

## Application Note

### Contents

---

1	Übersicht .....	1
2	Programmierung eigener Anwendungen mit VCI .....	2
2.1	Allgemeiner Ablauf einer VCI Anwendung .....	2
2.2	Delphi Headerdateien.....	3
2.3	Callbacks vs. WM Handler.....	3
3	Boardswahl .....	3
4	VCI Funktionen in der Initialisierungsphase der Applikation .....	4
4.1	VCI2_PrepareBoard.....	4
4.2	VCI_InitCan .....	4
4.3	VCI_ConfigQueue.....	5
4.4	VCI_AssignRxQueObj.....	5
4.5	VCI_StartCan .....	5
4.6	Programmlisting einer Standardinitialisierung .....	5
5	VCI Funktionen in der Arbeitsphase der Applikation .....	6
5.1	VCI_ReadQueObj.....	6
5.2	Receive WM Handler .....	6
5.3	VCI_CAN_OBJ Struktur .....	7
5.4	VCI_TransmitObj .....	8
6	VCI Funktionen bei der Beendigung der Applikation .....	8
7	Further Literature .....	8
8	Contact Information .....	8

---

## 1 Übersicht

VCI (Virtual CAN Interface) ist ein universelles Treiberpaket für alle PC/CAN-Schnittstellenkarten der IXXAT Automation GmbH. Es wird gemeinsam mit den CAN-Karten ausgeliefert und ist für die Betriebssysteme Windows 9x/NT/2000/XP erhältlich. Durch das VCI wird eine einheitliche Programmierschnittstelle (API) für die gängigsten Programmiersprachen in Form einer DLL (Dynamic Link Library) bereitgestellt.

In diesem Dokument soll die Verwendung des VCI unter Delphi erläutert werden. Es wird ein typischer Anwendungsfall mit einer Sende- und einer Empfangsqueue basierend auf Windows Messages beschrieben. Die hier aufgeführten Programmierbeispiele sind für Delphi 4 bis Delphi 7 geeignet.

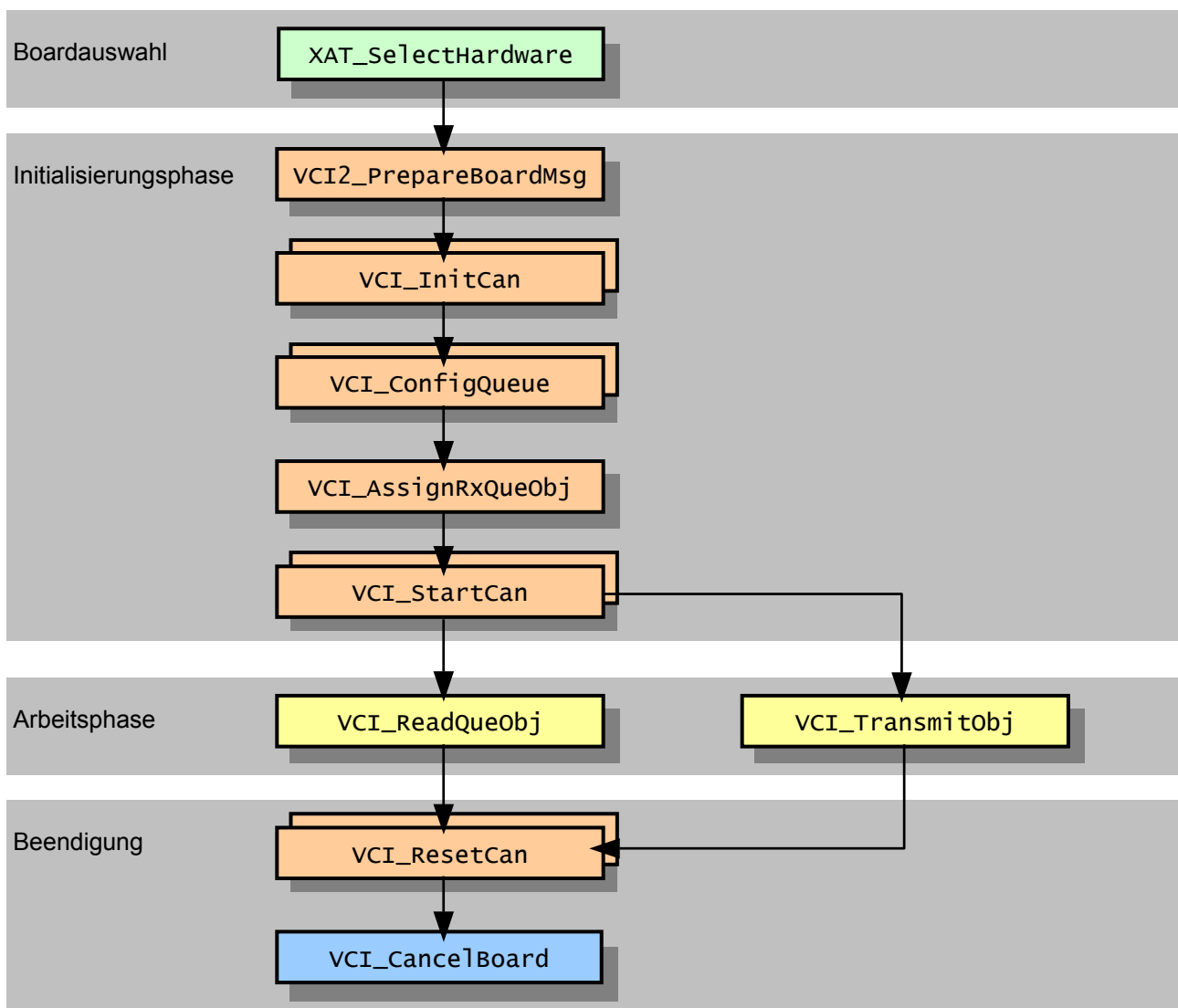
## 2 Programmierung eigener Anwendungen mit VCI

VCI stellt einen umfassenden Satz von Funktionen zur Programmierung der CAN Controller und zur Durchführung der CAN Kommunikation mit anderen Busteilnehmern bereit. Die breite Parametrierbarkeit der VCI Funktionen erlaubt den hochflexiblen Einsatz der Programmierbibliothek in allen CAN Anwendungsfeldern.

Alle Funktionen sind nach der C \_\_stdcall Aufrufkonvention implementiert. Für Rückmeldungen des API an die Anwendung kommen nach Wahl des Programmierers entweder Callbacks oder Windows Messages zum Einsatz. Einen umfassenden Überblick über alle Aspekte der VCI Programmierung bietet das VCI-Programmierhandbuch [2].

### 2.1 Allgemeiner Ablauf einer VCI Anwendung

Zum leichten Einstieg ist im folgenden Bild die Reihenfolge der VCI Funktionsaufrufe in einem typischen Einsatzfall aufgezeigt. Dieser Ablauf ist zunächst unterteilt in Boardauswahl, Initialisierungsphase, Arbeitsphase und Beendigung.



## 2.2 Delphi Headerdateien

Der Delphi Header VCI2.pas enthält alle VCI Funktionen. Zur Auswahl der CAN Karte und zur Abfrage der Kartenmerkmale und -fähigkeiten ist der Header XatDyn1.pas einzubinden.

## 2.3 Callbacks vs. WM Handler

Im typischen Einsatzfall wird VCI mit einer Empfangs- und einer Sendequelle betrieben. Für Delphi Anwendungen empfiehlt es sich, auf Callbacks zu verzichten, und stattdessen eigene Windows Messages zu definieren. Da Delphi eine oberflächenorientierte ereignisgesteuerte Programmierung bedingt, müßten die asynchronen Callbacks entweder - wie bei der Threadprogrammierung - synchronisiert, oder die Daten in einem gemeinsamen Speicherbereich mit verriegeltem Zugriff gepuffert werden. Beides ist sowohl aufwendig zu implementieren als auch ressourcenintensiv. Zudem arbeiteten die Callbacks in VCI-internen Threadkontexten, wobei die Threadverarbeitung solange verzögert wird, bis der Callback-Aufruf aus der Client-Anwendung zurückkehrt.

Durch die Verwendung eigener Windows Messages kommen die CAN Daten bereits in dem Formular an, wo sie auch verarbeitet werden, und das völlig synchron zu den übrigen Ereignissen. Im entsprechenden Message Handler kann unmittelbar auf die Formularelemente zugegriffen werden.

## 3 Boardauswahl

Zur Initialisierung des CAN Interfaceboards werden der Kartentyp und die rechner spezifische Kartenidentifikation benötigt. Beide Informationen können unter Verwendung der sog. IXXAT Registryfunktionen bestimmt werden. Dazu muß der XatDyn1.pas Header eingebunden und mit der Befehlsfolge LoadXatLib; MapXatDialogs; XAT\_SelectHardware() der Standard IXXAT Hardwareauswahldialog geöffnet werden. Im Hardwareauswahldialog kann der Benutzer dann die zu verwendende Karte selektieren. Ihre Haupteigenschaften werden in der Struktur TXAT\_BoardCFG zurückgeliefert. Die Elemente dieser Struktur, insbesondere board\_no und board\_type, werden für die Initialisierung des CAN Interfaceboards benötigt; bei einigen Karten (wie z.B. [can@net](#)) ist auch das Strukturelement sz\_CardAddString notwendig. Soll eine Anwendung immer das gleiche CAN Board verwenden, so empfiehlt es sich den Inhalt der TXAT\_BoardCFG Struktur in den persistenten Anwendungsdaten zu hinterlegen, und auf die Boardauswahl bei den nachfolgenden Programmläufen zu verzichten.

Hier zusammenfassend die Befehlsfolge zur Boardauswahl:

```
var
  sBoard : TXAT_BoardCFG;           // Local allocation of Board configuration record
begin
  // Dynamically load IXXAT Registry functions
  if not(Assigned(XAT_SelectHardware)) then
    if (LoadXatLib) then
      if (1 <> MapXatDialogs) then
        exit;

  // Open up standard hardware dialog
  sBoard.board_no := 0;
  if (1 = XAT_SelectHardware(self.Handle, @sBoard)) then
    begin
      { Save Board configuration data sBoard persistent or programglobal }
    end;

  // Unload IXXAT Registry functions
  FreeXatLib;
end;
```

## 4 VCI Funktionen in der Initialisierungsphase der Applikation

### 4.1 VCI2\_PrepareBoard

Sobald die Karte eindeutig identifiziert ist, kann die Initialisierungsphase durchlaufen werden. Zuerst wird das betreffende CAN Board für die Anwendung geöffnet und belegt. Dazu dient die mit Callbacks arbeitende Funktion VCI2\_PrepareBoard. Da bei Delphi jedoch mit Windows Messages gearbeitet werden soll, ist die Geschwisterfunktion VCI2\_PrepareBoardMsg zu verwenden. Diese hat folgende Syntax:

```
function VCI2_PrepareBoardMsg ( board_type      : VCI_BOARD_TYPE;  
                               board_index     : word;  
                               s_addinfo       : PChar;  
                               b_addLength     : Byte;  
                               fp_puts        : VCI_t_PutS;  
                               msg_rx_int_hdlr: Cardinal;  
                               fp_exc_hdlr    : VCI_t_UsrExcHdlr;  
                               ap1_handle     : Hwnd ): Integer;
```

board\_type, board\_index und s\_addinfo sind Parameter zur Kartenidentifikation, die direkt aus der Struktur TXAT\_BoardCFG der Boardauswahl übernommen werden können. b\_addLength ist die Bytelänge der nullterminierten Zeichenkette s\_addinfo und somit ebenfalls problemlos zu bestimmen. fp\_puts ist ein Funktionszeiger auf eine optionale Callbackfunktion zur Protokollierung der VCI Initialisierung. Sie soll hier nicht weiter betrachtet werden. Der nächste Parameter mit dem Bezeichner msg\_rx\_int\_hdlr ist tatsächlich der Windowsbotschaftsidentifizierer, der zur Empfangssignalisierung an das Anwendungsfenster geschickt werden soll, sobald ein CAN Telegramm empfangen wurde. Hier können Sie nach Ihren Vorstellungen einen Offsetwert zu WM\_USER angeben. Im folgenden Parameter fp\_exc\_hdlr kann wiederum eine Callbackfunktion angegeben werden, die bei einem schwerwiegenden Fehler während der VCI Initialisierungsphase aufgerufen wird. Dieser ist insbesondere am Beginn der Implementierung nützlich, da ein Fehlerbeschreibungstext übergeben wird. Ein Implementierungsbeispiel für eine passende Delphi Funktion folgt im weiteren Verlauf. Als letzter Parameter ist das Handle des Fensters anzugeben, an das die VCI Empfangsbotschaft geschickt werden soll.

Bis auf die Werte zur Kartenidentifikation sind alle Funktionsparameter optional. Selbst als Windowsbotschaftsidentifizierer und als Fensterhandle kann 0 angegeben werden. In diesem Fall müsste die VCI Empfangsqueue durch die Anwendung gepollt werden.

Der Rückgabewert der Funktion ist das Handle des betreffenden CAN Boards, auch als board\_hdl bezeichnet. Im Fehlerfall wird ein VCI Fehlercode mit Wert < 0 zurückgegeben.

Es folgt ein einfaches Implementierungsbeispiel für den VCI Exceptionhandler fp\_exc\_hdlr:

```
procedure VCI_UsrExcHdlr (func_num: VCI_FUNC_NUM; err_code: Integer; ext_err: word;  
                          s: PChar); cdecl;  
begin  
  Application.MessageBox (s, 'VCI_UsrExcHdlr', MB_TOPMOST);  
end;
```

Zu beachten ist hierbei, daß die Funktion nicht als member eines Delphi Objektes, sondern als lokale Funktion der Unit definiert werden muß.

### 4.2 VCI\_InitCan

Als zweiten Schritt der VCI Initialisierung ist der gewünschte CAN Controller zu parametrieren. Dazu dient die Funktion VCI\_InitCan. Diese hat folgende Syntax:

```
function VCI_InitCan ( board_hdl : word;  
                       can_num   : Byte;  
                       bt0       : Byte;  
                       bt1       : Byte;  
                       const mode: Byte ): Integer;
```

board\_hdl identifiziert das mittels VCI\_PrepareBoardMsg belegte CAN Interface Board und wird von eben jener Funktion zurückgeliefert. Unter can\_num wird der gewünschte CAN Controller auf dem Board angegeben. Die auf dem Board vorhandenen CAN Controller sind beginnend mit 0 fortlaufend durchnummeriert. In den beiden Parametern bt0 und bt1 werden die Werte der Bittiming-Register des CAN Controllers angegeben - VCI2.pas enthält bereits Konstanten für die Programmierung der geläufigen Baudraten, z.B. VCI\_1000KB\_BT0 und VCI\_1000KB\_BT1. Im letzten Parameter der Funktion wird die Betriebsart des CAN Controllers angegeben. Hier sind drei verschiedene Werte möglich, die ebenfalls als Konstanten vordefiniert

sind: VCI\_11B für standard Identifier und VCI\_29B für extended Identifier. Mischbetrieb wird von VCI2 nicht unterstützt!

Der Rückgabewert der Funktion ist ein VCI Fehlercode. Im Erfolgsfall wird VCI\_OK, im Fehlerfall ein Wert < 0 zurückgeliefert.

### 4.3 VCI\_ConfigQueue

Im nächsten Schritt ist die VCI Empfangsqueue und ggf. die VCI Sendequeue einzurichten. Dies erfolgt mit der Funktion VCI\_ConfigQueue. Sie hat folgende Syntax:

```
function VCI_ConfigQueue ( board_hdl      : word;  
                          can_num       : Byte;  
                          que_type      : Byte;  
                          que_size     : word;  
                          int_limit    : word;  
                          int_time     : word;  
                          ts_res       : word;  
                          var p_que_hdl : word ) : Integer;
```

Die Parametrierung der VCI Queues ist eine diffizile Angelegenheit. Für Standardanwendungen hat sich folgende Konfiguration bewährt:

```
VCI_ConfigQueue (m_hXatBoard, 0, VCI_RX_QUE, 100, 1, 100, 100, m_hRxQueue);
```

bzw.

```
VCI_ConfigQueue (m_hXatBoard, 0, VCI_TX_QUE, 100, 0, 0, 0, m_hTxQueue);
```

Entscheidend ist der letzte Parameter p\_que\_hdl, denn hier trägt VCI den Queuehandle ein, der die betreffende Queue eindeutig identifiziert.

Der Rückgabewert der Funktion ist ein VCI Fehlercode. Im Erfolgsfall wird VCI\_OK, im Fehlerfall ein Wert < 0 zurückgeliefert.

### 4.4 VCI\_AssignRxQueObj

Nun ist die Initialisierung des VCI abgeschlossen. Sie werden jedoch noch nichts empfangen, da die Filterung der Empfangsqueue standardmäßig so eingestellt ist, daß alle Identifier geblockt sind. Deshalb ist in einem weiteren Schritt die Empfangsqueue teilweise oder vollständig freizuschalten. Dies geschieht mit der Funktion VCI\_AssignRxQueObj. Um alle CAN Telegramme zu empfangen, verwenden Sie die folgende Anweisung:

```
VCI_AssignRxQueObj (m_hXatBoard, m_hRxQueue, VCI_ACCEPT, 0, 0);
```

### 4.5 VCI\_StartCan

Als letzten Schritt, der bereits den Übergang zur Arbeitsphase markiert, starten Sie den soeben parametrisierten CAN Controller. Dazu gibt es die Funktion VCI\_StartCan:

```
function VCI_StartCan ( board_hdl : word;  
                      can_num  : Byte ) : Integer;
```

### 4.6 Programmlisting einer Standardinitialisierung

Hier zusammenfassend die Befehlsfolge der VCI Initialisierungsphase in einer typischen Anwendung:

```
var  
  res : integer;  
  
begin  
  // Open IXXAT CAN Board described in m_sBoard  
  // On CAN message reception, send WM_VCIRX to ourselves  
  // For VCI errors, use callbackfunction named VCI_UsrExChdlr  
  m_hXatBoard := VCI2_PrepareBoardMsg (m_sBoard.board_type, m_sBoard.board_no,  
                                     m_sBoard.sz_CardAddString,  
                                     StrLen(m_sBoard.sz_CardAddString), nil,  
                                     WM_VCIRX, VCI_UsrExChdlr, self.Handle);  
  
  // Parameterisation of first CAN controller  
  res := 0;
```

```
if (0 <= m_hXatBoard) then
    res := VCI_InitCan (m_hXatBoard, 0, VCI_125KB_BT0, VCI_125KB_BT1, VCI_11B);

    // Configuration of receive queue
    if (VCI_OK = res) then
        res := VCI_ConfigQueue (m_hXatBoard, 0, VCI_RX_QUE, 100, 1, 100, 100, m_hRxQueue);
    if (VCI_OK = res) then
        res := VCI_AssignRxQueObj (m_hXatBoard, m_hRxQueue, VCI_ACCEPT, 0, 0);

    // Configuration of transmit queue
    if (VCI_OK = res) then
        res := VCI_ConfigQueue (m_hXatBoard, 0, VCI_TX_QUE, 100, 1, 100, 100, m_hTxQueue);

    // Let's go
    if (VCI_OK = res) then
        res := VCI_StartCan (m_hXatBoard, 0);
end;
```

## 5 VCI Funktionen in der Arbeitsphase der Applikation

### 5.1 VCI\_ReadQueObj

In einer VCI Anwendung, die auf Callbacks und Windows Messages verzichtet, muß die Empfangsqueue gepollt werden. VCI stellt dafür die Funktion VCI\_ReadQueObj bereit:

```
function VCI_ReadQueObj ( board_hdl: word;
                        que_hdl  : word;
                        count   : word;
                        p_obj   : PPCI_CAN_OBJ_ARRAY ): Integer;
```

Die Funktion kopiert eine wählbare Anzahl von CAN Empfangstelegrammen in einen vom Programmierer verwalteten Speicherbereich. Der Speicherbereich muß als Vektor (array) oder Einzelelement der Struktur VCI\_CAN\_OBJ allokiert sein. Größe und Anfangsadresse des Speicherbereichs werden der Funktion als count und p\_obj übergeben; board\_hdl und que\_hdl identifizieren die CAN Karte und die Queue. Die maximale unterstützte Vektordimension ist 13, d.h. es können mit einem Funktionsaufruf bis zu 13 CAN Objekte durch VCI übergeben werden.

Der Rückgabewert der Funktion ist die Anzahl der tatsächlich kopierten CAN Objekte (ggf. 0), im Fehlerfall wird ein VCI Fehlercode < 0 zurückgeliefert.

Hier ist ein Implementierungsbeispiel für die fortlaufende Abfrage der VCI Empfangsqueue:

```
var
    asCanObj : VCI_CAN_OBJ_ARRAY;           // Local allocation of CAN receive buffer
    pCanObj  : PPCI_CAN_OBJ;
    i        : integer;
    iVCIRes  : integer;                     // VCI result

begin
    while not(Application.Terminated) do
        begin
            pCanObj := @asCanObj;
            iVCIRes := VCI_ReadQueObj ( m_hXatBoard, m_hRxQueue,
                                       sizeof(asCanObj) div sizeof(VCI_CAN_OBJ), @asCanObj );

            if (0 < iVCIRes) then
                for i:=1 to iVCIRes do
                    begin
                        { Process received CAN message pCanObj }
                        inc(pCanObj);
                    end;
                Sleep(10);
            end;
        end;
    end;
```

### 5.2 Receive WM Handler

Ist ein Window Message Handler definiert, oder wird mit einer VCI Rx-Callbackfunktion gearbeitet, so ist die Allokierung eines lokalen VCI Empfangspuffers wie bei VCI\_ReadQueObj nicht erforderlich. Denn sowohl

die Anfangsadresse des VCI Empfangspuffers als auch die Anzahl der Empfangsobjekte werden gleich mitgeliefert. Im Receive WM Handler kann somit sofort auf die empfangenen CAN Objekte zugegriffen werden:

```

procedure WMVCIRX (var Msg: TVciRxMsg); message WM_VCIRX;
var
  pCanObj : PPCI_CAN_OBJ;
  i       : integer;

begin
  pCanObj := @Msg.pData.sData;
  for i:=1 to Msg.dwCnt do
  begin
    { Process received CAN message pCanObj }
    inc(pCanObj);
  end;
end;

```

Voraussetzung für die Zuweisung des pCanObj ist, daß der Compilerschalter {\$TYPEDADDRESS} ("Typisierter @-Operator") deaktiviert ist – dies entspricht passenderweise bereits der Projektstandardeinstellung. Viel wichtiger aber ist die Typdefinition der VCI Empfangs Windowsbotschaft TVciRxMsg, die im VCI2.pas Header erst ab VCI Version 2.16 vorhanden ist. Fügen Sie deshalb ggf. in den interface-Abschnitt Ihrer Delphi Unit diese Typdefinitionen ein:

```

type
  PPCI_RxMsgData = ^TVciRxMsgData;
  TVciRxMsgData = packed record
    bQue : byte; // Handle of VCI RxQueue
    sData : VCI_CAN_OBJ_ARRAY;
  end;
  TVciRxMsg = packed record // Equals Messages.TMessage
    Msg : Cardinal; // WM_VCIRX (or equivalent)
    dwCnt : dword; // WPARAM
    pData : PPCI_RxMsgData; // LPARAM
    Result: Longint
  end;

```

Damit wird zum einen die VCI Empfangs Windows Message delphikonform deklariert, zum anderen werden Datenstrukturen für den direkten Zugriff auf die in der Windowsbotschaft übergebenen VCI CAN Daten eingeführt.

### 5.3 VCI\_CAN\_OBJ Struktur

Abschließend noch einige Worte zum Format der VCI CAN Daten selbst. Jede vom VCI empfangene CAN Nachricht wird in einer Struktur namens VCI\_CAN\_OBJ bereitgestellt. Diese enthält neben dem CAN Telegramm (ID + 8 Datenbytes) noch zusätzlich Zeitstempel, RTR-Bit sowie diverse Statusinformationen:

```

VCI_CAN_OBJ = packed record
  time_stamp : Longint;
  id         : Longint;
  len4_rtr1_res3: Byte;
  a_data     : VCI_CAN_DATA;
  sts       : Byte;
end;

```

time\_stamp ist der absolute Wert des Empfangszeitpunktes des CAN Rahmens normiert auf das in der Funktion VCI\_ConfigQueue angegebene Zeitintervall ts\_res. In id steht der Identifier des CAN Telegramms, sowohl für 11-Bit wie für 29-Bit Rahmen, rechtsbündig. len4\_rtr1\_res3 definiert ein Bitfeld. Es setzt sich zusammen aus dem DLC des CAN Rahmens, und dem RTR-Bit. Die obersten drei Bits des Strukturelements sind reserviert, aber nicht zu 0 gesetzt:



**Aufbau des Strukturelements VCI\_CAN\_OBJ.len4\_rtr1\_res3**

a\_data enthält das Datenfeld des CAN Rahmens. Die Anzahl der gültigen Bytes, also die Länge des Datenfeldes, wird durch Maskierung des DLCs gewonnen: (VCI\_CAN\_OBJ.len4\_rtr1\_res3 and \$F). Im letzten Element VCI\_CAN\_OBJ.sts stehen VCI-interne Statusinformationen.

### 5.4 VCI\_TransmitObj

Verglichen mit der Entgegennahme empfangener CAN Daten ist das Versenden einer CAN Nachricht vergleichsweise einfach. Die hierfür zur Verfügung stehende Funktion `VCI_TransmitObj` hat folgende Syntax:

```
function VCI_TransmitObj ( board_hdl: word;  
                          que_hdl  : word;  
                          id       : Longint;  
                          len      : Byte;  
                          pData    : PPCI_CAN_DATA ): Integer;
```

`board_hdl` und `que_hdl` identifizieren das CAN Interface Board und die VCI Sendequueue. Der CAN Identifier `id`, das CAN Datenfeld `pData` und die benutzte Länge des Datenfeldes `len` werden als Einzelparameter übergeben.

Der Rückgabewert der Funktion ist ein VCI Fehlercode. Im Erfolgsfall wird `VCI_OK`, im Fehlerfall ein Wert  $< 0$  zurückgeliefert.

## 6 VCI Funktionen bei der Beendigung der Applikation

Unerlässlich ist der Aufruf von `VCI_CancelBoard` beim Beenden der Anwendung:

```
function VCI_CancelBoard (board_hdl: word): Integer;
```

Dieser Aufruf darf nicht erst beim Entladen der Anwendung (bzw. DLL) erfolgen, sondern muß noch während der regulären Programmlaufzeit gemacht werden, beispielsweise im `OnDestroy`-Handler des Hauptformulars:

```
procedure TFormMain.FormDestroy (Sender: TObject);  
begin  
  if (0 <= m_hxatBoard) then  
    VCI_CancelBoard (m_hxatBoard);  
  { Further cleanup }  
end;
```

Es empfiehlt sich, bereits zu Beginn der Programmdeinitialisierung den verwendeten CAN Controller mit `VCI_ResetCan` zu deaktivieren, damit keine CAN Empfangstelegramme den Programmabbau stören. Die Syntax der entsprechenden VCI Funktion lautet

```
function VCI_ResetCan ( board_hdl: word;  
                       can_num  : Byte ): Integer;
```

## 7 Further Literature

- [1] Application Note "Building Applications with the VCI CAN driver"
- [2] VCI Programmierhandbuch
- [3] CAN Buch

## 8 Contact Information

---

### IXXAT Automation GmbH

Leibnizstr. 15  
88250 Weingarten  
Germany  
Tel.: +49-(0)7 51 / 5 61 46-0  
Fax: +49-(0)7 51 / 5 61 46-29  
E-Mail: [info@ixxat.de](mailto:info@ixxat.de)  
Internet: [www.ixxat.de](http://www.ixxat.de)

### IXXAT Inc.

120 Bedford Center Road  
Bedford, NH 03110  
USA  
Phone: +1-603-471-0800  
Fax: +1-603-471-0880  
E-mail: [sales@ixxat.com](mailto:sales@ixxat.com)  
Internet: [www.ixxat.com](http://www.ixxat.com)

### Others

For a list of international distributors please consult the IXXAT web page under:  
  
[www.ixxat.de](http://www.ixxat.de) | contact | distributors