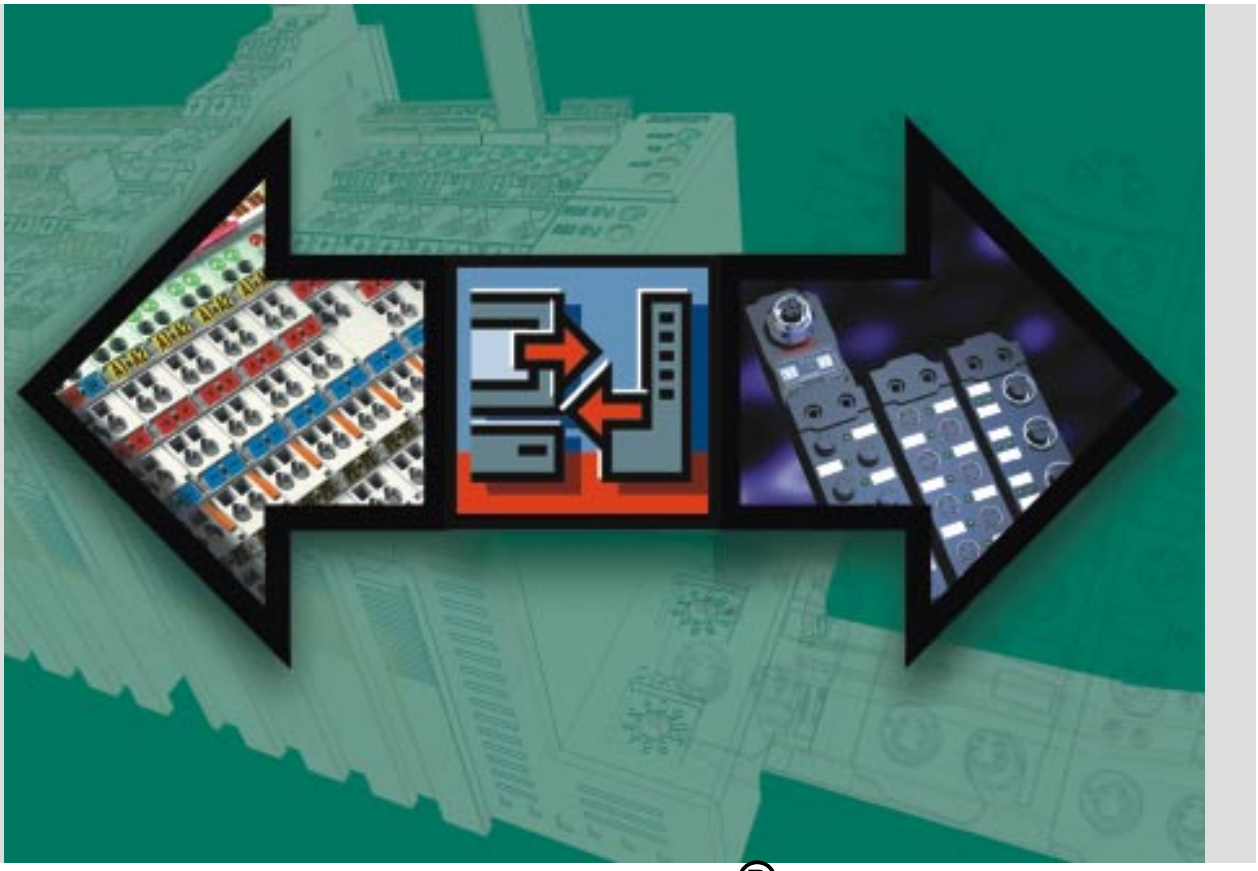


Beckhoff® TwinCAT

Total Windows Control and Automation Technology

KS8000: Kommunikations-Library für serielle Buskoppler BK8xxx

Letzte Änderung: 18.05.2000

BECKHOFF
INDUSTRIE ELEKTRONIK

Inhaltsverzeichnis

KS8000: Kommunikations-Library für serielle Buskoppler BK8xxx

Übersicht

Die "Beckhoff KS8000 Kommunikations-Library" stellt Funktionalitäten zur Verfügung, mit denen auf einfache Weise über eine serielle PC Schnittstelle mit Beckhoff BK8x00 Buskopplern kommuniziert werden kann.

Spezifikation

KS8000 ermöglicht den Zugriff auf das Ein- und Ausgangsprozeßabbild der BK8xxx Buskoppler über die serielle PC Schnittstelle. Pro serieller PC Schnittstelle kann eine Kommunikation zu einem BK8100 Koppler (RS232) oder bis zu 99 BK8000 Kopplern (RS485) aufgebaut werden. Bei einer Kommunikation wird das gesamte Ein- und Ausgangsabbild übertragen. Die Kommunikationsdauer ist somit abhängig von der Größe des Prozeßabbildes, also der Anzahl und Art der angehängten Klemmen.

(Messung: RS232 Koppler, 38400 Baud, Prozeßabbild 1 Wort ergibt sich ca. 6ms. Bei 15 Worten ca. 20ms).

Des weiteren kann die WatchDog-Zeit der Koppler verändert werden.

Lieferumfang

- KS8000 ActiveX Komponente:

  \Bin\BkcomOCX.ocx
  \Bin\BkcomOCX.tlb

  Die "KS8000 ActiveX Komponente" ist von sämtlichen Programmiersprachen einsetzbar, die mit den Spezifikationen des Component Object Modell (COM) von Microsoft arbeiten: VC++, Visual Basic, Borland C++ Builder, Delphi, Java ...

- KS8000-DLL: Standard C-DLL zum Einsatz unter VC++.

  \Bin\BkcomDLL.dll
  \Bin\BkcomDLL.h
  \Bin\BkcomDLL.lib

- LabView-VI-DLL:

  \Bin\BkcomLV.dll

  Labview Templates:
  \LabView Template\Close-BkComLV.vi
  \LabView Template\Open-BkComLV.vi
  \LabView Template\ReadWrite-BkComLV.vi
  \LabView Template\Read-BkComLV.vi

  Das grafische Programmiersystem LabVIEW von National Instruments unterstützt das Erstellen von Anwendungen ohne das aufwendige Schreiben von Programmtext. Programmiert wird durch Auswählen, Einfügen und Verbinden von grafischen Symbolen, zu sogenannten Blockdiagrammen, mit Hilfe der Maus. LabVIEW -Programme heißen Virtuelle Instrumente, abgekürzt VI.

Neben der Dokumentation stehen ausführliche Beispiele für C++, Visual Basic, Borland C++ Builder und Delphi zur Verfügung.

KS8000 ActiveX Control

Eigenschaften

BKxBaudrate

Legt die Baudrate der Kommunikation fest

Type
Long

Wertebereich

- Baud_9600 = 9600 Baud
- Baud_19200 = 19200 Baud
- Baud_38400 = 38400 Baud (Defaulteinstellung)

Bemerkungen
Die weiteren Kommunikationsparameter sind auf 8 Datenbits, 1 Stopbit und gerade Partität festgelegt und können nicht verändert werden.

Die BK8x00 Buskoppler passen sich automatisch der vorgegebenen Baudrate an. Der Buskoppler benötigt eventuell 4 Kommunikationsversuche um sich auf die korrekte Baudrate zu synchronisieren.

BKxTyp

Legt den Typ des Buskopplers fest.

Type
Long

Wertebereich

- (1) BKxType_RS485 = 1 (Koppler mit RS485 Kommunikation) (Defaulteinstellung)
- (2) BKxType_RS232 = 2 (Koppler mit RS232 Kommunikation)

Bemerkungen
Die Einstellung dieser Eigenschaft ist notwendig, um einen korrekten Datenaustausch zu gewährleisten. Die Bx8000 Koppler arbeiten über die RS485 Schnittstelle, alle anderen Koppler mit RS232.

WICHTIGER HINWEIS Bx8000 (RS485):
Wir empfehlen dringend RS-485 PC-Interface-Karte -oder externe Konverter- zu verwenden, die eine automatische Umschaltung der Sende- und Empfangsrichtung vornehmen. In diesem Fall ist der BkxTyp auf BKxType_RS232 = 2 (Koppler mit RS232 Kommunikation) zu setzen.

BKxCommPort

Legt die serielle PC Schnittstelle fest, über welche die Kommunikation mit den Kopplern erfolgt.

Type
Long

Wertebereich
Com-Port 1-8 (Default ist 2)

BKxCommPort

Öffnet / Schließt die serielle Schnittstelle.

Type
Boolean

Wertebereich

- True = Com-Port öffnen
- False = Com-Port schließen

Bemerkungen

Der Property-Zugriff ist nur zur Laufzeit möglich.

Das OCX erzeugt eine Exception, wenn eine Schnittstelle geöffnet werden soll, die physikalisch nicht vorhanden ist bzw. von einem anderen Prozeß bereits belegt ist.

BKxTimeout

Legt Dauer der Wartezeit auf das Response-Telegramm vom Koppler fest.

Type

Long

Wertebereich

Zeit in ms

Methoden

BK8xProcSyncReadReq

Bei der Methode BK8xProcSyncReadReq handelt es sich um einen synchronen Kommunikationsaufruf zum Auslesen des gesamten Eingangsprozeßabbildes eines BK8x00 Buskopplers. Die Größe des Prozeßabbildes hängt von der Anzahl und Art der gesteckten Klemmen an diesem Koppler ab.

```
BkcomComErr BK8xProcSyncReadReq (
          long lMultiPoint,
          long lStatus,
          long cwRecLength,
          long lpRecBuff
);
```

Parameters

lMultiPoint
: [in]
(1..99) Spezifiziert den Empfänger

lStatus
: [out]
(1..99) Kopplerstatus (siehe Anhang)

cwRecLength
: [out]
Anzahl der gelesenen Worte (ein Wort in einem Long-Wert) ab Offset 0 im Prozeßeingangsabbild

lpRecBuff
: [in, out]
Empfangspuffer Long (Array)

Return Values

BKcomComErr
: Returncode (siehe Anhang)

Beispiel

```
Dim lRet as long, lMultiPoint as long, lStatus as Long, cwRecLength as long
Dim RecBuff(255) as long

lmultiPoint = 11 ' ZielStation 11
LRet = Bkcom1.BK8xProcSyncReadReq( lMultiPoint, lStatus, cwRecLength, RecBuff(0))
```

```
' RecBuff(0) enthält Eingangswort[0] vom Koppler
' RecBuff(1) enthält Eingangswort[1] vom Koppler
' RecBuff(2) enthält Eingangswort[2] vom Koppler
```

BK8xProcSyncReadWriteReq

Bei der Methode BK8xProcSyncReadWriteReq handelt es sich um einen synchronen Kommunikationsaufruf zum Schreiben des gesamten Ausgangs- und Auslesen des gesamten Eingangsprozeßabbildes eines BK8x00 Buskopplers. Die Größe des gelesenen Prozeßabbildes hängt von der Anzahl und Art der gesteckten Klemmen an diesem Koppler ab. Es MUSS das gesamte Prozeßausgangsabbild beschrieben werden, es ist nicht möglich nur einen Teil bzw. Ausschnitt zu beschreiben. Die Sende- bzw. Empfangspuffer sind vom Typ LONG (32bit), es wird jedoch nur das Low-Word zum/vom Koppler übertragen (siehe Beispiel unten).

```
BkcomComErr BK8xProcSyncReadWriteReq (
          long lMultiPoint,
          long lStatus,
          long cwSendLength,
          long lpSendBuff,
          long cwRecLength,
          long lpRecBuff
);
```

Parameters

lMultiPoint
[in]
(1..99) Spezifiziert den Empfänger

lStatus
[out]
(1..99) Kopplerstatus (siehe Anhang)

cwSendLength
[in]
Anzahl der zu schreibenden Worte (ein Wort in einem Long-Wert) ab Offset 0 im Prozeßausgangsabbild

lpSendBuff
[in, out]
Empfangspuffer Long (Array)

cwRecLength
[out]
Anzahl der gelesenen Worte (ein Wort in einem Long-Wert) ab Offset 0 im Prozeßeingangsabbild

lpRecBuff
[in, out]
Empfangspuffer Long (Array)

Return Values

BKcomComErr
Returncode (siehe Anhang)

Beispiel

```
Dim lRet as long, lMultiPoint as long, lStatus as Long, cwRecLength as long, cwSendLength as long
Dim RecBuff(255) as long
Dim SendBuff(255) as long

lmultiPoint = 11 ' ZielStation 11

SendBuff(0) = 1       ' Ausgangswort[0] auf Koppler wird 1
SendBuff(1) = &H55    ' Ausgangswort[1] auf Koppler wird &H55
SendBuff(2) = &HAAEE  ' Ausgangswort[2] auf Koppler wird &HAAEE
```

```
LRet = Bkcom1.BK8xProcSyncReadWriteReq( IMultiPoint, IStatus, cwSendLength, SendBuff(0),cwRecLength, RecBuff(0))

' RecBuff(0) enthält Eingangswort[0] vom Koppler
' RecBuff(1) enthält Eingangswort[1] vom Koppler
' RecBuff(2) enthält Eingangswort[2] vom Koppler
```

BK8xWatchDogReadReq

Bei der Methode BK8xWatchDogReadReq handelt es sich um einen synchronen Kommunikationsaufruf zum Lesen der eingestellten WatchDog Zeit auf dem über <IMultiPoint> gewählten Ziel-Koppler.

```
BkcomComErr BK8xWatchDogReadReq(
        long lMultiPoint,
        long WatchDog
);
```

Parameters

IMultiPoint
[in]
(1..99) Spezifiziert den Empfänger

WatchDog
[out]
WatchDog in ms

Return Values

BKcomComErr
Returncode (siehe Anhang)

Beispiel

```
Dim lRet as long
Dim lValue as long

LRet = BkcomOCX1.BK8xWatchDogReadReq(11,lValue)

' die Variable <lValue> enthält die Watchdogzeit der Station 11
```

BK8xWatchDogWriteReq

Bei der Methode BK8xWatchDogWriteReq handelt es sich um einen synchronen Kommunikationsaufruf zum Setzen einer neuen WatchDog Zeit auf dem über <MultiPoint> gewählten Ziel-Koppler.

Diese Methode führt zu einem Neustart des Koppler (Reboot), erst dann hat der Koppler die neue WatchDog Zeit übernommen.

Der Koppler sollte sich zuvor auf die korrekte Baudrate synchronisiert haben, d.h. eventuell müssen zunächst 4 Kommunikationsversuche zur Synchronisation stattfinden.

```
BkcomComErr BK8xWatchDogWriteReq(
        long lMultiPoint,
        long WatchDog
);
```

Parameters

IMultiPoint
[in]
(1..99) Spezifiziert den Empfänger

WatchDog
[in]
WatchDog in ms

Parameters

Return Values

BKcomComErr
Returncode (siehe Anhang)

Beispiel

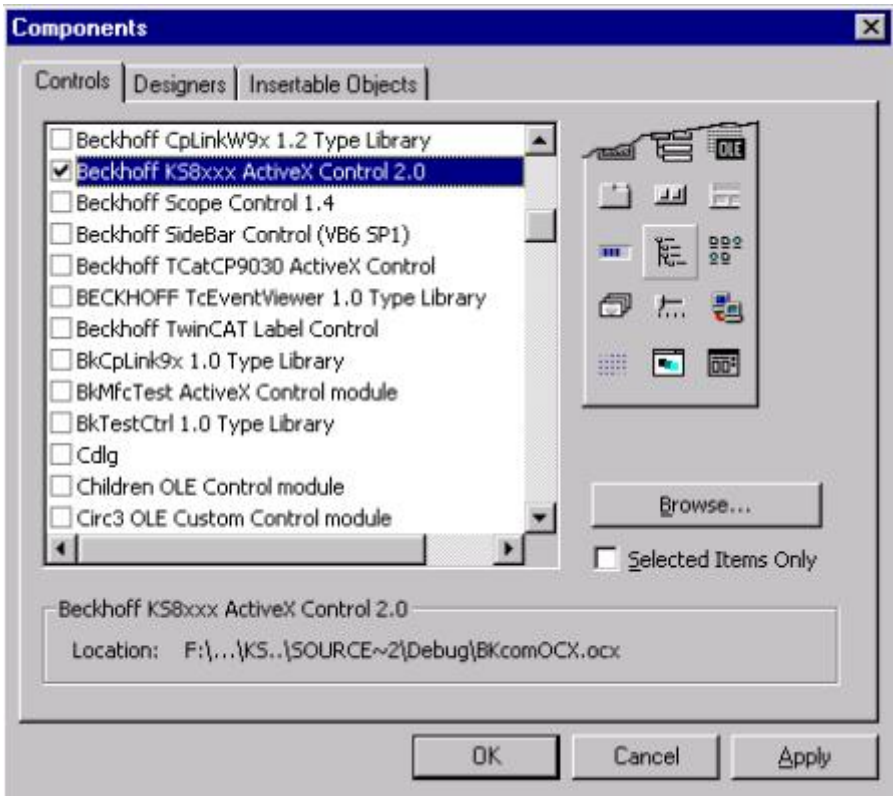
```
Dim lRet as long


LRet = BkcomOCX1.BK8xWatchDogWriteReq(11, 2000)

' Die Watchdogzeit der Station 11 wird auf 2000ms eingestellt. Der Koppler bootet.
```

Einbindung OCX in Applikationen

Einbinden des OCX in Visual Basic

In Visual Basic kann das KS8000 ActiveX Control eingesetzt werden.

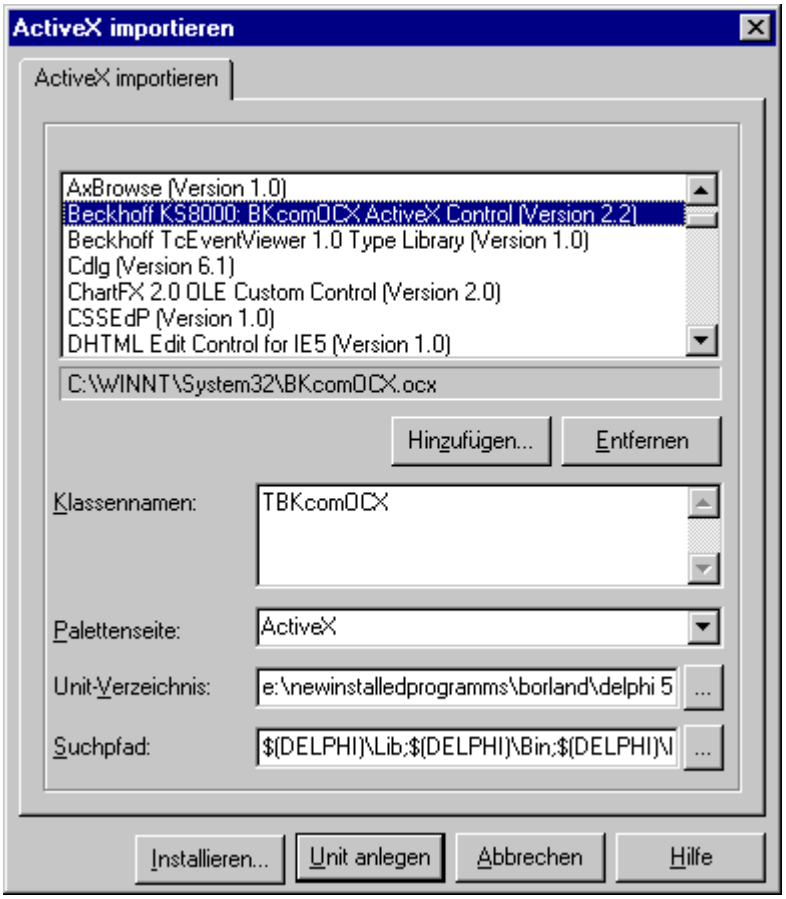
Dazu müssen Sie in Visual Basic unter dem Menüpunkt ‚Projekt' den Befehl ‚Komponenten...' auswählen und den Eintrag ‚Beckhoff KS8000 ActiveX Control' markieren.



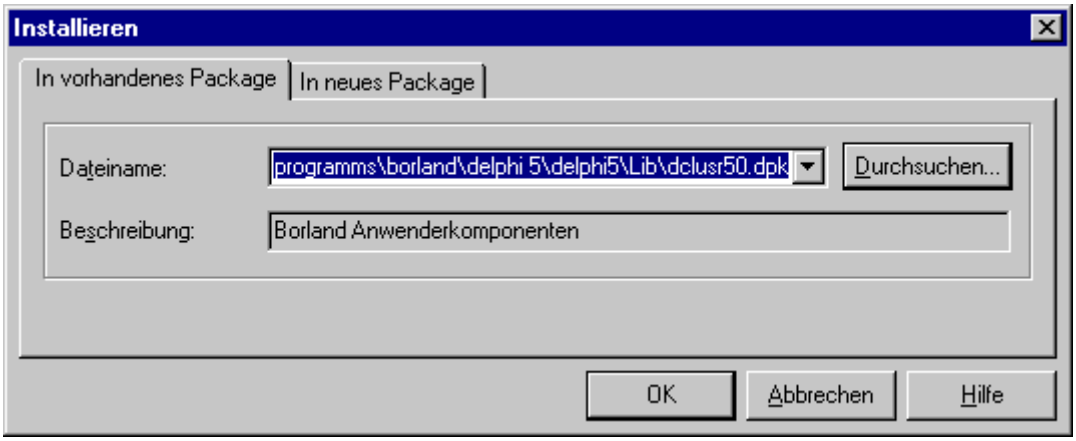
Anschließend erscheint das KS8000 ActiveX Control in der Toolbox von Visual Basic (rechts unten).

Einbinden des OCX in Delphi und C++Builder

Um das KS8000-OCX in Delphi oder C++Builder Applikationen einbinden zu können, müssen Sie das KS8000-OCX in die Komponenten-Palette von Delphi bzw. C++Builder einbinden. Dazu müssen Sie unter dem Menüpunkt *Komponente* den Befehl *ActiveX importieren…* auswählen und in dem Dialogfenster den Eintrag *Beckhoff KS8000: BKcomOCX ActiveX Control* markieren.

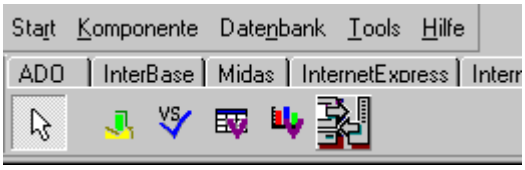


Anschließend mit *Installieren* und *OK* bestätigen. Das Package muß danach neu kompiliert werden.



Nach einer erfolgreichen Kompilierung wird das ActiveX-Komponent registriert. Beim Schließen des Packages müssen die Änderungen gespeichert werden.

Die neue Komponente erscheint dann in der ActiveX-Komponenten-Palette von Delphi bzw. C++Builder.



Um das ActiveX-Komponent benutzen zu können, müssen Sie es auf die Form ziehen.

Beispiele

Übersicht Beispiele

Beschreibung	Sprache	Quelltexte
Beispiel 1: Demo der Methoden	Visual Basic 5.0	OcxSample01.exe
Beispiel 2: Synchrones Lesen des Prozeß-Eingangsabbildes	Delphi 5.0	OcxSample02.exe
Beispiel 3: Synchrones Schreiben/Lesen der Prozeßabbilder	Delphi 5.0	OcxSample03.exe
Beispiel 4: Lesen/Setzen der Koppler-Watchdogzeit	Delphi 5.0	OcxSample04.exe
Beispiel 5: Synchrones Lesen des Prozeß-Eingangsabbildes	C++Builder 5.0	OcxSample05.exe
Beispiel 6: Synchrones Schreiben/Lesen der Prozeßabbilder	C++Builder 5.0	OcxSample06.exe
Beispiel 7: Lesen/Setzen der Koppler-Watchdogzeit	C++Builder 5.0	OcxSample07.exe
Beispiel 8: Demo aller Methoden	C++Builder 5.0	OcxSample08.exe

VB: Demo der Methoden

Das VB-Demo zeigt die Einbindung des KS8000-OCX in ein Visual Basic Projekt und die Benutzung der Properties und Methoden.

Beispielprogramm 'VB: Demo der Methoden' entpacken

Delphi: Synchrones Lesen der Prozeßeingangsdaten

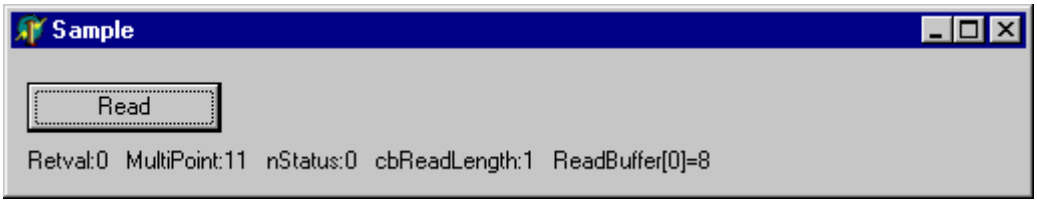
Aufgabe

Aus der Windows-Anwendung soll das Prozeßeingangsabbild des Kopplers ausgelesen werden. Der Koppler wurde mit einer digitalen Eingangs- und einer digitalen Ausgangsklemme (z.B. KL1104 und KL2114) bestückt. Jede Klemme besitzt jeweils 4 Ein/Ausgänge. Auf dem Koppler wurde mit den Drehschaltern die Stationsadresse 11 eingestellt.

Beschreibung

In der Ereignisfunktion *FormCreate()* werden fogende Eigenschaften (Parameter) für die Kommunikation mit dem Koppler gesetzt: BKxBaudrate, BKxTyp, BKxTimeout, BKxCommPort. Durch setzen der Eigenschaft BKxPortOpen auf *true* wird der Kommunikationsport geöfftnet. In der Erreignisfunktion *OnDestroy()* wird der Kommunikationsport durch setzen der Eigenschaft *BKxPortOpen* auf *false* geschlossen.

Beim Betätigen des Buttons auf der Form, wird die Ereignisfunktion *OnReadButtonClick()* aufgerufen. In dieser Ereignisfunktion wird synchron über die Methode BK8xProcSyncReadReq() das gesamte (reale) Prozeßeingangsabbild des Kopplers ausgelesen und als Text auf der Form ausgegeben. In unserem Beispiel beträgt die tatsächliche Länge der Prozeßeingangsdaten des Kopplers 4 Bit. Der Parameter *cbReadLength* bestimmt die tatsächliche Länge der Prozeßeingangsdaten des Kopplers als Anzahl der vielfachen einer Word-Grösse.



Delphi Programm

```
procedure TForm1.FormCreate(Sender: TObject);
begin
   BKcomOCX1.BKxBaudrate := Baud_38400;
   BKcomOCX1.BKxTyp := BKxType_RS485;
   BKcomOCX1.BKxTimeout := 1000;
   BKcomOCX1.BKxCommPort := 2;
   BKcomOCX1.BKxPortOpen := true;
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
   BKcomOCX1.BKxPortOpen := false;
end;

procedure TForm1.OnReadButtonClick(Sender: TObject);
var
   nMultiPoint     :integer;
   nStatus         :integer;
   nRet            :integer;
   ReadBuffer      :Array[0..254] of integer;
   cbReadLength    :integer;
begin
   nMultiPoint     := 11;
   cbReadLength    := 1;

   nRet := BKcomOCX1.BK8xProcSyncReadReq( nMultiPoint, nStatus, cbReadLength, ReadBuffer[0] );
   DataLabel.Caption :='Retval:' + IntToStr( nRet ) +
                       ' MultiPoint:' + IntToStr( nMultiPoint ) +
                       ' nStatus:' + IntToStr( nStatus ) +
                       ' cbReadLength:' + IntToStr( cbReadLength ) +
                       ' ReadBuffer[0]=' + IntToStr( ReadBuffer[0] );
end;
```

Beispielprogramm 'Synchrones Lesen der Prozeßeingangsdaten' entpacken

Delphi: Synchrones Lesen/Schreiben der Prozeßabbilder

Aufgabe

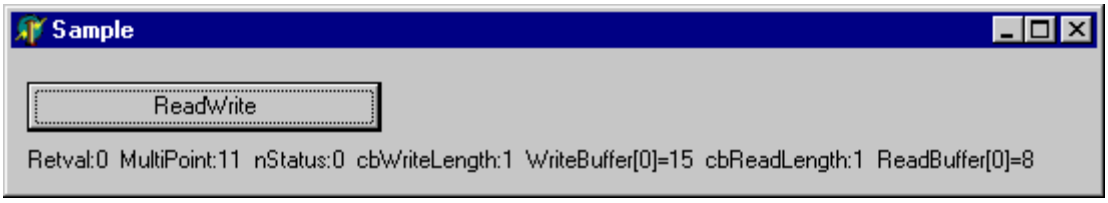
Aus der Windows-Anwendung soll das Prozeßeingangsabbild ausgelesen und gleichzeitig das Prozeßausgangsabbild des Kopplers gesetzt werden. Der Koppler wurde mit einer digitalen Eingangs- und einer digitalen Ausgangsklemme (z.B. KL1104 und KL2114) bestückt. Jede Klemme besitzt jeweils 4 Ein/Ausgänge. Auf dem Koppler wurde mit den Drehschaltern die Stationsadresse 11 eingestellt.

Beschreibung

In der Ereignisfunktion *FormCreate()* werden fogende Eigenschaften (Parameter) für die Kommunikation mit dem Koppler gesetzt: BKxBaudrate, BKxTyp, BKxTimeout, BKxCommPort. Durch setzen der Eigenschaft BKxPortOpen auf *true* wird der Kommunikationsport geöfftnet. In der Erreignisfunktion *OnDestroy()* wird der Kommunikationsport durch setzen der Eigenschaft *BKxPortOpen* auf *false* geschlossen.

Beim Betätigen des Buttons auf der Form, wird die Ereignisfunktion *OnReadWriteButtonClick()* aufgerufen. In dieser Ereignisfunktion wird synchron über die Methode BK8xProcSyncReadWriteReq() das gesamte Prozeßeingangsabbild des Kopplers in ein Array von Integer-Werten eingelesen und gleichzeitig das Prozeßausgangsabbild mit neuen Werten aus einem weiteren Array gesetzt. Die Werte der übergebenen Parameter werden als Text auf der Form ausgegeben. Die Parameter *cbWirteLength* und *cbReadLength* bestimmen die tatsächliche Länge der beiden Prozeßabbilder in vielfachen von Word-Grössen. In unserem Fall beträgt die reale

Länge der Prozeßabbilder jeweils 4 Bits, deshalb werden die beiden Parameter =1 gesetzt.



Delphi Programm

```
procedure TForm1.FormCreate(Sender: TObject);
begin
   BKcomOCX1.BKxBaudrate := Baud_38400;
   BKcomOCX1.BKxTyp := BKxType_RS485;
   BKcomOCX1.BKxTimeout := 1000;
   BKcomOCX1.BKxCommPort := 2;
   BKcomOCX1.BKxPortOpen := true;
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
   BKcomOCX1.BKxPortOpen := false;
end;

procedure TForm1.OnReadWriteButtonClick(Sender: TObject);
var
   nMultiPoint :integer;
   nStatus :integer;
   nRet :integer;
   ReadBuffer :Array[0..254] of integer;
   cbReadLength :integer;
   WriteBuffer :Array[0..254] of integer;
   cbWriteLength :integer;
begin
   nMultiPoint := 11;
   cbReadLength := 1;
   cbWriteLength := 1;
   WriteBuffer[0]:= $F;

   nRet := BKcomOCX1.BK8xProcSyncReadWriteReq( nMultiPoint, nStatus, cbWriteLength, WriteBuffer[0],
cbReadLength, ReadBuffer[0] );
   DataLabel.Caption :='Retval:' + IntToStr( nRet ) +
                       ' MultiPoint:' + IntToStr( nMultiPoint ) +
                       ' nStatus:' + IntToStr( nStatus ) +
                       ' cbWriteLength:' + IntToStr( cbWriteLength ) +
                       ' WriteBuffer[0]=' + IntToStr( WriteBuffer[0] ) +
                       ' cbReadLength:' + IntToStr( cbReadLength ) +
                       ' ReadBuffer[0]=' + IntToStr( ReadBuffer[0] );
end;
```

Beispielprogramm 'Synchrones Lesen/Schreiben der Prozeßabbilder' entpacken.

Delphi: Lesen/Setzen der Koppler-Watchdogzeit

Aufgabe

Aus der Windows-Anwendung soll die aktuelle Wachdogzeit des Kopplers gelesen und eine neue, anwenderspezifische Watchdogzeit (500 Millisekunden) gesetzt werden können. Auf dem Koppler wurde mit den Drehschaltern die Stationsadresse 11 eingestellt.

Beschreibung

In der Ereignisfunktion *FormCreate()* werden fogende Eigenschaften (Parameter) für die Kommunikation mit dem Koppler gesetzt: BKxBaudrate, BKxTyp, BKxTimeout, BKxCommPort. Durch setzen der Eigenschaft BKxPortOpen auf *true* wird der Kommunikationsport geöfftnet. In der Erreignisfunktion *OnDestroy()* wird der Kommunikationsport durch setzen der Eigenschaft *BKxPortOpen* auf *false* geschlossen.

Beim Betätigen des Buttons *Watchdog Read* auf der Form, wird die Ereignisfunktion *OnWatchdogReadButtonClick()* aufgerufen. In dieser Ereignisfunktion wird über die Methode BK8xWatchDogReadReq() die aktuelle Watchdogzeit des Kopplers gelesen und als Text auf der Form ausgegeben. Über den Button *Watchdog Write* kann eine neue, anwenderspezifische Watchdogzeit gesetzt werden. Dabei wird die Ereignisfunktion *OnWatchdogWriteButtonClick()* und die Methode BK8xWatchDogWriteReq() aufgerufen. Von dem Koppler wird anschließend ein Reset durchgeführt.

Delphi Programm

```
procedure TForm1.FormCreate(Sender: TObject);
begin
   BKcomOCX1.BKxBaudrate := Baud_38400;
   BKcomOCX1.BKxTyp := BKxType_RS485;
   BKcomOCX1.BKxTimeout := 1000;
   BKcomOCX1.BKxCommPort := 2;
   BKcomOCX1.BKxPortOpen := true;
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
   BKcomOCX1.BKxPortOpen := false;
end;

procedure TForm1.OnWatchdogReadButtonClick(Sender: TObject);
var
   nMultiPoint  :integer;
   nRet         :integer;
   nWatchdog    :integer;
begin
   nMultiPoint := 11;
   nRet := BKcomOCX1.BK8xWatchDogReadReq( nMultiPoint, nWatchdog );
   DataLabel.Caption :='Retval:' + IntToStr( nRet ) +
                       ' MultiPoint:' + IntToStr( nMultiPoint ) +
                       ' Watchdog:' + IntToStr( nWatchdog ) + 'ms';
end;

procedure TForm1.OnWatchdogWriteButtonClick(Sender: TObject);
var
```

```
   nMultiPoint  :integer;
   nRet         :integer;
   nWatchdog    :integer;
begin
   nMultiPoint := 11;
   nWatchdog :=500;
   nRet := BKcomOCX1.BK8xWatchDogWriteReq( nMultiPoint, nWatchdog );
   DataLabel.Caption :='Retval:' + IntToStr( nRet ) +
                       ' MultiPoint:' + IntToStr( nMultiPoint ) +
                       ' Watchdog:' + IntToStr( nWatchdog ) + 'ms';
end;
```

Beispielprogramm 'Lesen/Setzen der Koppler-Watchdogzeit' entpacken.

C++Builder: Synchrones Lesen der Prozeßeingangsdaten

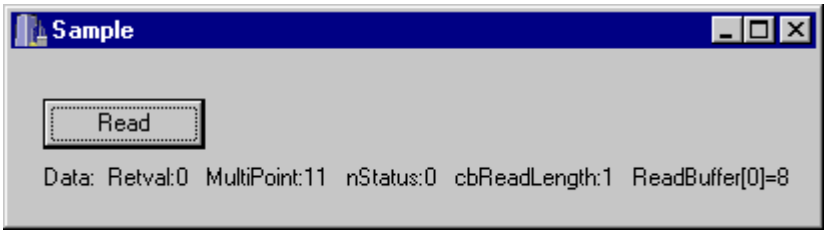
Aufgabe

Aus der Windows-Anwendung soll das Prozeßeingangsabbild des Kopplers ausgelesen werden. Der Koppler wurde mit einer digitalen Eingangs- und einer digitalen Ausgangsklemme (z.B. KL1104 und KL2114) bestückt. Jede Klemme besitzt jeweils 4 Ein/Ausgänge. Auf dem Koppler wurde mit den Drehschaltern die Stationsadresse 11 eingestellt.

Beschreibung

In der Ereignisfunktion *FormCreate()* werden fogende Eigenschaften (Parameter) für die Kommunikation mit dem Koppler gesetzt: BKxBaudrate, BKxTyp, BKxTimeout, BKxCommPort. Durch setzen der Eigenschaft BKxPortOpen auf *true* wird der Kommunikationsport geöfftnet. In der Erreignisfunktion *OnDestroy()* wird der Kommunikationsport durch setzen der Eigenschaft *BKxPortOpen* auf *false* geschlossen.

Beim Betätigen des Buttons auf der Form, wird die Ereignisfunktion *OnReadButtonClick()* aufgerufen. In dieser Ereignisfunktion wird synchron über die Methode BK8xProcSyncReadReq() das gesamte (reale) Prozeßeingangsabbild des Kopplers ausgelesen und als Text auf der Form ausgegeben. In unserem Beispiel beträgt die tatsächliche Länge der Prozeßeingangsdaten des Kopplers 4 Bit. Der Parameter *cbReadLength* bestimmt die tatsächliche Länge der Prozeßeingangsdaten des Kopplers als Anzahl der vielfachen einer Word-Grösse.



C++Builder Programm

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
   BKcomOCX1->BKxBaudrate = Baud_38400;
   BKcomOCX1->BKxTyp = BKxType_RS485;
   BKcomOCX1->BKxTimeout = 1000;
   BKcomOCX1->BKxCommPort = 2;
   BKcomOCX1->BKxPortOpen = true;
}

void __fastcall TForm1::FormDestroy(TObject *Sender)
```

```
{
   BKcomOCX1->BKxPortOpen = false;
}

void __fastcall TForm1::OnReadButtonClick(TObject *Sender)
{
   long nMultiPoint = 11;
   long nStatus;
   long nRet;
   long ReadBuffer[255];
   //ReadBuffer[0] == Coupler Output Word[0]
   //ReadBuffer[1] == Coupler Output Word[1]
   //ReadBuffer[2] == Coupler Output Word[2]
   long cbReadLength = 1;

   nRet = BKcomOCX1->BK8xProcSyncReadReq( nMultiPoint, &nStatus, &cbReadLength, &ReadBuffer[0] );
   DataLabel->Caption ="Retval:" + IntToStr( nRet ) +
                       " MultiPoint:" + IntToStr( nMultiPoint ) +
                       " nStatus:" + IntToStr( nStatus ) +
                       " cbReadLength:" + IntToStr( cbReadLength ) +
                       " ReadBuffer[0]=" + IntToStr( ReadBuffer[0] );
}
```

Beispielprogramm 'Synchrones Lesen der Prozeßeingangsdaten' entpacken

C++Builder: Synchrones Lesen/Schreiben der Prozeßabbilder

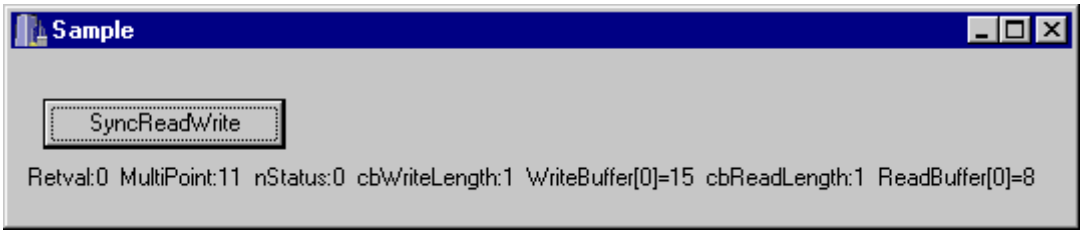
Aufgabe

Aus der Windows-Anwendung soll das Prozeßeingangsabbild ausgelesen und gleichzeitig das Prozeßausgangsabbild des Kopplers gesetzt werden. Der Koppler wurde mit einer digitalen Eingangs- und einer digitalen Ausgangsklemme (z.B. KL1104 und KL2114) bestückt. Jede Klemme besitzt jeweils 4 Ein/Ausgänge. Auf dem Koppler wurde mit den Drehschaltern die Stationsadresse 11 eingestellt.

Beschreibung

In der Ereignisfunktion *FormCreate()* werden fogende Eigenschaften (Parameter) für die Kommunikation mit dem Koppler gesetzt: BKxBaudrate, BKxTyp, BKxTimeout, BKxCommPort. Durch setzen der Eigenschaft BKxPortOpen auf *true* wird der Kommunikationsport geöfftnet. In der Erreignisfunktion *OnDestroy()* wird der Kommunikationsport durch setzen der Eigenschaft *BKxPortOpen* auf *false* geschlossen.

Beim Betätigen des Buttons auf der Form, wird die Ereignisfunktion *OnReadWriteButtonClick()* aufgerufen. In dieser Ereignisfunktion wird synchron über die Methode BK8xProcSyncReadWriteReq() das gesamte Prozeßeingangsabbild des Kopplers in ein Array von Integer-Werten eingelesen und gleichzeitig das Prozeßausgangsabbild mit neuen Werten aus einem weiteren Array gesetzt. Die Werte der übergebenen Parameter werden als Text auf der Form ausgegeben. Die Parameter *cwWirteLength* und *cwReadLength* bestimmen die tatsächliche Länge der beiden Prozeßabbilder in vielfachen von Word-Grössen. In unserem Fall beträgt die reale Länge der Prozeßabbilder jeweils 4 Bits, deshalb werden die beiden Parameter =1 gesetzt.



C++Builder Programm

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
   BKcomOCX1->BKxBaudrate = Baud_38400;
   BKcomOCX1->BKxTyp = BKxType_RS485;
   BKcomOCX1->BKxTimeout = 1000;
   BKcomOCX1->BKxCommPort = 2;
   BKcomOCX1->BKxPortOpen = true;
}

void __fastcall TForm1::FormDestroy(TObject *Sender)
{
   BKcomOCX1->BKxPortOpen = false;
}

void __fastcall TForm1::OnReadWriteButtonClick(TObject *Sender)
{
   long nMultiPoint = 11;
   long nStatus;
   long nRet;
   long ReadBuffer[255];
   long cwReadLength = 1;
   //ReadBuffer[0] == Coupler Output Word[0]
   //ReadBuffer[1] == Coupler Output Word[1]
   //ReadBuffer[2] == Coupler Output Word[2]
   long WriteBuffer[255];
   //WriteBuffer[0] == Coupler Input Word[0]
   //WriteBuffer[1] == Coupler Input Word[1]
   //WriteBuffer[2] == Coupler Input Word[2]
   long cwWriteLength = 1;

   //Set Data to Send:
   WriteBuffer[0]=0xF;

   nRet = BKcomOCX1->BK8xProcSyncReadWriteReq( nMultiPoint, &nStatus, cwWriteLength,
&WriteBuffer[0], &cwReadLength, &ReadBuffer[0] );
   DataLabel->Caption ="Retval:" + IntToStr( nRet ) +
                       " MultiPoint:" + IntToStr( nMultiPoint ) +
                       " nStatus:" + IntToStr( nStatus ) +
                       " cbWriteLength:" + IntToStr( cwWriteLength ) +
                       " WriteBuffer[0]=" + IntToStr( WriteBuffer[0] ) +
                       " cbReadLength:" + IntToStr( cwReadLength ) +
                       " ReadBuffer[0]=" + IntToStr( ReadBuffer[0] );
}
```

Beispielprogramm 'Synchrones Lesen/Schreiben der Prozeßabbilder' entpacken.

C++Builder: Lesen/Setzen der Koppler-Watchdogzeit

Aufgabe

Aus der Windows-Anwendung soll die aktuelle Wachdogzeit des Kopplers gelesen und eine neue, anwenderspezifische Watchdogzeit (1100 Millisekunden) gesetzt werden können. Auf dem Koppler wurde mit den Drehschaltern die Stationsadresse 11 eingestellt.

Beschreibung

In der Ereignisfunktion *FormCreate()* werden fogende Eigenschaften (Parameter) für die Kommunikation mit dem Koppler gesetzt: BKxBaudrate, BKxTyp, BKxTimeout, BKxCommPort. Durch setzen der Eigenschaft BKxPortOpen auf *true* wird der Kommunikationsport geöfftnet. In der Erreignisfunktion *OnDestroy()* wird der Kommunikationsport durch setzen der Eigenschaft *BKxPortOpen* auf *false* geschlossen.

Beim Betätigen des Buttons *Watchdog Read* auf der Form, wird die Ereignisfunktion *OnWatchdogReadButtonClick()* aufgerufen. In dieser Ereignisfunktion wird über die Methode BK8xWatchDogReadReq() die aktuelle Watchdogzeit des Kopplers gelesen und als Text auf der Form ausgegeben. Über den Button *Watchdog Write* kann eine neue, anwenderspezifische Watchdogzeit gesetzt werden. Dabei wird die Ereignisfunktion *OnWatchdogWriteButtonClick()* und die Methode BK8xWatchDogWriteReq() aufgerufen. Von dem Koppler wird anschließend ein Reset durchgeführt.

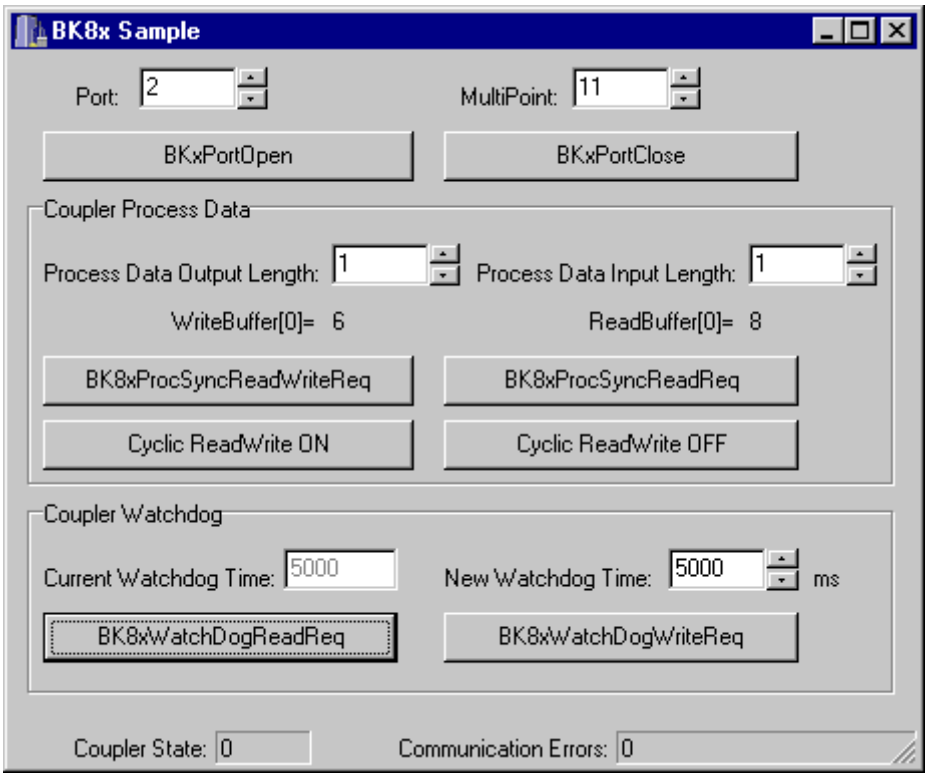
C++Builder Programm

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
   BKcomOCX1->BKxBaudrate = Baud_38400;
   BKcomOCX1->BKxTyp = BKxType_RS485;
   BKcomOCX1->BKxTimeout = 1000;
   BKcomOCX1->BKxCommPort = 2;
   BKcomOCX1->BKxPortOpen = true;
}

void __fastcall TForm1::FormDestroy(TObject *Sender)
{
   BKcomOCX1->BKxPortOpen = false;
}

void __fastcall TForm1::OnWatchdogReadButtonClick(TObject *Sender)
{
   long nMultiPoint = 11;
   long nRet;
   long nWatchdog;
   nRet = BKcomOCX1->BK8xWatchDogReadReq( nMultiPoint, &nWatchdog );
   DataLabel->Caption ="Retval:" + IntToStr( nRet ) +
                       " MultiPoint:" + IntToStr( nMultiPoint ) +
                       " Watchdog:" + IntToStr( nWatchdog ) + "ms";
}

void __fastcall TForm1::OnWatchdogWriteButtonClick(TObject *Sender)
{
   long nMultiPoint = 11;
   long nRet;
   long nWatchdog = 1100;
   nRet = BKcomOCX1->BK8xWatchDogWriteReq( nMultiPoint, nWatchdog );
   DataLabel->Caption ="Retval:" + IntToStr( nRet ) +
                       " MultiPoint:" + IntToStr( nMultiPoint ) +
                       " Watchdog:" + IntToStr( nWatchdog ) + "ms";
}
```

Beispielprogramm 'Lesen/Setzen der Koppler-Watchdogzeit' entpacken.

C++Builder: Demo aller Methoden

Bei diesem Beispiel handelt sich um eine umfangreichere Applikation, mit der alle verfügbaren Methoden verwendet und getestet werden können.



Beschreibung

Konfigurieren Sie die Portnummer und die MultiPoint-Adresse (Stationsadresse) befor Sie die verfügbaren Funktionen testen. Um auf das Prozeßabbild des Kopplers zugreifen zu können, muß der Port zuerst geöffnet werden. Über den Button *Cyclic ReadWrite ON* kann zyklisch auf die Prozeßabbilder der Koppler zugegriffen werden. Die Methode BK8xProcSyncReadWriteReq() wird dabei zyklisch von einem Multimedia-Timer aufgerufen.

Beispielprogramm 'Demo aller Methoden' entpacken.

KS8000 DLL

Referenz

Datentypen

BkComData

Beschreibt komplett einen zu öffnenden oder geöffneten Kommunikationsport, durch Dateihandle, Baudrate, Portnummer, Timeout und BK8x00-Typ.

```
struct BkComData{
        HANDLE hCommPort;
        Long nBaudrate;
        Long nPortNum;
        Long nTimeout;
        Long nBKxTyp;
};
```

Beispiel 1:

```
// Aggregatinitialisierung
BkComData tBkComObject = { NULL, Baud_38400, 1l, 2000l, BKxType_RS232 };
```

Beispiel 2:

```
// Einzelinitialisierung
BkComData tBkComObject;
tbkComObject.hCommPort = NULL;
tBkComObject.nBaudrate = Baud_38400;
tBkComObject.nPortNum = 1L; // COM 1
tBkComObject.nTimeout= 2000l;         // 2000ms
tBkComObject.nBKxTyp = BKxType_RS232; // BK8100
```

OpenBkComPort

Öffnet die serielle Schnittstelle, gemäß den Angaben des Strukturobjektes.

```
void OpenBkComPort(
        BkComData *pAnBkCom
);
```

Parameters

pAnBKCom
[in]
Spezifiziert COM-Port und Kommunikationsparameter

Beispiel

```
#include "BkComDll.h"

BkComData tBkComObject = { NULL, Baud_38400, 1l, 2000l, BKxType_RS232 };

if (OpenBkComPort(&tBkComObject) == ComErrNo)
  {
    // Port ist geöffnet

  }
```

Anmerkung

In der Funktion werden die Einstellungen festgelegt. Neben der angegebenen Baudrate sind die weiteren Kommunikationsparameter auf 8 Datenbits, 1 Stopbit und gerade Partität festgelegt und können nicht verändert werden.

Die BK8x00 Buskoppler passen sich automatisch der vorgegebenen Baudrate an. Dazu sind eventuell 4 Datenkommunikationen nötig.

In der Funktion wird der Dateihandle pAnBkCom->hCommPort gesetzt.

CloseBkComPort

Schließt die serielle Schnittstelle, gemäß den Angaben des Strukturobjektes.

```
void CloseBkComPort(
        BkComData *pAnBkCom
);
```

Parameters

pAnBKCom
[in]
Spezifiziert COM-Port und Kommunikationsparameter

Beispiel

```
#include "BkComDll.h"

BkComData tBkComObject;

// Port ist offen
CloseBkComPort(&tBkComObject);
```

Anmerkung

In der Funktion wird der Dateihandle pAnBkCom->hCommPort geschlossen und Null gesetzt.

BK8xProcSyncReadReq

Es handelt sich um einen synchronen Kommunikationsaufruf zum Auslesen des gesamten Eingangsprozeßabbildes eines BK8x00 Buskopplers.

```
long BK8xProcSyncReadReq (
        BkComData *pAnBKCom,
        long lMultiPoint,
        long FAR* lStatus,
        long FAR* cwRecLength,
        long FAR* lpRecBuff
);
```

Parameters

pAnBKCom
[in]
Spezifiziert COM-Port und Kommunikationsparameter

lMultiPoint
[in]
(1..99) Spezifiziert den Empfänger

lStatus
[out]
(1..99) Kopplerstatus (siehe Anhang)

cwRecLength
[out]
Anzahl der gelesenen Worte (ein Wort in einem Long-Wert) ab Offset 0 im Prozeßeingangsabbild

lpRecBuff
[in, out]
Empfangspuffer Long (Array)

Return Values

long
Returncode (siehe Anhang)

Beispiel

```
#include "BkComDll.h"

BkComData tBkComObject;
long nStatus, nRecLength;long lRecBuf[255];

// Port ist offen

{ // Prozeßdaten des BK8x00 mit Multipoint-Nummer 2..
if (BK8xProcSyncReadReq(    &tBkComObject,
                        2l,
                        &nStatus,
                        &nRecLength,
                        lRecBuf) == ComErrNo)

// stehen jetzt im Empfangspuffer , lRecBuf'}
```

Anmerkung

Vgl. BK8xProcSyncReadReq des KS8000-OCX.

BK8xProcSyncReadWriteReq

Es handelt es sich um einen synchronen Kommunikationsaufruf zum Schreiben des gesamten Ausgangs- und Auslesen des gesamten Eingangsprozeßabbildes eines BK8x00 Buskopplers. Die Größe des gelesenen Prozeßabbildes hängt von der Anzahl und Art der gesteckten Klemmen an diesem Koppler ab.

Es MUSS das gesamte Prozeßausgangsabbild beschrieben werden, es ist nicht möglich nur einen Teil bzw. Ausschnitt zu beschreiben. Die Sende- bzw. Empfangspuffer sind vom Typ Long (32bit), es wird jedoch nur das Low-Word zum/vom Koppler übertragen. (siehe auch Beispiel KS8000-OCX)

```
long BK8xProcSyncReadWriteReq (
         BkComData *pAnBKCom,
         long lMultiPoint,
         long FAR* lStatus,
         long FAR* cwSendLength,
         long FAR* lpSendBuff,
         long FAR* cwRecLength,
         long FAR* lpRecBuff
);
```

Parameters

pAnBKCom
: [in]
Spezifiziert COM-Port und Kommunikationsparameter

lMultiPoint
: [in]
(1..99) Spezifiziert den Empfänger

lStatus
: [out]
(1..99) Kopplerstatus (siehe Anhang)

cwSendLength
: [in]
Anzahl der zu schreibenden Worte (ein Wort in einem Long-Wert) ab Offset 0 im Prozeßausgangsabbild

lpSendBuff
: [in]
Sendepuffer Long (Array)

cwRecLength
: [out]
Anzahl der gelesenen Worte (ein Wort in einem Long-Wert) ab Offset 0 im Prozeßeingangsabbild

lpRecBuff
[in, out]
Empfangspuffer Long (Array)

Return Values

long
Returncode (siehe Anhang)

Beispiel

```
#include "BkComDll.h"

BkComData tBkComObject;
long nStatus, nSendLength, nRecLength;long lSendBuf[20], lRecBuf[20];

// Port ist offen

{ // Prozeßdaten des BK8x00 mit Multipoint-Nummer 2..
if (BK8xProcSyncReadWriteReq(   &tBkComObject,
                                2l,
                                &nStatus,
                                &nSendLength,
                                lSendBuf,                                   &nRecLength,
                                lRecBuf) == ComErrNo)

// stehen jetzt im Empfangspuffer , lRecBuf'}

// im Prozeßabbild stehen jetzt die Daten von ,lSendBuf'
```

Anmerkung
Vgl. BK8xProcSyncReadWriteReq des KS8000-OCX.

Einbindung DLL in C++

Notwendige Dateien

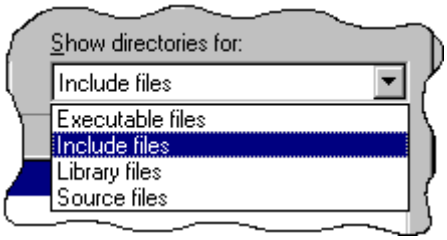
Zum Entwickeln der Programme benötigen Sie folgende Dateien:

- BkComDLL.dll - dynamische Funktion-Bibliothek
- BkComDLL.lib - Link-Library für die BkComDLL.dll
- BkComDLL.h - Deklarationen der BkCom-Funktionen

Die Dateien befindet sich im 'KS8000-Installations\BIN'-Verzeichnis.

Verzeichnis von *.h und *.lib angeben

Im Developer Studio muß unter dem Menüpunkt ‚Tools' der Befehl ‚Options' ausgewählt werden. Wählen Sie den Reiter ‚Directories' aus. Mit ‚Show directories for:' können Sie angeben, welchen Pfad Sie angeben wollen. Wählen Sie den Eintrag ‚Include files' aus.



Pfad für Headerdateien angeben

Am Ende der Liste ‚Directories:' ist ein leeres Rechteck vorhanden. Klicken Sie dieses mit der Maus an, um dort den Pfad für die Headerdateien einzugeben.

Pfad für Libraries angeben

Wählen Sie jetzt unter ‚Show directories for:' den Eintrag ‚Library files' aus. Geben Sie am Ende der Liste den Pfad der BkComDLL.lib an.

BkComDLL.lib zum Projekt hinzufügen

Nach dem Sie ein Projekt erstellt haben, müssen Sie die BkComDLL.lib dem Linken bekannt geben. Markieren Sie dazu in der Dateidarstellung das Projekt.

Betätigen Sie im Menü ‚Project' den Befehl ‚Settings'. Wählen Sie den Reiter ‚Link' aus.

In ‚Object/library modules:' tragen Sie, BkComDLL.lib' hinzu.

Beispiel

```
// BkComExample.cpp : Defines the entry point for the console application.
//
#include "BkComDll.h"

int main()
{
	long nStatus, nSendLength, nRecLength;
	long lSendBuf[20], lRecBuf[20], lRet; int idy, idx;
	char msg[30];

	// Handle , Baudrate 38400, COM 1, 2000ms Timeout, BK8100,
	BkComData aBkComObj = { NULL, Baud_38400, 2l, 2000l, BKxType_RS232 };

	if (!OpenBkComPort(&aBkComObj))
	{
		nSendLength = 1;		// ein Wort des Prozeßabbildes schreiben
		nRecLength = 1;			// erstes Wort des Prozeßabbildes lesen
		// Lauflicht erzeugen
		for (idy = 0; idy <= 1000; idy++)
		{
			for(idx = 0; idx <= 20; idx++)
			{
				lSendBuf[idx] = lSendBuf[idx] + 1;
				if (lSendBuf[idx] > 10000)
					lSendBuf[idx] = 0;
			}
			// Multipoint = 2
			lRet = BK8xProcSyncReadWriteReq(	&aBkComObj,
								2l,
								&nStatus,
								nSendLength,
								lSendBuf,
								&nRecLength,
								lRecBuf);
			if (lRet != 0)
			{
				sprintf(msg, "\nFehler: %ld\n", lRet);
				MessageBox(NULL, msg, "Read Write COM-Port", MB_OK
```

```
|MB_ICONEXCLAMATION);
                                }
                        }
                }
        CloseBkComPort(&aBkComObj);
        return 0;
}
```

Beispielcode 'VC++: KS8000-DLL Synchrones Schreib/Lesen der Prozeßdaten' entpacken

KS8000 LabView

Übersicht

LabVIEW

Das grafische Programmiersystem LabVIEW von National Instruments unterstützt das Erstellen von Anwendungen ohne das aufwendige Schreiben von Programmtext. Programmiert wird durch Auswählen, Einfügen und Verbinden von grafischen Symbolen, zu sogenannten Blockdiagrammen, mit Hilfe der Maus. LabVIEW - Programme heißen Virtuelle Instrumente, abgekürzt VI.

Weitergehende Informationen zur Benutzung und Programmierung finden Sie in den LabVIEW-Handbüchern.

SubVI - LabVIEW Unterprozeduren

Das Subroutinen-Konzept von LabVIEW ermöglicht das Einbinden eines VI in andere VI, als sogenanntes SubVI. Wichtig hierbei ist die beliebige Hierarchisierung, also die quantitativ unbegrenzte Verschachtelungstiefe von VI's. So kann ein SubVI seinerseits beliebig viele andere SubVI's enthalten.

SubVI ähneln den Unterprozeduren der klassischen Programmiersprachen, und können sowohl Werte empfangen als auch zurückgeben (Parametrierung).

Beim Einsatz in einem VI ist zu beachten, daß gleichnamige SubVI's stets nur in einer Instanz vorliegen, unabhängig von Anzahl und Ort der VI's in der sie als SubVI eingebunden sind. Demgemäß sind solche eingebundenen SubVI's lediglich Verweise auf das eine Ursprungs-VI, und verfügen daher auch nur über einen einzigen Datenbereich.

Falls sich ein SubVI bestimmte Daten oder Zustände merken muß, so ist es notwendig jedes SubVI vor der Einbindung unter einem neuen Namen zu kopieren. Genau diese Vorgehensweise muß beim Einsatz der BkComLV VI's in eigenen VI's befolgt werden.

Die BkComLV VI's

Das BkComLV besteht aus vier eigenständigen VI:

- Open-BkComLV
- Close-BkComLV
- Read-BkComLV
- ReadWrite-BkComLV

Es ist leicht, diese VI in eigenen VI, als SubVI's einzusetzen, und somit die Funktionalität der BkCom DLL von einem LabVIEW-Programm aus einzubinden.

Weitergehende Informationen zur Funktionsweise und Aufgabengebiet der BkCom-DLL finden Sie in den Hilfedateien zum KS8000-Ocx.

Zusätzlich für den Betrieb der BkComLV, ist die Einbindung der mitgelieferten BkComLV DLL vorzunehmen.

Einbinden der LabView-DLL in LabView

Im folgenden wird die prinzipielle Vorgehensweise bei der Erstellung eines LabVIEW VI mit eingebundenen BkComLV-VI dargestellt.

- Je nach Anzahl zu öffnender Kommunikationsports, müssen die VI's Open-BkComLV und Close-BkComLV entsprechend oft kopiert und zweckmäßiger Weise umbenannt werden. Originalkopien befinden sich im Verzeichnis (KS8000-Install\LabView)
- Je nach Anzahl der abzusetzenden Lese- bzw. Lese-Schreibfunktionen auf die Prozeßabbilder von BK8x00 Buskopplern muß eine entsprechende Anzahl von Kopien der VI's ‚Read-BkComLV' bzw. ‚Read Write-BkComLV' gemacht werden, wiederum zweckmäßiger Weise mit Umbennungen verbun-

den.

- Aus der Funktion-Palette via ‚Select User VI', die kopierten BkComLV-VI auswählen und innerhalb des eigenen VI-Blockdiagramms plazieren.
- Auf der Frontplatte des eigenen VI, geeignete LabVIEW Variablen (Controls und Indicators) erstellen (siehe Ein- und Ausgabedaten der BkComLV-VI) und mit den BkComLV-SubVI-Knoten verdrahten.

Referenz

Open-BkComLV-VI

Öffnet die serielle Schnittstelle.

Das Open-BkComLV.vi öffnet den angegebenen Kommunikationsport und liefert einen Handle auf diesen zurück.

Eingabedaten

Baudrate
Long Control
Wertebereich { 9600, 19200, 38400 }

CommPort
Long Control
Wertebereich { 1, : , n }

Timeout
Long Control
Zeit in Millisekunden, Wertebereich { 0, N }

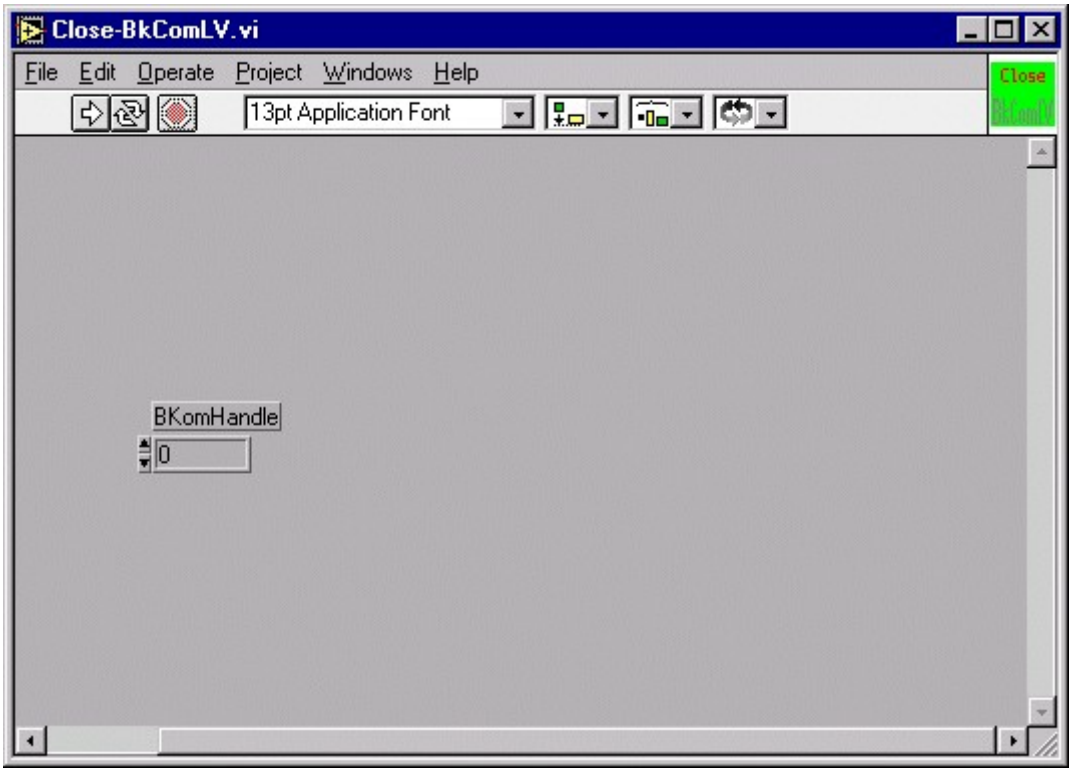
BkxTyp
Long Control
Konstante: BkxTyp = { 1, // BK8000 2 // BK8100 }

Anmerkung:Alle Eingabeparameter sind zwingend.

Ausgabedaten
BkcomHandle - Long Indicator
Wertebereich: BkcomHandle - Eindeutige Zahl <> 0

Close-BkComLV-VI

Schließt die serielle Schnittstelle.



Das Close-BkComLV.vi schliesst den angegebenen Kommunikationsport.

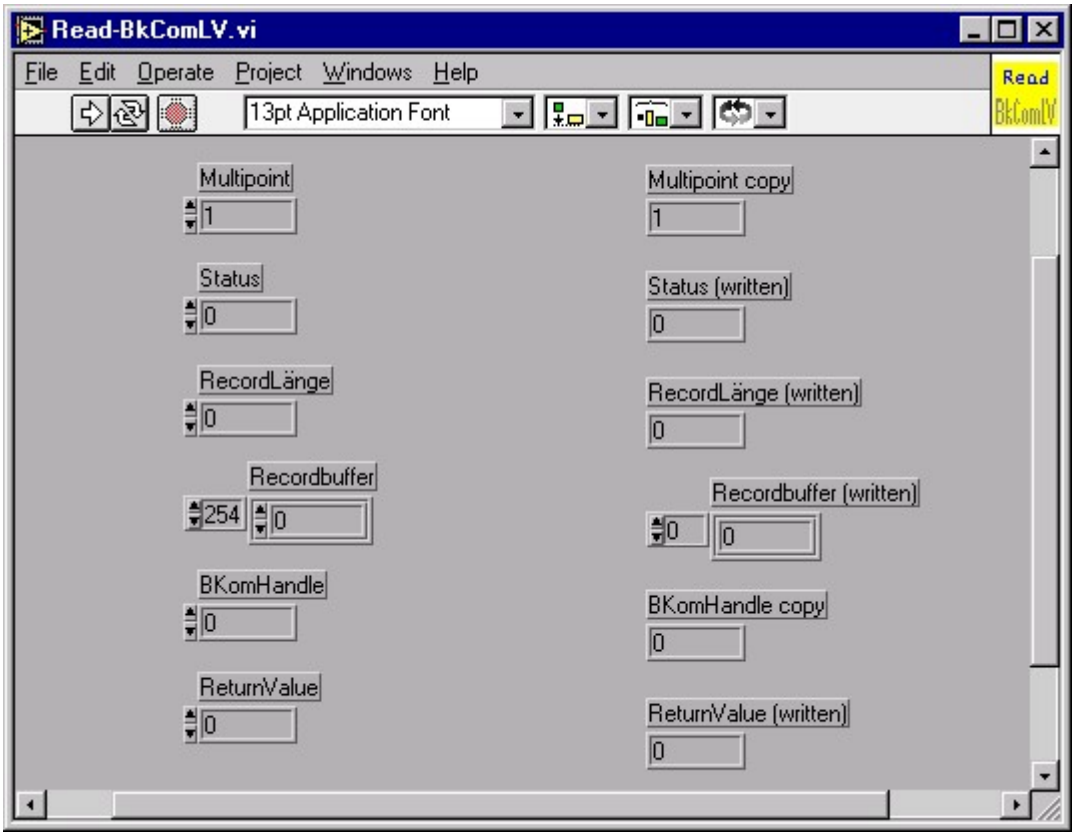
Eingabedaten
BkcomHandle - Long Indicator
Wertebereich: BkcomHandle - Eindeutige Zahl <> 0

Anmerkung: Alle Eingabeparameter sind zwingend.

Ausgabedaten
Keine

Read-BkComLV-VI

Liest Prozeßdaten der seriellen Schnittstelle.



Bei dem Read-BkComLV.vi handelt es sich um einen synchronen Kommunikationsaufruf zum Auslesen des gesamten Eingangsprozeßabbildes eines BK8x00 Buskopplers.

Die Größe des Prozeßabbildes hängt von der Anzahl und Art der gesteckten Klemmen an diesem Koppler ab. Der Empfangspuffer ist vom Typ Long (32bit), es wird jedoch vom Koppler nur in das Low-Word übertragen (siehe auch Beispiel BkComOCX).

Eingabedaten

Multipoint
Long Control
Wertebereich: Stationsadresse des Buskopplers

Status
Long Control
Wertebereich: irrelevant, Standardwert 0

RecordLänge
Long Control
Wertebereich: irrelevant, Standardwert 0

Recordbuffer
Eindimensionales Array von Long Controls
Wertebereich: irrelevant, Standardwert 0

ControlsBkcomHandle
Long Control
Wertebereich: BkcomHandle - Eindeutige Zahl <> 0

ControlReturnValue
Long Control
Wertebereich: irrelevant, Standardwert 0

Anmerkung:
Irrelevant bedeutet, daß der übergebene Wert keinen Einfluß auf die Funktion hat. Die Übergabe ist aber notwendig, um Typ und Größe bekanntzugeben, damit die Ausgabedaten korrekt zurückgeliefert werden.

Ausgabedaten

Multipoint copy
Long Indicator
Wertebereich: Stationsadresse des Buskopplers

Status (written)
Long Indicator
Wertebereich: Zustandsstatus des Buskoppler

RecordLänge (Written)
Long Indicator
Wertebereich: Anzahl gelesener Prozeßdaten

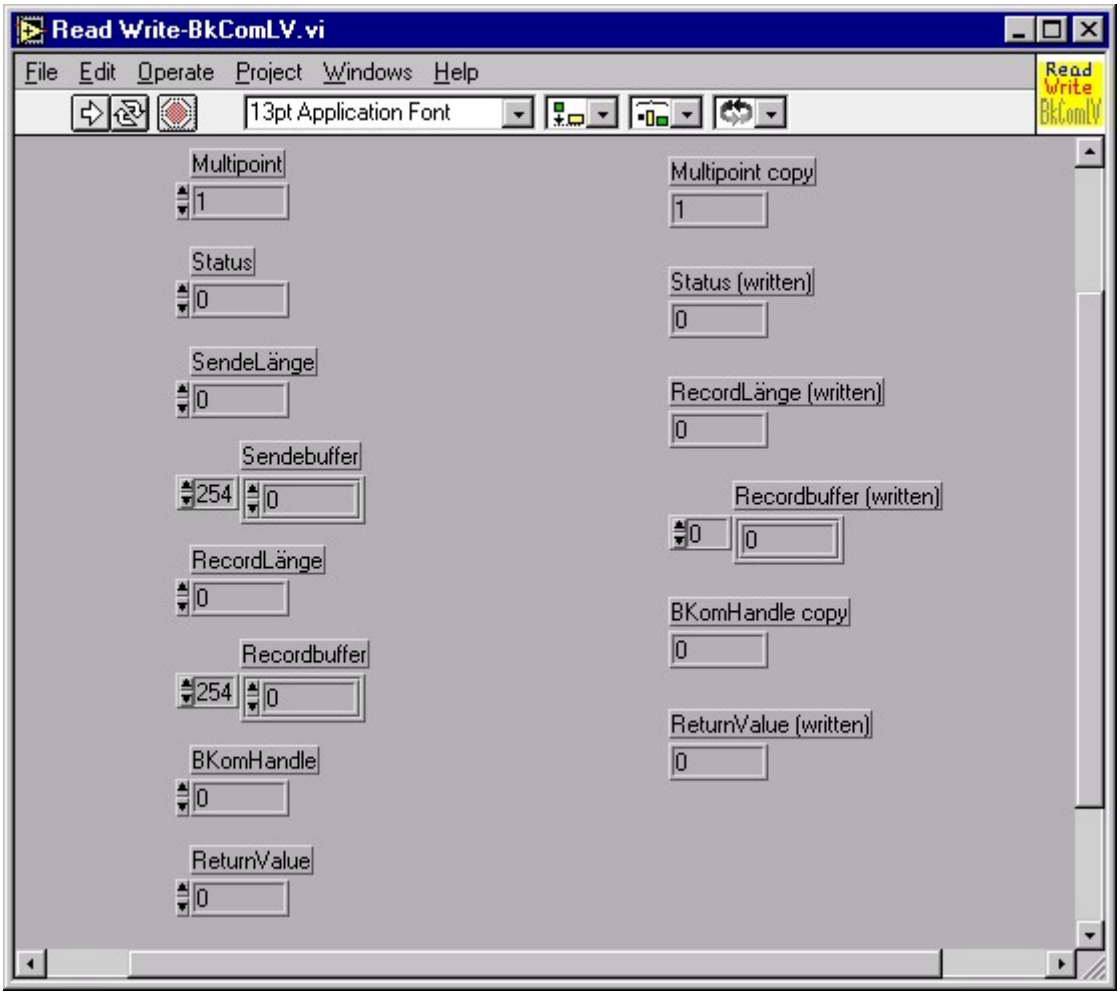
Recordbuffer (Written)
Eindimensionales Array von Long Indicators
Wertebereich: Gelesene Prozeßwerte

ControlsBkcomHandle copy
Long Indicator
Wertebereich: BkcomHandle - Eindeutige Zahl <> 0 muß von einem Open-BkComLV.vi herstammen

ControlReturnValue (written)
Long Indicator
Wertebereich: 0 OK, ... <> 0 Fehler

ReadWrite-BkComLV-VI

Schreibt und liest Prozeßdaten der seriellen Schnittstelle.



Bei dem Read Write-BkComLV.vi handelt es sich um einen synchronen Kommunikationsaufruf zum Schreiben des gesamten Ausgangs- und Auslesen des gesamten Eingangsprozeßabbildes eines BK8x00 Buskopplers. Die Größe des gelesenen Prozeßabbildes hängt von der Anzahl und Art der gesteckten Klemmen an diesem Koppler ab.

Es MUSS das gesamte Prozeßausgangsabbild beschrieben werden, es ist nicht möglich nur einen Teil bzw. Ausschnitt zu beschreiben.

Die Sende- bzw. Empfangspuffer sind vom Typ Long (32bit), es wird jedoch nur das Low-Word zum/vom Koppler übertragen (siehe auch Beispiel BkComOCX).

Eingabedaten

Multipoint
Long Control
Wertebereich: Stationsadresse des Buskopplers

Status
Long Control
Wertebereich: irrelevant, Standardwert 0

SendeLänge
Long Control
Wertebereich: Anzahl der zu schreibenden Worte (ein Wort in einem Long Wert) ab Offset 0 im Prozeßausgangsabbild

Sendbuffer
Eindimensionales Array von Long Controls
Wertebereich: zu sendende Datenworte

RecordLänge
Long Control
Wertebereich: irrelevant, Standardwert 0

Recordbuffer
Eindimensionales Array von Long Controls
Wertebereich: irrelevant, Standardwert 0

ControlsBkcomHandle
Long Control
Wertebereich: BkcomHandle - Eindeutige Zahl <> 0

ControlReturnValue
Long Control
Wertebereich: irrelevant, Standardwert 0

Anmerkung:
Irrelevant bedeutet, daß der übergebene Wert keinen Einfluß auf die Funktion hat. Die Übergabe ist aber notwendig, um Typ und Größe bekanntzugeben, damit die Ausgabedaten korrekt zurückgeliefert werden.

Ausgabedaten

Multipoint copy
Long Indicator
Wertebereich: Stationsadresse des Buskopplers

Status (written)
Long Indicator
Wertebereich: Zustandsstatus des Buskoppler

RecordLänge (Written)
Long Indicator
Wertebereich: Anzahl gelesener Prozeßdaten

Recordbuffer (Written)
Eindimensionales Array von Long Indicators
Wertebereich: Gelesene Prozeßwerte

ControlsBkcomHandle copy
Long Indicator
Wertebereich: BkcomHandle - Eindeutige Zahl <> 0 muß von einem Open-BkComLV.vi herstammen

ControlReturnValue (written)
Long Indicator
Wertebereich: 0 OK, ... <> 0 Fehler

Beispiele

Allgemein

Zur Vorbereitung müssen ein bzw. zwei Bk8x00 Buskoppler mit mind. 3 digitalen Ausgangsklemmen (z.B. KL2012) gesteckt werden, optional können weitere Eingangsklemmen dahintergesteckt werden. Die Buskoppler sind an die COM-Port(s) anzuschließen. Die korrekten Port-Nummer sowie die Stationsnummer der Koppler (Multipoint) sind in die Beispiel-VI einzutragen.

Alle Beispiele werden durch ‚Run' (Strg+r) innerhalb von LabVIEW gestartet, und durch Umlegen des Programm-Schalters auf ‚Beenden' ordentlich wieder beendet.

Beim erfolgreichen Test ist an den Ausgangsklemmen ein ‚Lauflicht' Effekt zu beobachten. Gesetzte Signale an den Eingangsklemmen müssen eine Wertänderung in den entsprechenden Recordbuffer-Arrays zur Folge haben.

Folgende Beispiele stehen zur Verfügung:

Beschreibung	Quelltexte
Beispiel 1: Lauflicht-Effekt läuft über einen Kommunikationsport, einen Buskoppler in einem VI ab: BkcomLVCompleteSubVI.vi	OnePort
Beispiel 2: Lauflicht-Effekt läuft über zwei Kommunikationsports, zwei Buskopplern in einem VI ab: DblBKcomLVCompleteSubVI.vi.	TwoPort
Beispiel 3: Effekt läuft über zwei Kommunikationsports, zwei Buskoppler und in zwei gleichzeitig zu öffnenden VI's ab: BKcomLVCompleteSubVI_1.vi und BKcomLVCompleteSubVI_2.vi.	OneAndOnePort

Anhang

BkComBkxType

```
enum BkComBKxType
{
        BkxType_RS485 = 1, // BK8000
        BkxType_RS232 = 2 // BK8100
};
```

BkComBaud

```
enum BkComBaud
{
        Baud_9600 = 9600,
        Baud_19200 = 19200,
        Baud_38400 = 38400
};
```

BkComErr

```
enum BkComErr
{
        ComErrNotImplemented = -1,   // Funktion ist nicht implementiert
        ComErrNo = 0,                // Kein Fehler
        ComErrTimeout1 = 1,// Timeout: Keine Reaktion vom Koppler
        ComErrTimeout2 = 2,// Timeout: Keine vollständige Kommunikation mit Koppler
        ComErrCRC = 3,               // Fehlerhafte Prüfsumme in Kommunikation
        ComErrTargetNr = 4,
        ComErrTableNr = 5,
        ComErrOffset = 6,
        ComErrDataLength = 7,
        ComErrMultipoint = 8,
        ComErrDataBuff = 9,
        ComErrStartPattern = 10,     // Falscher Telegrammheader im Koppler-Response Tele-
gramm
        ComErrSendTel = 11,// Fehler bei Schreiben auf serielle PC-Schnittstelle
        ComErrIdent = 12,
        ComErrRegResponse = 13,      // Im Response Telegramm des Kopplers ist das Fehler-
flag gesetzt
        ComErrHexFileFault = 14,
        ComErrChecksumFault = 15,
        ComErrAddressMappingFault = 16,
        ComErrFaultInByte = 17,
        ComErrWrongAck = 18,
        ComErrWrongBtlAck = 19,
        ComErrReadBackFault = 20
};
```

Koppler-Status

```
Bit      0 : Klemmbusfehler
Bit      1 : Konfigurationsfehler
Bit      2 :
Bit      3 :
Bit      4 : Ausgangs-Prozeßdaten: Fehler bei zu schreibender Länge
Bit      5 :
Bit      6 :
Bit      7 :
```