

# Installations- und Programmierhandbuch

Version 4.2

▶ [www.bmcm.de](http://www.bmcm.de)

## STR-xxx

ActiveX Controls für  
USB, PCI und ISA Messsysteme





# Inhaltsverzeichnis

<b>1 Überblick</b>	<b>5</b>
1.1 Einleitung	5
1.2 BMC Messsysteme GmbH	7
1.3 Urheberrechte	8
1.4 Schnelleinstieg	9
<b>2 Installation</b>	<b>10</b>
2.1 Systemvoraussetzungen	10
2.2 Installation des ActiveX Controls	10
<b>3 Einbindung in Programmiersprachen</b>	<b>14</b>
3.1 Einbindung in Visual Basic® 4.0 - 6.0	14
3.2 Einbindung in Delphi® 3.0 - 5.0	17
3.3 Einbindung in Visual C++™ 5.0 - 6.0	20
3.4 Einbindung in C++®	22
<b>4 Programmierung</b>	<b>23</b>
4.1 Hinweise	23
4.2 Eigenschaften	24
4.2.1 Eigenschaften der meM-/USB-Geräte	24
4.2.2 Eigenschaften der PC16/PC20/P1000	27
4.2.3 Eigenschaften der PCI-BASE50/300/1000	29
4.2.4 Eigenschaften der PIO24II/PIO48II	31
4.3 Funktionen	33
4.3.1 Funktionen der meM-/USB-Geräte	33
4.3.1.1 Allgemeine Funktionen	33
4.3.1.2 Analoge Funktionen	34
4.3.1.3 Digitale Funktionen	34
4.3.1.4 Zählfunktionen (meM-INC)	37
4.3.2 Funktionen der PC16/PC20/P1000	39

4.3.2.1	Analoge Funktionen	39
4.3.2.2	Digitale Funktionen	40
4.3.3	Funktionen der PCI-BASE50/300/1000	41
4.3.3.1	Allgemeine Funktionen	41
4.3.3.2	Analoge Funktionen	41
4.3.3.3	Digitale Funktionen	42
4.4	Funktionen der PIO24II/PIO48II	45
4.5	Programmierbeispiele	47
4.5.1	Programmierbeispiele für meM-/USB-Geräte	47
4.5.2	Programmierbeispiele für PC16/PC20/P1000	50
4.5.3	Programmierbeispiele für PCI-BASE50/300/1000	51
4.5.4	Programmierbeispiele für PIO24II/PIO48II	52
<b>5</b>	<b>Index</b>	<b>53</b>

# 1 Überblick

## 1.1 Einleitung

Werden Funktionsabläufe in der Mess- und Steuerungstechnik verlangt, welche nicht mit einer Standardsoftware gelöst werden können, ist es notwendig das Problem durch Programmierung zu lösen.

Hierzu werden produktspezifische ActiveX Controls für die Messsysteme der BMC Messsysteme GmbH für alle Programmiersprachen unter Windows® 2000/XP angeboten, in welche ActiveX Komponenten geladen werden können (Visual Basic®, Visual C++™, Delphi®, usw.). Dies sind im Einzelnen:

ActiveX Control	Schnittstelle	Messsysteme
<b>STR-meM</b>	USB	meM-AD/-ADDA/-ADf/-ADfo/-INC/-PIO, USB-AD12/-AD16/-PIO
<b>STR-PC</b>	ISA	PC16TR/PC20NHDL/PC20NVL/PC20TR/ P1000NV/P1000TR (im folgenden: PC16/PC20/P1000)
<b>STR-PCI</b>	PCI	PCI-BASE50/300/1000
<b>STR-PIO</b>	ISA	PIO24II/ PIO48II

Die Einstellungen der Hardware sind in der Eigenschaftenliste der jeweiligen Programmierumgebung veränderbar. Ist man mit den Fähigkeiten des Messsystems vertraut, kann so eine Steuerung der Hardware einfach und mittels aller Möglichkeiten von Windows® schnell programmiert und getestet werden.

Die Installation des produktspezifischen ActiveX Controls wird immer nach erfolgreicher Hardwareinstallation durchgeführt. Ob diese vom System erkannt wird, lässt sich unter Windows® 2000/XP im Geräte-Manager des Systems (*Start / (Einstellungen) / Systemsteuerung / System*) überprüfen. Dort ist die verwendete Hardware unter dem Eintrag "BMC Messsysteme" aufgelistet. Bei Verwendung mehrerer Messsysteme eines Typs muss das ActiveX Control nur einmal installiert werden.

Die ActiveX Controls **STR-meM**, **STR-PC**, **STR-PCI**, **STR-PIO** befinden sich auf der "Software Collection"-CD, die im Lieferumfang der Hardware inbegriffen

ist. Beim Einlegen der CD startet automatisch ein Begrüßungsbildschirm, der alle installierbaren Produkte der BMC Messsysteme GmbH logisch untergliedert auflistet.

Die Installation eines Updates erfolgt genauso wie die Erstinstallation. Wurde das ActiveX Control von unserer Homepage geladen, oder wird eine Diskettenversion verwendet, startet man die Installation direkt durch Öffnen der folgenden Datei:

- meM-/USB-Geräte:        **mem-actx.exe**
- PC16/PC20/P1000:      **plk-actx.exe**
- PCI-BASE50/300/1000: **pcibase-actx.exe**
- PIO24II/PIO48II:      **pioii-actx.exe**



**Da die USB-Geräte USB-AD12/16 und USB-PIO vollständig softwarekompatibel zu ihren Vorgängern meM-ADDA und meM-PIO sind, wird zwischen diesen USB- und meM-Geräten nicht unterschieden. Die Geräte werden bei der Programmierung als meM-ADDA und meM-PIO angesprochen.**

---

## 1.2 BMC Messsysteme GmbH

BMC Messsysteme GmbH steht für innovative Messtechnik "made in Germany". Vom Sensor bis zur Software bieten wir alle für die Messkette benötigten Komponenten an.

Unsere Hard- und Software ist aufeinander abgestimmt und dadurch besonders anwenderfreundlich. Darüber hinaus legen wir größten Wert auf die Einhaltung gängiger Industriestandards, die das Zusammenspiel vieler Komponenten erleichtern.

BMC Messsysteme Produkte finden Sie im industriellen Großeinsatz ebenso wie in Forschung und Entwicklung oder im privaten Anwenderbereich. Wir fertigen unter Einhaltung der ISO-9000-Vorschriften, denn Standards und Zuverlässigkeit sind uns wichtig - für Sie und für uns!

Neueste Informationen finden Sie im Internet auf unserer Homepage unter <http://www.bmcm.de>.



▶ w w w . b m c m . d e

## 1.3 Urheberrechte

Die Programmierschnittstellen **STR-meM**, **STR-PC**, **STR-PCI**, **STR-PIO** (ActiveX Controls) wurden mit größtmöglicher Sorgfalt entwickelt, gefertigt und geprüft. BMC Messsysteme GmbH gibt keine Garantien, weder in Bezug auf dieses Handbuch noch in Bezug auf die in diesem Buch beschriebene Software, deren Qualität oder Verwendbarkeit für einen bestimmten Zweck. BMC Messsysteme GmbH haftet in keinem Fall für direkt oder indirekt verursachte oder folgende Schäden, die entweder aus unsachgemäßer Bedienung oder aus irgendwelchen Fehlern resultieren. Änderungen, die dem technischen Fortschritt dienen, bleiben uns vorbehalten.

Die Programmierschnittstellen **STR-meM**, **STR-PC**, **STR-PCI**, **STR-PIO** (ActiveX Controls) sowie das vorliegende Handbuch und sämtliche darin verwendeten Namen, Marken, Bilder und sonstige Bezeichnungen und Symbole sind ihrerseits gesetzlich sowie aufgrund nationaler und internationaler Verträge geschützt. Die dadurch begründeten Rechte, insbesondere die der Übersetzung, des Nachdrucks, der Entnahme von Abbildungen, der Funksendung, der Wiedergabe auf photomechanischem oder ähnlichem Wege bleiben, auch bei nur auszugsweiser Verwertung, vorbehalten. Die Reproduktion der Programme und des Programmhandbuchs sowie die Weitergabe an Dritte ist nicht gestattet. Ihre rechtswidrige Verwendung oder sonstige rechtliche Beeinträchtigung wird straf- und zivilrechtlich verfolgt und kann zu empfindlichen Sanktionen führen.

**Copyright © 2006**

Stand: 01. November 2006

**BMC Messsysteme GmbH**

Hauptstraße 21  
82216 Maisach  
DEUTSCHLAND

Tel.: +49 8141/404180-1

Fax: +49 8141/404180-9

E-Mail: [info@bmcm.de](mailto:info@bmcm.de)

---

## 1.4 Schnelleinstieg

---



**Führen Sie die Hardwareinstallation, wie in Ihrer Dokumentation beschrieben, vor Installation des ActiveX Controls durch und überprüfen Sie, ob die Hardware vom PC erkannt wird.**

---

- Installieren Sie das ActiveX Control **STR-meM, STR-PC, STR-PCI, STR-PIO** mit Hilfe der beigelegten "Software Collection"-CD. Selektieren Sie im CD-Starter den Eintrag "Produkte" und anschließend unter der Schnittstelle Ihres Messsystems (USB, PCI, ISA) die verwendete Hardware.
- Die Installation wird nach Auswahl des ActiveX Controls gestartet, das auf der jeweiligen Produktseite im Abschnitt "Programmierung" verfügbar ist.
  - meM-/USB-Geräte: "STR-meM"
  - PC16/PC20/P1000: "STR-PC"
  - PCI-BASE50/300/1000: "STR-PCI"
  - PIO24II/PIO48II: "STR-PIO"
- Führen Sie das Installationsprogramm schrittweise durch. Die entsprechende OCX-Datei wird ins angegebene Verzeichnis kopiert.
- Das ActiveX Control muss nun in die Programmierumgebung eingebunden werden. In Delphi<sup>®</sup> geschieht dies durch eine Installation in ein bestehendes Package Ihrer Wahl. Selektieren Sie dazu im Menü *Projekt / Komponenten* (Visual Basic<sup>®</sup>) bzw. *Komponenten / ActiveX importieren* (Delphi<sup>®</sup>) den folgenden Eintrag:
  - meM-/USB-Geräte: "meM-ActiveX Control module"
  - PC16/PC20/P1000: "PC16/PC20/P1000 ActiveX Control module"
  - PCI-BASE50/300/1000: "PCI-Base ActiveX Controls 1.2"
  - PIO24II/PIO48II: "PIO II ActiveX Control module"
- Die Geräte spezifischen Icons, Eigenschaften und Funktionen stehen anschließend zur Programmierung des Messsystems zur Verfügung.

## 2 Installation

### 2.1 Systemvoraussetzungen

- jeder PC mit Windows® 2000/XP
- Programmiersprache mit ActiveX Control Unterstützung:  
z. B. Visual Basic®, Delphi®, Visual C++™, Borland C++ Builder, Visual Basic .NET®, Visual C++ .NET®

### 2.2 Installation des ActiveX Controls

---

---



- **Führen Sie die Installation der Hardware vor Installation des ActiveX Controls STR-meM, STR-PC, STR-PCI, STR-PIO durch und überprüfen Sie, ob die Karte vom System erkannt wird. Setzen Sie die Installation fort, wenn das Messsystem betriebsbereit und konfliktfrei ist.**
  - **Bei Verwendung mehrerer Geräte des gleichen Typs muss das ActiveX Control nur ein einziges Mal installiert werden.**
  - **Die Installation von Updates erfolgt ebenso wie die Erstinstallation. Eine vorherige Deinstallation ist nicht erforderlich.**
  - **Die Programmierung der PCI-BASE50/300/1000 mit Hilfe von STR-PCI ist nur bei Verwendung von Analogmodulen möglich.**
- 
- 

Um das ActiveX Control **STR-meM, STR-PC, STR-PCI, STR-PIO** zu installieren, legen Sie die im Lieferumfang inbegriffene "Software Collection"-CD ins CD-ROM Laufwerk. Es erscheint nach wenigen Sekunden ein Begrüßungsbildschirm, auf dem die auf der CD enthaltenen Produkte aufgelistet werden.

Ist die AutoPlay Funktion Ihres CD-ROMs nicht eingeschaltet, öffnen Sie bitte die Datei **index.html** oder **setup.exe**.

The screenshot shows the BMC Messsysteme GmbH homepage with the following sections:

- Homepage | Englisch** (top left)
- HOME** (top center)
- Logo** (top right): **bmc** messsysteme gmbh
- Navigation:**
  - NEXTVIEW®**: Wählen Sie diesen Punkt installieren. Dort finden Handbücher und Daten.
  - PRODUKTE**: Datenblätter, Treiber, Programmierschnittstelle Messsysteme GmbH.
  - PROGRAMMIERUNG**: Programmierschnittstelle.
- USB (USB-AD/PIO MEM-AD/ADDA/ADF/ADFO/PIO/INC/LOG)**: Datenblätter, Treiber, ActiveX Controls und Software für die analogen USB Geräte USB-AD und meM-AD/ADDA/ADf/ADfo, die digitalen USB Geräte USB-PIO und meM-PIO/INC, sowie den Datenlogger meM-LOG.
  - USB-AD
  - meM-AD
  - meM-ADDA
  - meM-ADF
  - meM-ADfo
  - meM-LOG
  - USB-PIO
  - meM-PIO
  - meM-INC
- PCI (PCI-BASE50/300/1000 UND MODULE)**: Datenblätter, Treiber, ActiveX Controls und Software für die PCI Messkarten PCI-Base50/300/1000 und die Eingangsmodule MAD12f/12a/12f/16/16f und MCAN, sowie die analogen Ausgangsmodule MDA12f/12-4/16.
  - MAD
  - MDA
  - MCAN
  - PIO24II/PIO48II/PIO48I
- PROGRAMMIERUNG**: STR-meM ist ein ActiveX Control zur Ansteuerung der USB-basierten Messgeräte USB-AD/PIO und meM-AD/ADDA/ADf/ADfo/PIO/INC. Das Handbuch beschreibt ausführlich die Installation des ActiveX Controls und die Programmierung des Messsystems mit Hilfe des ActiveX Controls (erfordert Acrobat Reader®). Bitte beachten Sie, vor der Installation des ActiveX Controls den passenden Treiber für Ihr Messgerät zu installieren.
 

Produkt	Bezeichnung	Version	Größe
<a href="#">STR-meM-IG</a>	Installations- und Programmierhandbuch	4.2.341	742,8kB
<a href="#">STR-meM</a>	ActiveX Control für meM-AD, USB-AD/meM-ADDA, meM-ADf, meM-ADfo, meM-INC, USB-PIO/meM-PIO, und meM-PIO-OEM	4.2.341	1,8MB
<a href="#">STR-meM-EX</a>	Beispielprogramme für meM-AD, USB-AD/meM-ADDA, meM-ADf, meM-ADfo, meM-INC, USB-PIO/meM-PIO, und meM-PIO-OEM (inkl. Sourcecode)	4.2.341	4,0MB

Abbildung 1

Wählen Sie zuerst im CD-Starter den Eintrag "Produkte" und aufgelistet unter der verwendeten Schnittstelle das betreffende Produkt aus, um die jeweilige Produktseite zu öffnen.

Die Installation des ActiveX Controls wird durch Auswahl des folgenden Eintrags im Abschnitt "Programmierung" gestartet:

- meM-/USB-Geräte: "STR-meM"
- PC16/PC20/P1000: "STR-PC"
- PCI-BASE50/300/1000: "STR-PCI"
- PIO24II/PIO48II: "STR-PIO"

Dabei muss die Installationsdatei nicht zuerst auf Festplatte gespeichert werden, sondern kann direkt geöffnet werden.

Sollten Sie über eine Diskettenversion verfügen oder das ActiveX Control von unserer Homepage geladen haben, starten Sie die Installation direkt durch Öffnen des folgenden Installationsprogramms:

- meM-/USB-Geräte: **mem-actx.exe**
- PC16/PC20/P1000: **plk-actx.exe**
- PCI-BASE50/300/1000: **pcibase-actx.exe**
- PIO24II/PIO48II: **pioii-actx.exe**

Sie werden nun in überschaubaren Schritten durch die Installation geführt, die mit der Schaltfläche "Abbrechen" jederzeit vorzeitig gestoppt werden kann. Um mit der Installation fortzufahren betätigen Sie die Schaltfläche "Weiter", während "Zurück" einen Schritt zurückgeht.

Wählen Sie nach dem Begrüßungsbildschirm das Verzeichnis aus, in dem **STR-meM**, **STR-PC**, **STR-PCI** oder **STR-PIO** installiert wird.

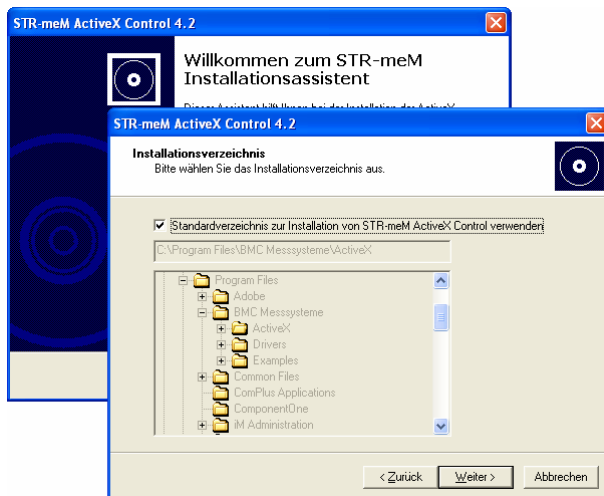


Abbildung 2

Falls Sie nicht das "Standardverzeichnis zur Installation des ActiveX Controls verwenden" wollen, entfernen Sie das Häkchen an der entsprechenden Option und geben Sie den gewünschten Verzeichnispfad an.

Nachdem festgestellt wurde, ob genügend Speicherplatz zur Verfügung steht, startet der Installationsvorgang und die erforderlichen Dateien werden ins angegebene Verzeichnis kopiert. Dort befindet sich nun das entsprechende ActiveX Control:

- meM-/USB-Geräte: **memx.ocx**
- PC16/PC20/P1000: **p1000.ocx**
- PCI-BASE50/300/1000: **pci300x.ocx**
- PIO24II/PIO48II: **pioii.ocx**

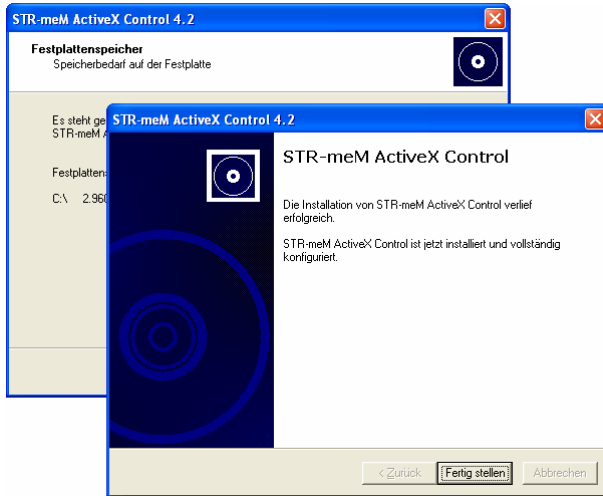


Abbildung 3

Nachdem die erfolgreiche Installation gemeldet wurde, starten Sie, falls erforderlich, den Rechner neu.

# 3 Einbindung in Programmiersprachen

## 3.1 Einbindung in Visual Basic® 4.0 - 6.0

Im Menü "Projekt" in Visual Basic® erreichen Sie über den Eintrag "Komponenten" einen Dialog, der die aktuell verwendeten ActiveX Controls auflistet.

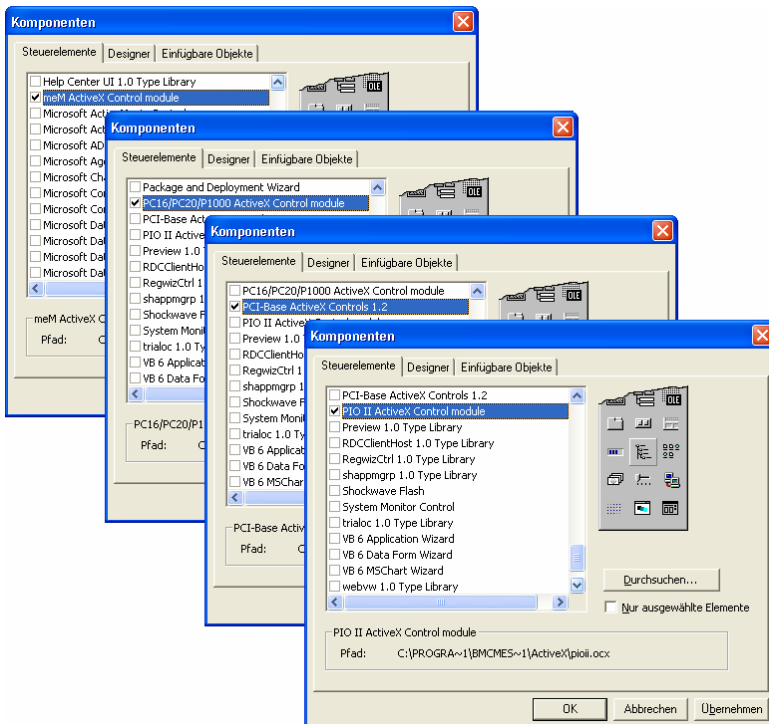


Abbildung 4

Aktivieren Sie den Eintrag des zu verwendenden ActiveX Controls, damit die Befehle zur Programmierung des bmcM Messsystems in Visual Basic® zur Verfügung stehen.

- meM-/USB-Geräte: "meM ActiveX Control module"
- PC16/PC20/P1000: "PC16/PC20/P1000 ActiveX Control module"
- PCI-BASE50/300/1000: "PCI-Base ActiveX Controls 1.2"
- PIO24II/PIO48II: "PIO II ActiveX Control module"

In der Toolbar von Visual Basic® erscheint nun das entsprechende Icon für das Messsystem, das das geladene ActiveX Control zur Verfügung stellt.

Klicken Sie wie gewohnt darauf und ziehen Sie einen Rahmen in der Form auf, in der die Hardware verwendet werden soll. Dieser Rahmen wird nach Einfügen des Objekts wieder auf die ursprüngliche Icongröße reduziert.

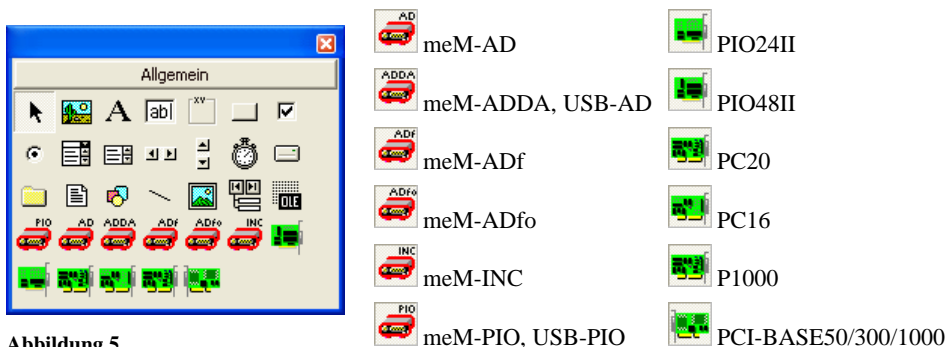


Abbildung 5

Öffnet man den Objektkatalog (Menü "Ansicht" oder <F2>-Taste) befindet sich nun in der Auflistung aller Bibliotheken ein zusätzlicher Eintrag für die Produktgruppen spezifische Objektbibliothek. Wählt man diese aus, werden die enthaltenen Produktklassen aufgelistet.

Gerätetyp	Objektbibliothek	zur Verfügung gestellte Klassen
meM-/USB-Geräte	<b>MEMXLib</b>	meMAD, meMADDA, meMADf, meMADfo, meMINC, meMPIO
PC16/PC20/P1000	<b>PC20Lib</b>	P1000, PC16, PC20
PCI-BASE50/300/1000	<b>PCIBaseLib</b>	PCIBase
PIO24II/PIO48II	<b>PIOIILib</b>	PIO24II, PIO48II

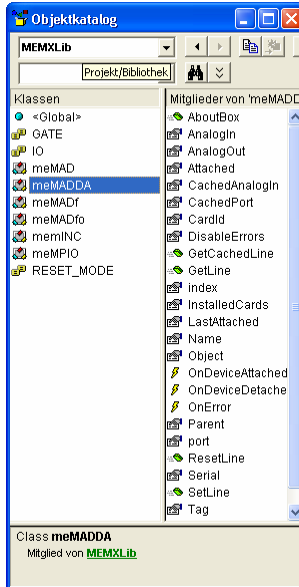


Abbildung 6

Rechts wird eine Übersicht über alle verfügbaren Eigenschaften und Funktionen der selektierten Klasse (s. "Eigenschaften", S. 24 und "Funktionen", S. 33) angezeigt.



**Da die USB-Geräte USB-AD12/16 und USB-PIO vollständig softwarekompatibel zu ihren Vorgängern meM-ADDA und meM-PIO sind, wird zwischen diesen USB- und meM-Geräten nicht unterschieden. Die Geräte werden bei der Programmierung als meM-ADDA und meM-PIO angesprochen.**

---

## 3.2 Einbindung in Delphi® 3.0 - 5.0

In Delphi® muss vor der ersten Nutzung des bmc Messsystems zuerst das bei der Installation angegebene Verzeichnis kopierte ActiveX Control angemeldet und anschließend in Delphi® installiert werden. Gehen Sie dazu in folgenden Schritten vor:

Starten Sie Delphi® und rufen Sie im Menü "Komponente" den Befehl "ActiveX importieren..." auf.

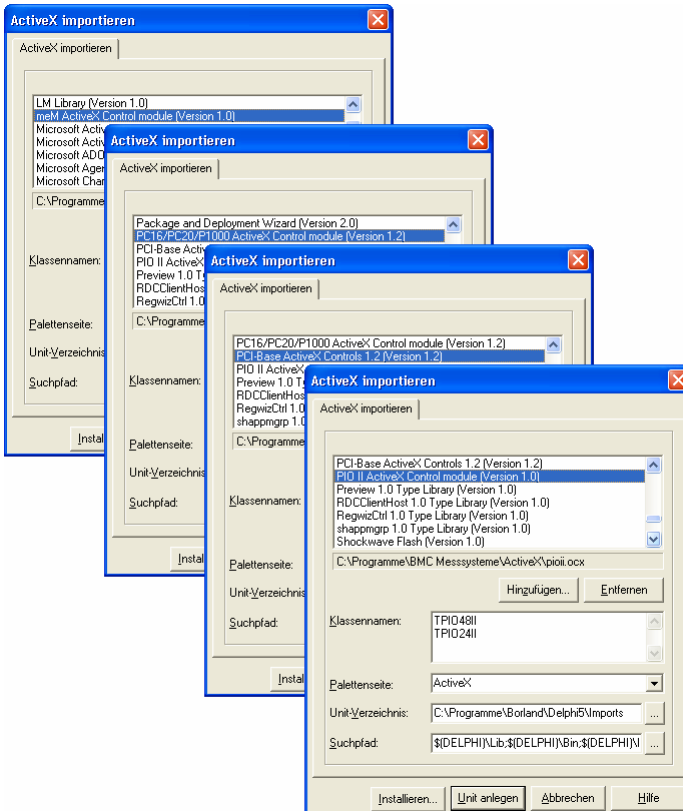


Abbildung 7

Klicken Sie hier den Eintrag des installierten ActiveX Controls an.

- meM-/USB-Geräte: "meM ActiveX Control module (Version 1.0)"
- PC16/PC20/P1000: "PC16/PC20/P1000 ActiveX Control module (Version 1.2)"
- PCI-BASE50/300/1000: "PCI-Base ActiveX Controls 1.2 (Version 1.2)"
- PIO24II/PIO48II: "PIO II ActiveX Control module (Version 1.0)"

Drücken Sie die Schaltfläche "Installieren" und bestätigen Sie anschließend mit OK die Installation ins bestehende Package.

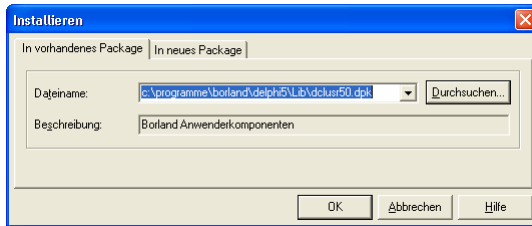


Abbildung 8

Das ausgewählte Package wird nun neu kompiliert, damit die Informationen über das neue ActiveX Control darin aufgenommen werden. Nachdem dieser Vorgang abgeschlossen ist, erhalten Sie eine Rückmeldung über die vorgenommenen Änderungen.



Abbildung 9

Dabei wurden Importdateien für **MEMXLib\_TLB**, **PCIBaseLib\_TLB**, **PC20Lib\_TLB** bzw. **PIOIILib\_TLB** erzeugt, die für alle Programme mit dem jeweils verwendbaren bmcM Messsystem verwendet werden.



### 3.3 Einbindung in Visual C++™ 5.0 - 6.0

Visual C++™ 5.0/6.0 bietet mit Hilfe der Preprozessoranweisung **#import** die Möglichkeit einfach COM-Schnittstellen in ein C++®-Programm zu integrieren. Folgende Beispielcodes demonstrieren das Vorgehen. Bitte beachten Sie, dass diese Codestücke sowie alle anderen Beispiele in diesem Handbuch aus Gründen der Einfachheit bewusst auf eine Fehlerbehandlung verzichten. Selbstverständlich muss diese in selbst geschriebenen Programmen realisiert werden.

- **meM-/USB-Geräte:**

```
meM-PIO
...
#import "c:\programme\bmc messsysteme\activex\memx.ocx"
/* wie bei Installation angegeben */
...
MEMXLib::_DmeMPIOPtr pio;
...
pio.CreateInstance (__uuidof(MEMXLib::meMPIO));
if (pio == NULL)
/* Fehlerbehandlung */
...
if (pio->InstalledCards == 0)
printf ("meM-PIO nicht angeschlossen!\n");
...
```

- **PCI-BASE50/300/1000:**

```
PCI-BASE50
PCI-BASE300
PCI-BASE1000
...
#import "c:\programme\bmc messsysteme\activex\pci300x.ocx"
/* wie bei Installation angegeben */
...
PCIBaseLib::_DPci300xPtr pci_base;
...
pci_base.CreateInstance
(__uuidof(PCIBaseLib::PciBase));

if (pci_base == NULL)
/* Fehlerbehandlung */
...
if (pci_base->InstalledCards == 0)
printf ("Keine PCI-BASE vorhanden!\n");
...
```

- **PC16/PC20/P1000:**

```

PC16      ...
PC20      #import "c:\programme\bmc messsysteme\activex\p1000.ocx"
P1000     /* wie bei Installation angegeben */

...

PC20Lib::_DPC20Ptr pc20;
PC20Lib::_DP1000Ptr p1000;

...

pc20.CreateInstance (__uuidof(PC20Lib::PC20));
p1000.CreateInstance (__uuidof(PC20Lib::P1000));

if (pc20 == NULL || p1000 == NULL)
/* Fehlerbehandlung */
...
if (pc20->InstalledCards == 0)
    printf ("Keine PC16/PC20 Karte vorhanden!\n");
...

if (p1000->InstalledCards == 0)
    printf ("Keine P1000 Karte vorhanden!\n");
...
    
```

- **PIO24II/PIO48II:**

```

PIO24II   ...
PIO48II   #import "c:\programme\bmc messsysteme\activex\pioii.ocx"
          /* wie bei Installation angegeben */

...

PIOIILib::_DPIO48IIPtr pioii;

...

pioii.CreateInstance (__uuidof(PIOIILib::PIO48II));
if (pioii == NULL)
/* Fehlerbehandlung */
...
if (pioii->InstalledCards == 0)
    printf ("Keine PIO24II/48II vorhanden!\n");
...
    
```

## 3.4 Einbindung in C++®

Die ActiveX Controls **STR-meM**, **STR-PC**, **STR-PCI**, **STR-PIO** lassen sich auch ohne die Microsoft® spezifische **#import**-Anweisung und die zugehörigen Compiler Support Klassen verwenden. Dazu ruft man die entsprechenden OLE und COM Funktionen direkt auf. Die Eigenschaften und Methoden des installierten ActiveX Controls sind in diesem Fall über die IDispatch Schnittstelle erreichbar. Eine ausführliche Beschreibung der notwendigen OLE und COM Funktionen findet sich in der entsprechenden Dokumentation von Microsoft®.

# 4 Programmierung

## 4.1 Hinweise

Eine Übersicht über alle zur Verfügung stehenden Eigenschaften und Funktionen der verwendbaren Messsysteme ist im Objektkatalog (Menü "Ansicht" oder <F2>-Taste, s. Abbildung 6) in der Bibliothek **MEMXLib**, **PC20Lib**, **PCIBaseLib** bzw. **PIOILib** zu sehen. Alle folgenden Beispiele zu den Befehlen sind in Visual Basic® ausgeführt.

Das Eigenschaftenfenster zeigt immer die Parameter des selektierten Objekts 'CardId' im Arbeitsbereich an.

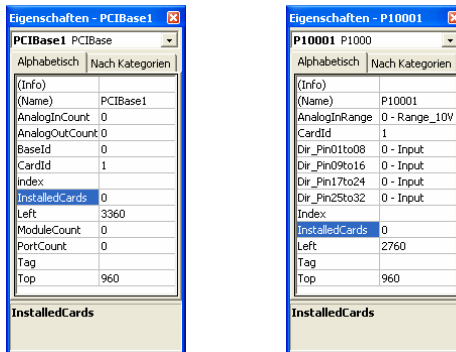


Abbildung 12



Die meisten Eigenschaften des ActiveX Controls sind nur zur Laufzeit einstellbar. Damit können sie nicht über das Eigenschaftenfenster verändert werden, da der Treiber für das Messsystem erst mit dem Start des Programms geöffnet wird.

## 4.2 Eigenschaften

Die Beschreibung der allgemeinen Einträge (**About**), **Left**, (**Name**), **index**, **Tag** und **Top** entnehmen Sie bitte der Dokumentation Ihrer Programmiersprache. Die Geräte spezifischen Eigenschaften werden alphabetisch geordnet im Folgenden beschrieben.

### 4.2.1 Eigenschaften der meM-/USB-Geräte

- **ATTACHED:** True/False  
Zeigt an, ob das selektierte Gerät (Nummer in 'CardId') am USB-Bus angeschlossen ist.

Dieses Beispiel zeigt auf, wie der USB-Bus nach allen angeschlossenen USB-AD/meM-ADDA Geräten abgesucht wird.

```
for i = 1 to meMADDA1.LastAttached
    meMADDA1.CardId = i
    if meMADDA1.Attached then... 'Device found
next
```

- **CARDID:** 1..127  
Die Geräte eines Typs (USB-PIO/meM-PIO, USB-AD/meM-ADDA, meM-AD, meM-ADf, meM-ADfo, meM-INC) sind in der Reihenfolge des Ansteckens durchnummeriert. Unter **CardId** wird das entsprechende meM-Gerät selektiert. Für jedes am USB-Bus angeschlossene Gerät, das mit **CardId** selektiert wird, wird **Attached = True**.

Dieses Beispiel wählt nacheinander das erste und das zweite meM-ADf aus.

```
meMADf1.CardId = 1
meMADf1.CardId = 2
```



Erst wenn der Eigenschaft **CardId** ein Wert zugewiesen wurde, können die Eigenschaften des ActiveX Controls **STR-meM** genutzt werden.

- DIRPORT1, DIRPORT2, DIRPORT3H, DIRPORT3L:** (USB-PIO/meM-PIO) 0-DirIn / 1-DirOut  
 Die 24 Digitalkanäle der USB-PIO/meM-PIO können portweise in 8er- bzw. 4er-Gruppen auf Ein- oder Ausgang gesetzt werden. Dafür existieren vier Eigenschaften (s. Tabelle), die die Werte **0-DirIn** bzw. **1-DirOut** annehmen.

Folgende Tabelle legt fest, welche Leitungen der USB-PIO/meM-PIO mit den entsprechenden Richtungsvariablen eingestellt werden können:

Variable	Leitungen
<b>DirPort1</b>	1..8
<b>DirPort2</b>	9..16
<b>DirPort3L</b>	17..20
<b>DirPort3H</b>	21..24

Die Befehle zum Setzen und Abfragen der Digitalein- bzw. Ausgänge werden auf Seite 34 beschrieben.



Wird ein Port auf Ausgang gestellt, sind dessen Leitungen immer *low*.

- DISABLEERRORS:** True/False  
 Schaltet den Fehlerreport ein oder aus. Ist diese Eigenschaft **True**, so wird bei einem Fehler der meM-Geräte der Event **OnError** ausgelöst. Bei **DisableErrors = False** muss im Programmcode die entsprechende Exceptionbehandlung stattfinden (**OnError goto ...**).

- **ENABLEGATE:** (meM-INC) True/False  
Dieser Befehl, wird er auf **True** gesetzt, schaltet meM-INC auf Frequenzmessung um. Andernfalls befindet sich das Gerät im Zählmodus.

- **GATE:** (meM-INC) 0.1, 1.0, 10.0  
Stellt die Torzeit (in Sekunden), in der die Anzahl der Zählimpulse von meM-INC gemessen wird, ein. Es stehen 3 Torzeiten zur Verfügung (0.1, 1.0, 10.0). Der Befehl allein genügt nicht, das Gerät muss zusätzlich mit **EnableGate** auf Frequenzmessung umgestellt werden.

Dieses Beispiel setzt die Torzeit des ersten meM-INC's auf 1 Sekunde und stellt dann das Gerät auf Frequenzmessung um.

```
meMINC1.Gate = 1.0
EnableGate = true
meMINC1.Counter(3) = i+s
```

- **INSTALLEDCARDS:** 1..127  
Zeigt an, wie viele meM-Geräte dieses Typs (USB-AD/-PIO, meM-AD/-ADDA/-ADf/-ADfo/-INC/-PIO) gegenwärtig am USB-Bus angeschlossen sind. Ändert sich die Zahl der angeschlossenen Geräte dieses Typs, wird der Wert entsprechend angepasst.
- **LASTATTACHED:** 1..127  
Höchste CardId am USB-Bus. CardId's höher als in **LastAttached** sind nicht vergeben worden. Dieser Wert wird mit jedem An- und Abstecken eines Geräts aktualisiert.
- **SERIAL:**  
Gibt die Seriennummer des selektierten Geräts zurück. Diese wurde in die Hardware eingegraben und ist am Gerät außen angebracht. Die Seriennummer kann dazu verwendet werden, zwischenzeitlich abgesteckte Geräte wieder zu finden.

## 4.2.2 Eigenschaften der PC16/PC20/P1000

Im Unterschied zur PC16 verfügen die PC20 und die P1000 über zwei Digitalports mit 16 (anstatt 8) Leitungen und zusätzlich über zwei Analogausgänge. Außerdem ist der Spannungsbereich der Analogeingänge der PC20/P1000 einstellbar (PC16:  $\pm 10V$ ).

- **ANALOGINRANGE:** (PC20/P1000) 0..3  
Wählt den Messbereich aus, der für alle Analogeingänge gilt. Folgende Einstellungen sind zulässig:

- 0 – Range\_10V
- 1 – Range\_5V
- 2 – Range\_2V
- 3 – Range\_1V

- **CARDID:** 1..4  
Die Karten eines Typs (PC16/PC20 oder P1000) sind in der Reihenfolge durchnummeriert, wie sie installiert wurden oder wie sie im Gerätemanager (Windows® 2000/XP: *Start / (Einstellungen) / Systemsteuerung / System*) im Eintrag "BMC Messsysteme" aufgelistet sind. Unter **CardId** wird die entsprechende PC16/PC20 oder P1000 selektiert.

Dieses Beispiel wählt nacheinander die erste und die zweite PC20 aus.

```
PC201.CardId = 1
PC201.CardId = 2
```

- **DIR\_PIN1TO8, DIR\_PIN9TO16, DIR\_PIN17TO24, DIR\_PIN25TO32:** 0..1  
Die Digitalkanäle können in 8er-Gruppen in der Richtung umgeschaltet werden. Folgende Werte können diese Properties annehmen:

- 0 – Input
- 1 – Output

Für die beiden Digitalports ergibt sich folgende Pinbelegung:

- Digitalport 1: Pin 1..8
- Pin 9..16 (PC20/P1000)

Digitalport 2: Pin 17..24  
Pin 25..32 (PC20/P1000)

Da die PC16 über zwei anstatt vier Digitalports verfügt, befinden sich nur die Einträge **Dir\_Pin1to8** und **Dir\_Pin17to24** im zugehörigen Eigenschaftsfenster.

Da **Input** und **Output** selbst Basisbefehle sind, muss bei der Programmierung die Bibliothek **PC20Lib**, in der das OCX der PC16/PC20/P1000 enthalten ist, mit aufgerufen werden.

Dieses Beispiel setzt den Digitalport 1 auf Eingang.

```
PC201Dir_Pin1to8 = pc20lib.Input
```

- **INSTALLEDCARDS:**

1..4

Zeigt an, wie viele PC-Karten dieses Typs (PC16/PC20 oder P1000) gegenwärtig im Rechner eingebaut sind. Ändert sich dies, wird der Wert entsprechend angepasst.

## 4.2.3 Eigenschaften der PCI-BASE50/300/1000

Die Programmierung der PCI-BASE50/300/1000 mit Hilfe von **STR-PCI** ist bei Verwendung von Analogmodulen (MAD/MDA) möglich.

- **ANALOGINCOUNT:** 0..32  
 Liefert die Anzahl aller analogen Eingangskanäle, die die installierten MAD-Module der PCI-BASE50/300/1000 zur Verfügung stellen.
- **ANALOGINRANGE:** 0.4  
 Wählt den Messbereich aus, der für alle Analogeingänge gilt. Der zur Verfügung gestellte Messbereich ist abhängig von den verwendeten Modulen auf der PCI-BASE50/300/1000. Folgende Einstellungen sind zulässig:

  - 0 – Range\_1V
  - 1 – Range\_2V
  - 2 – Range\_5V
  - 3 – Range\_10V
  - 3 – Range\_0\_5V
- **ANALOGOUTCOUNT:** 0.8  
 Liefert die Anzahl aller analogen Ausgangskanäle, die die installierten MDA-Module der PCI-BASE50/300/1000 zur Verfügung stellen.
- **BASEID:** 1000, 300, 50  
 Liefert '1000' für PCI-BASE1000, '300' für PCI-BASE300 und '50' für PCI-BASE50 zurück.
- **CARDID:** 1..16  
 Mit **CardId** wird eine PCI-BASE50/300/1000 selektiert. Bei der Vergabe der **CardId** wird nicht zwischen den PCI-Kartenversionen unterschieden.  
  
 Dieses Beispiel wählt nacheinander eine PCI-BASE300 und eine PCI-BASE1000 aus (unter der Voraussetzung, dass diese Karten im System installiert wurden).

```
PciBase1.CardId = 1  
PciBase1.CardId = 2
```

- **INSTALLEDCARDS:** 1..16  
Zeigt an, wie viele PCI-BASE50/300/1000 insgesamt gegenwärtig im Rechner eingebaut sind. Ändert sich dies, wird der Wert entsprechend angepasst. Es wird dabei nicht zwischen den verschiedenen PCI-BASE Versionen unterschieden.

- **MODULECOUNT:** 0..2  
Liefert die Anzahl der installierten Module einer PCI-BASE50/300/1000.

- **MODULEID(index):**  
Liefert die ID des gewählten Moduls '**index**' zurück. Diese ID beschreibt die Art des installierten Moduls. Der exakte Zahlenwert für die **ModuleId** lässt sich im Objektkatalog (Menü "Ansicht" oder <F2>-Taste, s. Abbildung 6) der Bibliothek **PCIBaseLib** entnehmen. Dazu wählt man den Eintrag **enumModuleId** und markiert das gewünschte Modul. Die entsprechende Konstante wird im grauen Feld darunter angezeigt.

Dieses Beispiel liest die **ModuleID** des Moduls 1 aus und gibt einen String mit dem Namen des Moduls zurück. Dabei wird nur angegeben, ob es sich um ein Ein-/Ausgabemodul handelt (MAD/MDA) und dessen Auflösung (12/16).

```
Dim i As Long, s As String  
i = PciBase1.ModuleId(1)  
Select Case i  
    Case pcibaselib.MAD12: s = "MAD12"  
    Case pcibaselib.MAD16: s = "MAD16"  
    Case pcibaselib.MDA12: s = "MDA12"  
    Case pcibaselib.MDA16: s = "MDA16"  
End Select
```

- **PORTCOUNT:** 2  
Gibt die Anzahl der digitalen Ports zurück.

## 4.2.4 Eigenschaften der PIO24II/PIO48II

Im Unterschied zur PIO24II besitzt die PIO48II sechs (anstatt drei) Digitalports mit je 8 Leitungen. Die digitalen Leitungen werden immer portweise auf Ein- bzw. Ausgang umgestellt, mit Ausnahme von Port 3 und Port 6 (PIO48II), die in 4er-Gruppen in der Richtung geändert werden.

- **CARDID:** 1..4

Die PIO24II/PIO48II Karten sind in der Reihenfolge durchnummeriert, wie sie installiert wurden oder wie sie im Gerätemanager (Windows® 2000/XP: *Start / (Einstellungen) / Systemsteuerung / System*) im Eintrag "BMC Messsysteme" aufgelistet sind. Unter **CardId** wird die entsprechende Messkarte selektiert.

Dieses Beispiel wählt nacheinander die erste und die zweite PIO24II aus.

```
PIO24II1.CardId = 1
PIO24II1.CardId = 2
```

- **DIR\_PORT1, DIR\_PORT2, DIR\_PORT3L, DIR\_PORT3H  
DIR\_PORT4, DIR\_PORT5, DIR\_PORT6L, DIR\_PORT6H :** 0..1

Die PIO24II/PIO48II besitzt 24/48 Digitalkanäle, die portweise in 8er- bzw. 4er-Gruppen zwischen Ein-/ und Ausgang umgeschaltet werden können. Dafür existieren vier/acht Eigenschaften (s. Tabelle), die die Werte **0-DirIn** bzw. **1-DirOut** annehmen können:

Folgende Tabelle legt fest, welche Leitungen der PIO24II/PIO48II mit den entsprechenden Richtungsvariablen eingestellt werden können:

Variable	Leitungen
<b>DirPort1</b>	1..8
<b>DirPort2</b>	9..16
<b>DirPort3L</b>	17..20
<b>DirPort3H</b>	21..24
<b>DirPort4</b>	25..32 (PIO48II)
<b>DirPort5</b>	33..40 (PIO48II)
<b>DirPort6L</b>	41..44 (PIO48II)
<b>DirPort6H</b>	45..48 (PIO48II)

Die Befehle zum Setzen und Abfragen der Digitalein- bzw. -ausgänge werden auf Seite 45 beschrieben.



**Wird ein Port auf Ausgang gestellt, sind dessen Leitungen immer *low*.**

---

- **INSTALLED CARDS:** 1..4  
Zeigt an, wie viele PIO-Karten gegenwärtig im Rechner eingebaut sind.  
Ändert sich dies, wird der Wert entsprechend angepasst.

## 4.3 Funktionen

### 4.3.1 Funktionen der meM-/USB-Geräte

Die analogen Geräte der meM-/USB-Serie (meM-AD/-ADDA/-ADf/-ADfo, USB-AD) stellen 16 analoge Eingänge, einen Analogausgang (meM-ADDA/-ADf/-ADfo, USB-AD) und zwei Digitalports (meM-ADDA/-ADf/-ADfo, USB-AD) mit je 4 Bit (meM-ADfo: 8 Bit) zur Verfügung bei einem festen Eingangsbereich von  $\pm 5V$ . Die Richtung der Digitalkanäle ist nicht umstellbar. Die Digitaleingänge von USB-AD/meM-ADDA/-ADf/-ADfo liegen mit Leitung 1..4 (meM-ADfo: 1..8) auf Port 1, die Digitalausgänge mit Leitung 1..4 (meM-ADfo: 1..8) auf Port 2.

Die digitalen USB-Messsysteme USB-PIO/meM-PIO besitzen 3 umschaltbare 8-Bit Digitalports.

Spezielle Zählfunktionen (s. "Zählfunktionen (meM-INC)", S. 37) werden für das Inkrementalgebermesssystem meM-INC zur Verfügung gestellt.

#### 4.3.1.1 Allgemeine Funktionen

- **UPDATECACHE:**

Liest, je nach meM-Gerät, alle analogen und/oder digitalen Leitungen in einen internen Puffer ein. Danach kann über die Geräte spezifischen **Cached...**-Eigenschaften sofort auf diese Werte zugegriffen werden. Diese Vorgehensweise ist besonders schnell.

Dieses Beispiel zeigt die aktuellen Zählerwerte der drei angeschlossenen Inkrementalgeber eines meM-INC in einer Listbox an.

```
meMINC1.UpdateCache
for i = 1 to 3
    List1.AddItem(meMINC1.CachedCounter(i), 0)
next
```

### 4.3.1.2 Analoge Funktionen

- **ANALOGIN(Cha):** (meM-AD/-ADDA/-ADf/-ADfo, USB-AD)  
Liest den analogen Wert am Kanal '**Cha**' ('**Cha**'=1..16) ein. Die Werte liegen im Eingangsspannungsbereich von  $\pm 5V$ .
- **ANALOGOUT(Cha):** (meM-ADDA/-ADf/-ADfo, USB-AD)  
Setzt den analogen Ausgang '**Cha**' auf einen analogen Wert oder liefert den momentan gesetzten analogen Wert zurück. Da die Geräte meM-ADDA/meM-ADf/meM-ADfo bzw. USB-AD genau einen analogen Ausgang haben, ist als Wert für '**Cha**' nur '**1**' zulässig. Die zulässigen Werte liegen im Ausgangsspannungsbereich von  $\pm 5V$ .

Dieses Beispiel liest den analogen Ausgangswert eines USB-AD/meM-ADDA Gerätes, zieht 1V ab und setzt den analogen Ausgang neu.

```
i = meMADDA1.AnalogOut(1)
i = i - 1.0
meMADDA1.AnalogOut(1) = i
```

- **CACHEDANALOGIN(Cha):** (meM-AD/-ADDA/-ADf/-ADfo, USB-AD)  
Liest den analogen Wert vom Kanal '**Cha**' ('**Cha**'=1..16) aus dem internen OCX-Cache. Die Werte liegen im Eingangsspannungsbereich von  $\pm 5V$ . Um die aktuellen Werte zu erhalten, sollte vorher **UpdateCache** aufgerufen werden.

### 4.3.1.3 Digitale Funktionen

- **CACHEDPORT(Num):** (meM-ADDA/-ADf/-ADfo/-PIO, USB-AD/-PIO)  
Wie **Port**, jedoch werden die Werte der digitalen Leitungen des Ports aus dem internen OCX-Cache gelesen. Um die aktuellen Werte zu erhalten, sollte vorher **UpdateCache** aufgerufen werden.

Dieses Beispiel liest den Port 2 einer USB-PIO/meM-PIO aus dem OCX-Cache ein und schreibt ihn unverändert wieder zurück.

```
meMPIO1.DirPort(2) = DirOut
meMPIO1.UpdateCache
i = meMPIO1.CachedPort(2)
meMPIO1.Port(2) = i
```

- **GETCACHEDLINE(Port, Line):**

(meM-ADDA/-ADf/-ADfo/-PIO, USB-AD/-PIO)

Wie **GetLine**, jedoch wird der Wert der digitalen Leitung 'Line' (1..4, USB-PIO/ meM-PIO/meM-ADfo: 1..8) im Port '**Port**' (1..2, USB-PIO/meM-PIO: 1..3) aus dem internen OCX-Cache gelesen.

Um die aktuellen Werte zu erhalten, sollte vorher **UpdateCache** aufgerufen werden.

Dieses Beispiel liest alle Eingangswerte eines meM-ADfos mit der Geräteummer 2 aus dem internen OCX-Cache.

```
meMADfo1.CardId = 2
meMADfo1.UpdateCache
For i = 1 to 8
    DigitalValue.Caption(i) = meMADfo1.GetCachedLine(1, i)
Next i
```

- **GETLINE(Port, Line):** (meM-ADDA/-ADf/-ADfo/-PIO, USB-AD/-PIO)

Fragt den Zustand einer einzelnen digitalen Leitung '**Line**' im Port '**Port**' ab. Die Funktion erwartet die Nummer des Ports als Argument (1..2, USB-PIO/meM-PIO: 1..3) und die Nummer der Leitung (1..4, USB-PIO/meM-PIO/meM-ADfo: 1..8) in diesem Port. Der Rückgabewert ist 0, wenn die Leitung nicht gesetzt ist, und 1, wenn die Leitung gesetzt ist. Die Funktion gibt auch bei Ausgängen immer den aktuellen Zustand der Leitung zurück (d. h. ein Ausgang, der auf 1 gesetzt wurde, aber durch Fehlbeschaltung auf Masse liegt, liefert 0). Die Funktion ist auch als **GetCachedLine** verfügbar.

Dieses Beispiel zeigt alle Eingangswerte eines meM-ADfs mit der Geräteummer 2.

```
meMADf1.CardId = 2
For i = 1 to 4
    DigitalValue.Caption(i) = meMADf1.GetLine(1, i)
Next i
```

- **PORT(Port):** (meM-ADDA/-ADf/-ADfo/-PIO, USB-AD/-PIO)  
Liest den digitalen Port 1 (USB-PIO/meM-PIO: 1..3) im Ganzen ein oder schreibt einen Wert an den Port 2 (USB-PIO/meM-PIO: 1..3). Bei USB-PIO/meM-PIO muss zum Einlesen der Werte der Port zuvor auf Eingang, zum Schreiben auf Ausgang gesetzt werden. Die Funktion erwartet als Argument die Nummer des Ports (Digitaleingänge: Port 1; Digitalausgänge: Port 2; USB-PIO/meM-PIO: 1..3). Die Funktion gibt auch bei Ausgängen immer den aktuellen Zustand der Leitung zurück (d. h. eine Portleitung, die auf **1** gesetzt wurde, aber durch Fehlbeschaltung auf Masse liegt, liefert **0**). Die Funktion ist auch als **CachedPort** verfügbar.

Dieses Beispiel liest den Port 1 von USB-AD/meM-ADDA ein und überträgt die Werte auf Port 2.

```
i = meMADDA1.Port(1)
meMADDA1.Port(2) = i
```

- **RESETLINE(Port, Line):** (meM-ADDA/-ADf/-ADfo/-PIO, USB-AD/-PIO)  
Setzt eine einzelne digitale Leitung '**Line**' im Port '**Port**' unabhängig von ihrem bisherigen Zustand auf *low*. Die Funktion erwartet die Nummer des Ports (Digitalausgänge: Port 2, USB-PIO/meM-PIO: Port 1..3) und der Leitung (1..4, USB-PIO/meM-PIO/meM-ADfo: 1..8) als Argument. Zuvor muss bei USB-PIO/meM-PIO der zugehörige Port auf Ausgang gestellt worden sein (s. "Eigenschaften der meM-/USB-Geräte", S. 24).

Dieses Beispiel setzt Leitung 1 und 3 eines USB-AD/meM-ADDA zurück ohne die anderen Leitungen des Ports zu ändern.

```
meMPIO1.ResetLine 2, 1
meMPIO1.ResetLine 2, 3
```

- **SETLINE(Port, Line):** (meM-ADDA/-ADf/-ADfo/-PIO, USB-AD/-PIO)  
Setzt eine einzelne digitale Leitung '**Line**' im Port '**Port**' auf *high*. Die Funktion erwartet die Nummer des Ports (Digitalausgänge: Port, USB-PIO/meM-PIO: Port 1..3) als Argument und die Nummer der Leitung (1..4, USB-PIO/meM-PIO/meM-ADfo: 1..8) in diesem Port. Zuvor muss bei USB-PIO/meM-PIO der zugehörige Port auf Ausgang gestellt worden sein (s. "Eigenschaften der meM-/USB-Geräte", S. 24).

Dieses Beispiel setzt Leitung 1 und 3 von Port 1 einer USB-PIO/meM-PIO ohne die anderen Leitungen des Ports zu ändern. Dazu wird die Richtung von Port 1 auf Ausgang gestellt.

```
meMPIO1.DirPort1 = DirOut
meMPIO1.SetLine 1, 1
meMPIO1.SetLine 1, 3
```

#### 4.3.1.4 Zählerfunktionen (meM-INC)

- **CACHEDCOUNTER(index):** (meM-INC)

Wie **COUNTER**, jedoch wird der Wert des Inkrementalgebers '**index**' aus dem inneren OCX-Cache gelesen. Um die aktuellen Werte zu erhalten, sollte vorher **UpdateCache** aufgerufen werden.

Dieses Beispiel liest alle Zählerwerte eines meM-INC mit der Gerätenummer 2 aus dem internen OCX-Cache.

```
meMINC1.CardId = 2
meMINC1.UpdateCache
For i = 1 to 3
    CounterValue.Caption(i) = meMINC1.CachedCounter(i)
next
```

- **COUNTER(index):** (meM-INC)

Liest den aktuellen Zählerstand des Inkrementalgebers '**index**' (1..3) ein oder weist ihm einen Wert zu.

Dieses Beispiel addiert die Werte der Inkrementalgeber 1 und 2 und weist das Ergebnis dem Zähler des Inkrementalgebers 3 zu.

```
i = meMINC.Counter(1)
s = meMINC.Counter(2)
meMINC1.Counter(3) = i+s
```

- **RESETCOUNTERS:** (meM-INC)

Setzt alle Zähler unabhängig von ihrem bisherigen Wert auf '0'.

Dieses Beispiel setzt alle Zähler eines meM-INC zurück.

meMINC1.ResetCounters

- **RESETMODE(index):** (meM-INC)

Liest den aktuellen Zählerresetmodus des Inkrementalgebers '**index**' (1..3) ein oder weist diesem eine Einstellung zu. Der externe Zählerreset kann die Einstellungen '**ResetHi**', '**ResetLo**' und '**ResetDisabled**' annehmen.

Dieses Beispiel liest den aktuellen Zählermodus ein und wechselt dann auf die andere Einstellung (**Hi**<->**Lo**). Ist der Reset ausgeschaltet, bleibt er in diesem Zustand.

```
For i = 1 to 3
  if meMINC1.ResetMode(i) <> ResetDisabled then
    if meMINC1.ResetMode(i) = ResetHi then
      meMINC1.ResetMode(i) = ResetLo
    else
      meMINC1.ResetMode(i) = ResetHi
    end if
  end if
next
```

## 4.3.2 Funktionen der PC16/PC20/P1000

### 4.3.2.1 Analoge Funktionen

- **ANALOGIN(Cha):**

Liest den analogen Wert am Kanal '**Cha**' ('**Cha**'=1..16) ein. Die Werte liegen im Eingangsspannungsbereich von  $\pm 1V$ ,  $\pm 2V$ ,  $\pm 5V$ ,  $\pm 10V$  (PC20/P1000). Dieser wird mit **AnalogInRange** eingestellt. Bei der PC16 liegen die Analogeingänge im Messbereich von  $\pm 10V$ .

Die Werte der ersten vier analogen Eingänge sollen in einem festgelegten Intervall periodisch erfasst und in Textfeldern angezeigt werden. Dazu wird das Textfeld (**Label1.Caption**) entsprechend oft auf die Form kopiert und mit einem Index (0-3) versehen. Mit Hilfe eines Timers, den man auf dem Fenster einfügt, legt man das Abfrageintervall fest.

```
Private Sub Timer1_Timer()
    For i = 1 To 4
        Label1(i-1).Caption = PC201.AnalogIn(i)
    Next i
End Sub
```

- **ANALOGOUT(Cha):** (PC20/P1000)

Setzt den analogen Ausgang '**Cha**' ('**Cha**'=1..2) auf einen analogen Wert oder liefert den momentan gesetzten analogen Wert des Kanals '**Cha**' zurück. Die zulässigen Werte liegen im Ausgangsspannungsbereich von  $\pm 5V$  bzw.  $\pm 10V$ , entsprechend den Jumpereinstellungen auf der Karte.

Dieses Beispiel liest den analogen Ausgangswert einer PC20, zieht 1V ab und setzt den analogen Ausgang neu.

```
i = PC201.AnalogOut(1)
i = i - 1.0
PC201.AnalogOut(1) = i
```

### 4.3.2.2 Digitale Funktionen

- **PIN(Line):**

Setzt eine einzelne digitale Leitung '**Line**' auf *high* oder fragt den Zustand der Leitung ab. Die Funktion erwartet die Nummer der Leitung (1..32 für PC20/P1000 bzw. 1..8 und 17..24 für PC16).

Dieses Beispiel setzt Leitung 15 auf *high*, fragt den Wert von Leitung 17 ab und weist ihn der Variablen A zu. Leitung 20 wird zudem getoggelt (ständiger Wechsel zwischen *low* und *high*). Zuvor wurde Port 1H auf Ausgang, Port 2L auf Eingang geschaltet.

```
PC201Dir_Pin09to16 = pc20lib.Output
PC201Dir_Pin17to24 = pc20lib.Input
PC201.Pin(15) = 1
A = PC201.Pin(17)
PC201.Pin(20) = 1 - PC201.Pin(20)
```

## 4.3.3 Funktionen der PCI-BASE50/300/1000

### 4.3.3.1 Allgemeine Funktionen

- MODULECHANNELS(index):** 0..16  
 Liefert die Anzahl der Kanäle des gewählten Moduls '**index**'. Dabei wird nicht zwischen analogen Ein- oder Ausgängen unterschieden. Es werden nur die Anzahl der Kanäle des Moduls zurückgeliefert.
- UPDATECACHE(range):**  
 Liest alle analogen Leitungen in den ActiveX internen Puffer ein. Danach kann über die analogen **Cached...**-Eigenschaften sofort auf diese Werte zugegriffen werden. Der Parameter '**range**' bestimmt in welchem Messbereich gemessen werden soll. Diese Vorgehensweise ist besonders schnell.

Dieses Beispiel zeigt die analogen Werte an allen 16 Eingängen einer PCI-BASE50/300/1000 mit einem MAD-Modul in einer Listbox an.

```
PciBasel.UpdateCache(pcibaselib.Range_10V)
for i = 0 to 15
    List1.AddItem(PciBasel.CachedAnalogIn(i))
next
```

### 4.3.3.2 Analoge Funktionen

Die PCI-BASE50/300/1000 kann mit bis zu zwei Modulen der Serie MAD/MDA bestückt werden und verfügt damit über 16 (single-ended) oder 8 (differenziell) analoge Eingänge pro MAD-Modul bzw. bis zu 2 analoge Ausgänge pro MDA-Modul (MDA12-4: 4 Analogausgänge). Die Eingangs- und Ausgangsspannungsbereiche sind Modul abhängig und dem MAD bzw. MDA-Datenblatt zu entnehmen.

Bitte beachten Sie, dass die Programmierung der PCI-BASE50/300/1000 kombiniert mit einem MCAN-Modul über dieses ActiveX Control nicht möglich ist.

- **ANALOGIN(Cha, range):** (PCI-BASE50/300/1000 mit MAD-Modul)  
Liest den analogen Wert des Kanals 'Cha' ('Cha'=1..16) ein. Die Werte liegen im Eingangsspannungsbereich von 'range'.
- **ANALOGOUT(Cha, range):** (PCI-BASE50/300/1000 mit MDA-Modul)  
Setzt den analogen Ausgang 'Cha' auf einen analogen Wert oder liefert den momentan gesetzten analogen Wert zurück. Die zulässigen Werte liegen im Ausgangsspannungsbereich von 'range'.

Dieses Beispiel liest den analogen Ausgangswert einer PCI-BASE50/300/1000, zieht 1V ab und setzt den analogen Ausgang neu.

```
i = PciBase1.AnalogOut(1)
i = i - 1.0
PciBase1.AnalogOut(1) = i
```

- **CACHEDANALOGIN(Cha):** (PCI-BASE50/300/1000 mit MAD-Modul)  
Liest den analogen Wert vom Kanal 'Cha' ('Cha'=1..16) aus dem internen OCX-Cache. Die Werte liegen im Eingangsbereich von **UpdateCache('range')**. Um die aktuellen Werte zu erhalten, sollte vorher **UpdateCache** aufgerufen werden.

### 4.3.3.3 Digitale Funktionen

- **GETLINE(Port, Line):**  
Fragt den Zustand einer einzelnen digitalen Leitung 'Line' im Port 'Port' ab. Die Funktion erwartet die Nummer des Ports als Argument (1..2) und die Nummer der Leitung (1..16) in diesem Port. Der Rückgabewert ist **0**, wenn die Leitung nicht gesetzt ist, und **1**, wenn die Leitung gesetzt ist. Die Funktion gibt auch bei Ausgängen (Port 2) immer den aktuellen Zustand der Leitung zurück (d.h. ein Ausgang, der auf **1** gesetzt wurde, aber durch Fehlbeschaltung auf Masse liegt, liefert **0**).

Dieses Beispiel zeigt alle Eingangswerte des Eingangsports (Port 1) einer PCI-BASE50/300/1000.

```
For i = 1 to 16
    DigitalValue.Caption(i) = PciBasel.GetLine(1, i)
Next i
```

- **PORT(Port):**

Liest den digitalen Port '**Port**' im Ganzen ein oder schreibt einen Wert an den Port '**Port**'. Die Funktion erwartet als Argument die Nummer des Ports (1..2). Wenn die Funktion gelesen wird, so gibt sie auch bei Ausgangsports immer den aktuellen Zustand der Leitung zurück (d. h. eine Portleitung, die auf **1** gesetzt wurde, aber durch Fehlbeschaltung auf Masse liegt, liefert **0**).

Folgende Übersicht gibt wieder, welche Leitungen in den einzelnen Ports zu finden sind.

Portnummer	PCI-BASE50/300/1000
1	1..16 (Eingang)
2	1..16 (Ausgang)

Dieses Beispiel liest den Port 1 einer PCI-BASE50/300/1000 ein und überträgt die Werte auf Port 2.

```
i = PciBasel.Port(1)
PciBasel.Port(2) = i
```

- **RESETLINE(Port, Line):**

Setzt eine einzelne digitale Leitung '**Line**' im Port '**Port**' unabhängig von ihrem bisherigen Zustand auf *low*. Die Funktion erwartet die Nummer des Ports (2) und der Leitung als Argument (1..16). Diese Funktion ist nicht für Port 1 geeignet, da dieser Port fest auf Eingang verdrahtet ist.

Dieses Beispiel löscht Leitung 1 und 3 einer PCI-BASE50/300/1000 ohne die anderen Leitungen des Ports zu ändern.

```
PciBasel.ResetLine 1, 1
PciBasel.ResetLine 1, 3
```

- **SETLINE(Port, Line):**

Setzt eine einzelne digitale Leitung '**Line**' im Port '**Port**' auf *high*. Die Funktion erwartet die Nummer des Ports als Argument (2) und die Nummer

der Leitung (1..16) in diesem Port. Diese Funktion ist nicht für Port 1 geeignet, da dieser fest auf Eingang verdrahtet ist.

Dieses Beispiel setzt Leitung 1 und 3 von Port 2 einer PCI-BASE50/300/1000 ohne die anderen Leitungen des Ports zu ändern.

```
PciBase1.SetLine 2, 1  
PciBase1.SetLine 2, 3
```

- **TOGGLELINE(Port, Line):**

Toggeln (Umschalten) der Leitung '**Line**' in Port '**Port**'. Unabhängig vom aktuellen Zustand der Leitung '**Line**' wird diese mit dieser Funktion umgeschaltet. War die Leitung z. B. vor Aufruf von **ToggleLine** *low*, so ist sie nach dem Aufruf nun *high*. Diese Funktion ist nicht für Port 1 geeignet, da dieser Port fest auf Eingang verdrahtet ist.

## 4.4 Funktionen der PIO24II/PIO48II

- **GETLINE(Port, Line):**

Fragt den Zustand einer einzelnen digitalen Leitung '**Line**' im Port '**Port**' ab. Die Funktion erwartet die Nummer des Ports als Argument (PIO24II: 1..3 / PIO48II: 1..6) und die Nummer der Leitung (1..8) in diesem Port. Der Rückgabewert ist **0**, wenn die Leitung nicht gesetzt ist, und **1**, wenn die Leitung gesetzt ist. Die Funktion gibt auch bei Ausgängen immer den aktuellen Zustand der Leitung zurück (d. h. ein Ausgang, der auf **1** gesetzt wurde, aber durch Fehlbeschaltung auf Masse liegt, liefert **0**).

Dieses Beispiel zeigt alle Eingangswerte des ersten Ports einer PIO24II mit der Gerätenummer 2.

```
PIO24III1.CardId = 2
For i = 1 to 8
    DigitalValue.Caption(i) = PIO24III1.GetLine(1, i)
Next i
```

- **PORT(Port):**

Liest den digitalen Port '**Port**' im Ganzen ein oder schreibt einen Wert an den Port '**Port**'. Zum Einlesen der Werte muss dieser zuvor auf Eingang, zum Schreiben auf Ausgang gesetzt werden. Die Funktion erwartet als Argument die Nummer des Ports (PIO24II: 1..3 / PIO48II: 1..6). Die Funktion gibt auch bei Ausgängen immer den aktuellen Zustand der Leitung zurück (d. h. eine Portleitung, die auf **1** gesetzt wurde, aber durch Fehlbeschaltung auf Masse liegt, liefert **0**).

Folgende Übersicht gibt wieder, welche Leitungen in den einzelnen Ports zu finden sind.

Portnummer	PIO24II/PIO48II
1	1..8 (Ein-/ Ausgang)
2	9..16 (Ein-/ Ausgang)
3	17..24 (Ein-/ Ausgang)
4	25..32 (Ein-/ Ausgang, nur PIO48II)
5	33..40 (Ein-/ Ausgang, nur PIO48II)
6	41..48 (Ein-/ Ausgang, nur PIO48II)

Dieses Beispiel liest den Port 1 einer PIO48II ein und überträgt die Werte auf Port 2.

```
i = PIO48III1.Port(1)
PIO48III1.Port(2) = i
```

- **RESETLINE(Port, Line):**

Setzt eine einzelne digitale Leitung '**Line**' im Port '**Port**' unabhängig von ihrem bisherigen Zustand auf *low*. Die Funktion erwartet die Nummer des Ports (PIO24II: 1..3 / PIO48II: 1..6) und der Leitung als Argument (1..8). Zuvor muss der zugehörige Port auf Ausgang gestellt worden sein (s. "Eigenschaften der PIO24II/PIO48II", S. 31).

Dieses Beispiel setzt Leitung 1 und 3 einer PIO48II zurück ohne die anderen Leitungen des Ports zu ändern. Der Port wurde vorher auf Ausgang gestellt.

```
PIO48III1.ResetLine 1, 1
PIO48III1.ResetLine 1, 3
```

- **SETLINE(Port, Line):**

Setzt eine einzelne digitale Leitung '**Line**' im Port '**Port**' auf *high*. Die Funktion erwartet die Nummer des Ports als Argument (PIO24II: 1..3 / PIO48II: 1..6) und die Nummer der Leitung (1..8) in diesem Port. Zuvor muss der zugehörige Port auf Ausgang gestellt worden sein (s. "Eigenschaften der PIO24II/PIO48II", S. 31).

Dieses Beispiel setzt Leitung 1 und 3 einer PIO48II ohne die anderen Leitungen des Ports zu ändern. Dazu wird die Richtung von Port 1 auf Ausgang gestellt.

```
PIO48III1.DirPort1 = DirOut
PIO48III1.SetLine 1, 1
PIO48III1.SetLine 1, 3
```

## 4.5 Programmierbeispiele

Die Programmierbeispiele wurden mit Visual Basic® erstellt und getestet. Weitere Anwendungen befinden sich auf der "Software Collection"-CD. Wählen Sie dazu den Eintrag "Beispielprogramme", der sich im selben Verzeichnis befindet wie das jeweilige ActiveX Control **STR-meM**, **STR-PC**, **STR-PCI**, **STR-PIO**.

### 4.5.1 Programmierbeispiele für meM-/USB-Geräte

Folgendes Beispiel setzt alle Ports der ersten USB-PIO/meM-PIO auf Eingang und zeigt die Anzahl der insgesamt angeschlossenen USB-Geräte dieses Typs an.

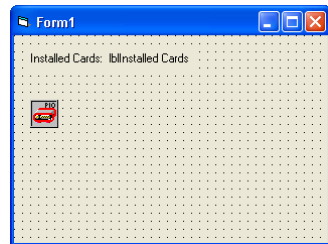


Abbildung 13

```

USB-PIO      Option Explicit
meM-PIO

Private Sub Form_Load()
Dim i As Integer
    'Aktiviere USB-PIO's OnError Event
meMPIO1.DisableErrors = True
    'Arbeite mit der ersten am USB-Bus gefundenen USB-PIO
For i = 1 To meMPIO1.LastAttached
    meMPIO1.CardId = i
    If meMPIO1.Attached Then Exit For
Next
With meMPIO1
    'Stelle alle Ports auf Eingang
    If .Attached Then
        .DirPort1 = DirIn
        .DirPort2 = DirIn
        .DirPort3H = DirIn
        .DirPort3L = DirIn
    End If
End With
'Zeige an, wieviele USB-PIOs insgesamt angeschlossen sind
lblInstalledCards = meMPIO1.InstalledCards
End Sub
    
```

Mit dem folgenden Programm wird der erste Zähler des ersten meM-INCs, das am USB-Bus gefunden wurde, bedient. Es lässt sich damit der aktuelle Wert des Zählers abrufen, der Zähler zurücksetzen und die Resetfunktion aktivieren.

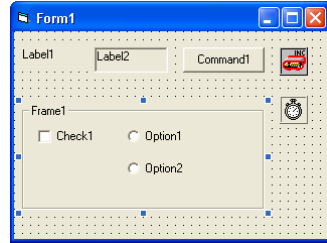


Abbildung 14

meM-INC

Option Explicit

```
Private Sub Form_Load()
    memINC1.DisableErrors = True

    'initialize graphic objects
    Label1.Caption = "Counter 1:"
    Label2.BorderStyle = 1
    Label2.Alignment = 1
    Label2.Caption = "0"
    Command1.Caption = "Reset to '0'"
    Frame1.Caption = "Reset-type"
    Check1.Caption = "Disable"
    Check1.Value = 1
    Option1.Caption = "Active HI"
    Option1.Value = True
    Option1.Enabled = False
    Option2.Caption = "Active LO"
    Option2.Enabled = False

    'adjust update timer
    Timer1.Enabled = False
    Timer1.Interval = 50

    'start update
    Start_Update
End Sub
'enable/disable hardware reset
Private Sub Check1_Click()
    If Check1.Value = 1 Then
        Option1.Enabled = False
        Option2.Enabled = False
        memINC1.ResetMode = ResetDisabled
    Else
        Option1.Enabled = True
        Option2.Enabled = True
        If Option1.Value = True Then
            memINC1.ResetMode = ResetHigh
        End If
    End If
End Sub
```

```

        Else
            memINC1.ResetMode = ResetLow
        End If
    End If
End Sub

'reset counter1 to zero
Private Sub Command1_Click()
    memINC1.Counter(1) = 0
    Label2.Caption = memINC1.Counter(1)
End Sub

'find the first meM-INC at the USB-Bus and start the
'update timer
Private Sub Start_Update()
Dim i As Integer
    'loop to the highest device number
    For i = 0 To memINC1.LastAttached
        'select a device
        memINC1.CardId = i
        'is this device attached?
        If memINC1.Attached Then
            Timer1.Enabled = True
            Exit Sub
        End If
    Next
End Sub

'set reset type to 'Active HI'
Private Sub Option1_Click()
    memINC1.ResetMode = ResetHigh
End Sub

'set reset type to 'Active LO'
Private Sub Option2_Click()
    memINC1.ResetMode = ResetLow
End Sub

'read out the current value from counter1 and display it
'in label2
Private Sub Timer1_Timer()
    memINC1.UpdateCache
    Label2 = memINC1.CachedCounter(1)
End Sub

```

## 4.5.2 Programmierbeispiele für PC16/PC20/P1000

Dieses Beispiel zeigt die Anzahl der installierten P1000 Karten an und startet einen Timer, der den Analogwert an Kanal 1 und die 16 Digitalkanäle von Port 1 anzeigt.

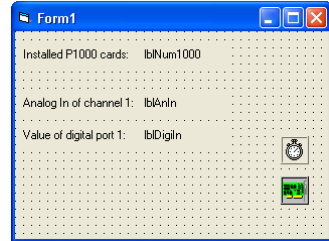


Abbildung 15

```

P1000      Option Explicit

Private Sub Form_Load()
    lblNum1000 = P10001.InstalledCards

    If P10001.InstalledCards > 0 Then
        P10001.CardId = 1
        P10001.Dir_Pin01to08 = pc20lib.Input
        tmrUpdate.Enabled = True
    End If
End Sub

Private Sub tmrUpdate_Timer()
    Dim i As Integer

    'first clear label
    lblDigiIn = ""

    'read analog value of channel 1
    lblAnIn = P10001.AnalogIn(1)

    'make a binary string of the digital value at port 1
    For i = 1 To 16
        lblDigiIn = lblDigiIn & IIf(P10001.Pin(i), "1", "0")
    Next
End Sub
    
```

## 4.5.3 Programmierbeispiele für PCI-BASE50/300/1000

Dieses Beispiel zeigt den ersten analogen Eingang und den ersten digitalen Eingangsport zyklisch an (**tmrUpdate\_Timer**). Beim Start des Programms werden einige Informationen über die eingebaute PCI-BASE50/300/1000 angezeigt (**Form\_Load**).

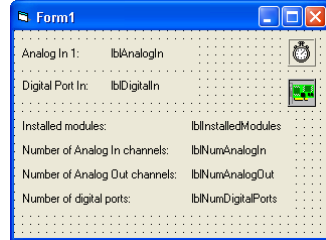


Abbildung 16

```

PCI-BASE50      Option Explicit
PCI-BASE300
PCI-BASE1000   Const DigiPortIn = 1
                Const DigiPortOut = 2

                Private Sub Form_Load()
                    With PciBase1
                        lblInstalledModules = .ModuleCount
                        lblNumAnalogIn = .AnalogInCount
                        lblNumAnalogOut = .AnalogOutCount
                        lblNumDigitalPorts = .PortCount
                    End With
                    tmrUpdate.Enabled = True
                End Sub

                Private Sub tmrUpdate_Timer()
                    lblAnalogIn = PciBase1.AnalogIn(1, Range_10V)
                    lblDigitalIn = Hex(PciBase1.Port(DigiPortIn))
                End Sub
    
```

## 4.5.4 Programmierbeispiele für PIO24II/PIO48II

Dieses Beispiel für die PIO24II/PIO48II stellt den Port 1 auf Eingang und startet dann einen Timer. Der Timer wertet die Optionsknöpfe aus. Ist der Port als Eingang geschaltet, wird der Wert am Port 1 binär dargestellt. Wird auf Ausgang geschaltet, wird der Timer ausgeschaltet.

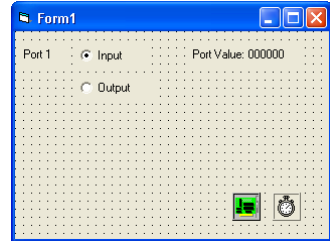


Abbildung 17

```

PIO24II Option Explicit
PIO48II

Private Sub Form_Load()
    'set port 1 to input
    PIO24II1.DirPort1 = pioilib.DirIn

    'start timer
    tmrUpdate.Enabled = True
End Sub

Private Sub Option1_Click(Index As Integer)
    Select Case Index
        Case 0: tmrUpdate.Enabled = True
        Case 1:
            tmrUpdate.Enabled = False
            lblDigiValue = "---"
    End Select
End Sub

Private Sub tmrUpdate_Timer()
    Dim i As Integer

    'make a binary string
    For i = 1 To 8
        lblDigiValue = lblDigiValue & _
            IIf(PIO24II1.GetLine(1, i), "1", "0")
    Next
End Sub

```

# 5 Index

## A

About 24  
ActiveX Control 5, 8, 9, 10, 12, 14, 17  
ActiveX importieren 17  
AnalogIn() 34, 39, 42  
AnalogInCount 29  
AnalogInRange 27, 29, 39  
Analogkanäle  
    Ausgang lesen 34, 39, 42  
    Ausgang setzen 34, 39, 42  
    Eingang lesen 34, 39, 42  
AnalogOut() 34, 39, 42  
AnalogOutCount 29  
Anzahl  
    Analogausgänge 29  
    Analogeingänge 29  
    angeschlossene Geräte 26  
    Digitalports 30  
    installierte Geräte 26  
    installierte Karten 28, 30, 32  
    installierte Module 30  
    Kanäle des Moduls 41  
Attached 24

## B

BaseId 29  
Bibliothek 15, 23, 30

## C

C++® 22  
CachedAnalogIn() 34, 42  
CachedCounter() 37  
CachedPort() 34  
CardId 24, 27, 29, 31  
Counter() 37

## D

Delphi® 9, 10, 17

## Digitalkanäle

Leitung lesen 35, 40, 42, 45  
Leitung setzen 36, 40, 43, 46  
Leitung zurücksetzen 36, 43, 46  
Richtung umschalten 25, 27, 31

## Digitalport

lesen 34, 36, 43, 45  
schreiben 36, 43, 45

DirIn 25, 31

DirOut 25, 31

DirPin 'x' to 'y' 27

DirPort 25, 31

DisableErrors 25

Diskettenversion 6, 11

## E

Eigenschaften 5, 16, 23, 24

meM-Geräte 24

PC16/PC20/P1000 27

PCI-BASE50/300/1000 29

PIO24II/PIO48II 31

USB-Geräte 24

Einbindung in Programmiersprachen 9

C++® 22

Delphi® 17

Visual Basic® 14

Visual C++™ 20

EnableGate 26

## F

Fehlerreport 25

Frequenzmessung 26

Funktionen 16, 23, 33, 46

meM-Geräte 33

PC16/PC20/P1000 39

PCI-BASE50/300/1000 41

PIO24II/PIO48II 45

USB-Geräte 33

## G

Gate 26

Gerät auswählen 24

GetCachedLine() 35

GetLine() 35, 42, 45

## I

Index 24  
Input 27  
Installation 5, 9, 10  
Installationsverzeichnis 12  
InstalledCards 26, 28, 30, 32  
Internetversion 6, 11

## K

Karte auswählen 27, 29, 31  
Kartentyp 29  
Klasse 15  
Komponenten 14, 17

## L

LastAttached 26  
Left 24  
Leitung  
  lesen 42  
  setzen 43  
  umschalten 44  
  zurücksetzen 43

## M

MAD-Module 29  
MDA-Module 29  
meM-Befehle  
  AnalogIn() 34  
  AnalogOut() 34  
  Attached 24  
  CachedAnalogIn() 34  
  CachedCounter() 37  
  CachedPort() 34  
  CardId 24  
  Counter() 37  
  DirPort 25  
  DisableErrors 25  
  EnableGate 26  
  Gate 26  
  GetCachedLine() 35  
  GetLine() 35  
  InstalledCards 26  
  LastAttached 26

  Port() 36  
  ResetCounters 37  
  ResetLine() 36  
  ResetMode() 38  
  Serial 26  
  SetLine() 36  
  UpdateCache 33

## meM-Geräte

  Anschluss prüfen 24  
  memx.ocx 13  
  MEMXLib 15, 18, 23  
  Messbereich 27, 29, 39, 41  
  ModuleChannels() 41  
  ModuleCount 30  
  ModuleId 30  
  Modultyp 30

## N

Name 24

## O

Objektkatalog 15, 23  
OCX-Cache 33, 34, 35, 41, 42  
Output 27

## P

p1000.ocx 13  
PC16/PC20/P1000-Befehle  
  AnalogIn() 39  
  AnalogInRange 27  
  AnalogOut() 39  
  CardId 27  
  DirPin'x'to'y' 27  
  InstalledCards 28  
  Pin() 40  
PC20Lib 15, 18, 23, 28  
pci300x.ocx 13  
PCI-BASE50/300/1000-Befehle  
  AnalogIn() 42  
  AnalogInCount 29  
  AnalogInRange 29  
  AnalogOut() 42  
  AnalogOutCount 29  
  BaseId 29

CachedAnalogIn() 42  
    CardId 29  
    GetLine() 42  
    InstalledCards 30  
    ModuleChannels() 41  
    ModuleCount 30  
    ModuleId 30  
    Port() 43  
    PortCount 30  
    ResetLine() 43  
    SetLine() 43  
    ToggleLine() 44  
    UpdateCache 41  
PCIBaseLib 15, 18, 23, 30  
Pin() 40  
PIO24II/PIO48II-Befehle  
    CardId 31  
    DirPort 31  
    GetLine() 45  
    InstalledCards 32  
    Port() 45  
    ResetLine() 46  
    SetLine() 46  
pioiii.ocx 13  
PIOIIILib 15, 18, 23  
Port() 36, 43, 45  
PortCount 30  
Programmierbeispiele 47  
    meM-Geräte 47  
    PCI16/PC20/P1000 50  
    PCI-BASE50/300/1000 51  
    PIO24II/PIO48II 52  
    USB-Geräte 47  
Programmierung 5, 9, 23  
    Beispiele 47

## R

ResetCounters 37  
ResetLine() 36, 43, 46  
ResetMode() 38

## S

Serial 26  
Seriennummer 26  
SetLine() 36, 43, 46

Spannungsbereich 33, 39, 42  
Speicher aktualisieren 33, 41  
Standardverzeichnis 12  
Systemvoraussetzungen 10

## T

Tag 24  
ToggleLine() 44  
Top 24  
Torzeit einstellen 26

## U

Update 6, 10  
UpdateCache 33, 41  
Urheberrechte 8  
USB-Befehle  
    AnalogIn() 34  
    AnalogOut() 34  
    Attached 24  
    CachedAnalogIn() 34  
    CachedPort() 34  
    CardId 24  
    DirPort 25  
    DisableErrors 25  
    GetCachedLine() 35  
    GetLine() 35  
    InstalledCards 26  
    LastAttached 26  
    Port() 36  
    ResetLine() 36  
    Serial 26  
    SetLine() 36  
    UpdateCache 33

## V

Versionsnummer 5  
Visual Basic® 9, 10, 14  
Visual C++™ 10, 20

## Z

Zähler  
    lesen 37  
    schreiben 37

zurücksetzen 37  
Zählermodus 38  
Zählerreset 37, 38  
Zählfunktionen 33, 37