:

```
insmod usbcanlx.ko debug=1
insmod usbcan.ko debug=1
```

Die Debug-Ausgaben dieser Module werden bei eingeschalteter Option nach *printk()* umgeleitet, so dass Sie die Ausgaben in der Datei */var/log/messages* finden.

4 Softwareunterstützung unter Windows CE

Das Software-Paket SO-1091 enthält einen Gerätetreiber für Windows CE, einer USBCAN-Library als DLL und eine Demo-Applikation im Source für Microsoft eMbedded Visual C++ 4.0. Folgende Windows CE Versionen und CPU-Typen wurden mit diesem Treiber getestet:

Windows CE Version	CPU Typ	Getestet auf CPU
5.0	ARMV4I	Intel PXA255 Intel PXA270

Tabelle 18: Getestete Windows CE Versionen und CPU-Typen

Auf Anfrage kann auch eine Portierung des Treibers auf andere CPU-Typen vorgenommen werden.

4.1 Installation des Treibers unter Windows CE

Der Gerätetreiber befindet sich nach dem Entpacken im Verzeichnis *Driver\XXX*, wobei XXX den Typ der CPU angibt. Für den CPU-Typ ARMV4I befindet sich der Treiber also im Verzeichnis *Driver\ARMV4I* und trägt für alle CPU-Typen den Namen **UsbCanDrv.dll**.

Dieser Treiber unterstützt in der derzeit aktuellen Version die folgenden USB-CANmodule:

GW-002, USB-CANmodul1 (3204000 und 3204001) sowie alle Derivate des USB-CANmodul2 (3204002, 3204003, …).

Kopieren Sie diese Datei in das Verzeichnis *\Windows* bevor Sie das USB-CANmodul anstecken. Verbinden Sie nun das USB-CANmodul über ein geeignetes USB-Kabel mit Ihrem Windows CE Gerät. Wenn Sie das USB-CANmodul das erste Mal anstecken, dann erscheint ein Fenster „Unidentified USB Device“. Geben Sie dort den Namen des Treibers „usbcandrv“ an (zwischen Groß- und Kleinschreibung wird nicht unterschieden). Nun können Sie mit einer Applikation auf das USB-CANmodul zugreifen. Verwenden Sie dazu zum Beispiel unser

Demo **ConsoleDemo.exe**, welches mit 1MBit/s genau 100 CAN-Nachrichten sendet und währenddessen empfangene CAN-Nachrichten ausgibt.

4.2 Funktions-API unter Windows CE

Die API-Funktionen für das USB-CANmodul sind in der **UsbCanCE.dll** enthalten. Sie befindet sich im Verzeichnis *Lib\XXX*, wobei XXX den Typ der CPU angibt. In dieser DLL sind alle Funktionen enthalten, die auch in der USBCAN32.DLL unter Windows 2000/XP/Vista enthalten sind – mit Ausnahme von drei Funktionen:

UcanInitHwConnectControl()
UcanInitHwConnectControlEx()
UcanDeinitHwConnectControl()

Entnehmen Sie bitte die Beschreibung der vorhandenen API-Funktionen aus dem *Kapitel 2.3*.

Es existieren weitere Unterschiede zur USBCAN32.DLL:

- Unter Windows CE werden nur bis zu 9 USB-CANmodule gleichzeitig an einem Rechner unterstützt.
- Die Funktion *UcanGetVersionEx()* kann nur mit den Parametern kVerTypeUserDll, kVerTypeUserLib und kVerTypeSysDrv verwendet werden.
- Der Aufruf von *UcanDeinitCan()* bzw. *UcanDeinitCanEx()* kann bis zu zwei Sekunden in Anspruch nehmen.

4.3 Erstellung von Debug-Informationen

Sollten Probleme mit dem Treiber auftreten, dann ist es oft sehr hilfreich, wenn Sie uns eine Log-Datei mit den Debug-Informationen der USBCAN-Library zusenden. Unter Windows CE ist es momentan nur möglich, dieses Feature mit der Funktion *UcanSetDebugMode()* einzuschalten. Lesen Sie dazu bitte das *Kapitel 2.3.2*.

Index

L

M

N

P

S

T

U

V

W

Dokument: USB-CANmodul
Dokumentnummer: **L-487d_21, Ausgabe Oktober 2008**

Wie würden Sie dieses Handbuch verbessern?

Haben Sie in diesem Handbuch Fehler entdeckt? Seite

Eingesandt von:
Kundennummer:

Name:

Firma:

Adresse:

Einsenden an:

SYS TEC electronic GmbH
August-Bebel-Str. 29
D-07973 Greiz, Germany
Fax : +49 (0) 3661 62 79 99

Published by

SYS TEC ELECTRONIC

Ordering No. L-487d_21
Printed in Germany