

LIBADX

Programmierschnittstelle (ActiveX Control) für
bmcm Messgerätetreiber

**Installations- und
Programmierhandbuch**

Version 4.5

► www.bmcm.de

bavarian measurement company munich

Inhaltsverzeichnis

1 Überblick	7
1.1 Einleitung	7
1.2 BMC Messsysteme GmbH	8
1.3 Urheberrechte	9
1.4 Schnelleinstieg	10
2 Installation und Anbindung	11
2.1 Allgemeines	11
2.2 LibadX Installation	12
2.3 Einbindung in Programmiersprachen	15
2.3.1 Anbindung an Visual Basic® 4.0 - 6.0	15
2.3.2 Anbindung an Delphi® 3.01 - 5.0	17
2.3.3 Anbindung an Visual C++® 5.0/6.0	19
2.3.4 Anbindung an Visual C#®.NET	20
2.3.5 Anbindung an VB.NET (Microsoft®)	21
2.4 Beispielprogramme	23
3 Grundlagen	25
3.1 Allgemeines	25
3.2 Verbindung zum Messsystem	26
3.2.1 Kanalnummern und Messbereiche	26
3.2.2 iM-AD25a / iM-AD25 / iM3250T / iM3250	27
3.2.3 LAN-AD16f	28
3.2.4 PCIe-BASE / PCI-BASEII/300/1000 / PCI-PIO	29
3.2.4.1 Digitalports und Zähler	29
3.2.4.2 MAD12/12a/12b/12f/16/16a/16b/16f	30
3.2.4.3 MADDA16/16n	30
3.2.4.4 MDA12/12-4/16/16-2i/16-4i/16-8i	31
3.2.5 meM-AD /-ADDA /-ADf / -ADfo	32
3.2.6 meM-PIO / meM-PIO-OEM	33
3.2.7 USB-AD	34

3.2.8	USB-AD12f	36
3.2.9	USB-AD16f	37
3.2.10	USB-PIO / USB-PIO-OEM	38
4	Schnittstellen und Funktionen	40
4.1	Die Schnittstelle LibadX	40
4.1.1	Überblick	40
4.1.2	Open	41
4.1.3	Close	42
4.1.4	GetVersion	42
4.1.5	LastError	43
4.1.6	LastErrorString	43
4.1.7	ScanPrepare	44
4.1.8	ScanAnalogIn	45
4.1.9	ScanDigitalIn	45
4.1.10	Scan	46
4.1.11	ScanSave	47
4.1.12	FileOpen	47
4.1.13	FileCreatePrepare	48
4.1.14	FileCreateAnalogIn	49
4.1.15	FileCreateDigital	49
4.1.16	FileCreate	50
4.1.17	AnalogIn	51
4.1.18	AnalogOut	51
4.1.19	DigitalIn	52
4.1.20	DigitalOut	53
4.1.21	DigitalInLine	53
4.1.22	DigitalOutLine	54
4.1.23	DigitalDirection	54
4.1.24	Sample	55
4.1.25	AboutBox	56
4.2	Die Schnittstelle INvxFile	56
4.2.1	Überblick	56
4.2.2	Open	57
4.2.3	Create	57
4.2.4	Close	58
4.2.5	SignalCount	58

4.2.6	Signal	59
4.3	Die Schnittstelle INvxSignal	59
4.3.1	Überblick	59
4.3.2	Name	61
4.3.3	GroupName	61
4.3.4	Comment	62
4.3.5	xStart	62
4.3.6	xEnd	63
4.3.7	xDelta	63
4.3.8	xUnit	64
4.3.9	xSetUsing	64
4.3.10	xGetUsing	65
4.3.11	yMin	66
4.3.12	yMax	67
4.3.13	yDefaultMin	67
4.3.14	yDefaultMax	68
4.3.15	yDelta	68
4.3.16	yUnit	69
4.3.17	ySetUsing	69
4.3.18	yGetUsing	70
4.3.19	ScanStart	71
4.3.20	SampleCount	72
4.3.21	ScaleX	72
4.3.22	ScaleY	73
4.3.23	ResetDataPosition	73
4.3.24	GetNextScaled	74
4.3.25	GetNextScaledDigital	74
4.3.26	Unscale	75
4.3.27	NextSample	75
4.3.28	NextDigitalSample	76
4.3.29	GetSampleAt	76
4.3.30	GetSampleAtOffset	77
4.3.31	IsAnalog	77
4.3.32	IsDigital	78

5 Index

79

1 Überblick

1.1 Einleitung

LibadX ist eine einheitliche Programmierschnittstelle zu allen Messsystemen von BMC Messsysteme GmbH. Diese Schnittstelle ist für alle Programmiersprachen unter Windows® 7/XP erreichbar, in welche ActiveX Komponenten geladen werden können (z. B. C++®, Visual C++®, Visual C#®, Visual Basic®, VB .NET, Delphi®).



- **LibadX ist eine 32-Bit Schnittstelle. Für 64-Bit Systeme muss die Applikation als 32-Bit Applikation erstellt werden.**
- **Bitte beachten Sie, dass alle Beispielpcodes in diesem Handbuch aus Gründen der Einfachheit bewusst auf eine Fehlerbehandlung verzichten. Selbstverständlich muss diese in selbst geschriebenen Programmen realisiert werden.**
- **Die Anbindung eines ActiveX Controls erfolgt durch die verwendete Entwicklungsumgebung. Nachdem jede Entwicklungsumgebung diese Anbindung unterschiedlich realisiert, kann dieses Handbuch nur einen Überblick über die Verwendung in den verschiedenen Entwicklungsumgebungen verschaffen. Bitte konsultieren Sie die Dokumentation Ihrer Entwicklungsumgebung, um Informationen über das genaue Vorgehen zum Einbinden von ActiveX Komponenten zu erhalten.**

Im Normalfall importiert die Entwicklungsumgebung eine ActiveX Komponente und erzeugt den Sourcecode für eine Hilfsklasse, mit der die Funktionen der Komponente aufgerufen werden. Diese Hilfsklasse legt letztendlich die genaue Aufrufkonvention der Funktionen fest.

Die in diesem Handbuch beschriebenen Funktionen können als je nach Entwicklungsumgebung unter anderen Namen oder mit leicht geänderten Parametern zur Verfügung stehen. Aus diesem Grund sollte die Dokumentation der

jeweiligen Entwicklungsumgebung konsultiert werden, um Informationen über die entsprechenden Konventionen beim Import von ActiveX Komponenten zu erhalten.

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>.

► www.bmcm.de
bavarian measurement company munich

1.3 Urheberrechte

Die Programmierschnittstelle **LibadX** mit allen Erweiterungen wurde mit größtmöglicher Sorgfalt erstellt und geprüft. Die BMC Messsysteme GmbH gibt keine Garantien, weder in Bezug auf dieses Handbuch noch in Bezug auf die in diesem Buch beschriebene Hard- und Software, ihre Qualität, Durchführbarkeit oder Verwendbarkeit für einen bestimmten Zweck. Die BMC Messsysteme GmbH haftet in keinem Fall für direkt oder indirekt verursachte oder erfolgte Schäden, die entweder aus unsachgemäßer Bedienung oder aus irgendwelchen Fehlern am System resultieren. Änderungen, die dem technischen Fortschritt dienen, bleiben uns vorbehalten.

Die Programmierschnittstelle **LibadX** 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 © 2011

Stand: 26. Oktober 2011

BMC Messsysteme GmbH

Hauptstraße 21
82216 Maisach
DEUTSCHLAND

Tel.: +49 8141/404180-1

Fax: +49 8141/404180-9

E-Mail: info@bmcm.de

1.4 Schnelleinstieg



Führen Sie die Geräteinstallation, wie in Ihrer Dokumentation beschrieben, vor Installation der LibadX durch und überprüfen Sie anschließend im Windows® Geräte-Manager die erfolgreiche Installation.

- Die erfolgreiche Installation der Messhardware wird im Geräte-Manager überprüft, der die aktuelle Rechnerkonfiguration anzeigt:
 - **Windows® 7:** Start / Systemsteuerung / System und Sicherheit / System / Geräte-Manager
 - **Windows® XP:** Start / Systemsteuerung / System / TAB "Hardware" / Schaltfläche "Geräte-Manager"
- Bei erfolgreicher Installation (Messsysteme angeschlossen und ggf. angeschaltet!) befindet sich im Eintrag "Messdatenerfassung (BMC Messsysteme)" die neu installierte Hardware. Durch einen Doppelklick auf diese werden deren Eigenschaften und eventuelle Gerätekonflikte angezeigt.
- Wird das Gerät vom PC fehlerlos erkannt, installieren Sie **LibadX** mit Hilfe der beigelegten "bmcm" Software Collection"-CD. Wechseln Sie dazu auf die Produktseite der von Ihnen verwendeten bmcm Hardware (Produkte / <Produktname>) und wählen Sie im Bereich "API (Programmierung)" den Eintrag "STR-LIBADX" zur Programmierung unter Windows®.
- Die Installation kann direkt geöffnet und ausgeführt werden. Lässt dies Ihr Browser nicht zu, speichern Sie zuerst das Installationsprogramm auf die Festplatte und starten Sie anschließend dieses durch Öffnen der Datei **libad-actx.exe**.
- Es müssen nur Angaben zum Verzeichnispfad gemacht werden, bevor der verfügbare Speicherplatz ermittelt wird und der Kopiervorgang startet. Die entsprechende ActiveX Komponente wird ins Windows® Systemverzeichnis kopiert.
- Das **LibadX** ActiveX Control steht nach der Installation zur Verwendung in eigenen Programmen zur Verfügung. Je nach Entwicklungsumgebung ist eine unterschiedliche Anbindung nötig (s. "Einbindung in Programmiersprachen", S. 15).

2 Installation und Anbindung

2.1 Allgemeines



Die Installation des Gerätetreibers muss unbedingt vor Installation von LibadX erfolgt sein!

Die Installation erfolgt mit Hilfe der bmcm"Software Collection"-CD, die Sie zur Installation bitte ins CD-ROM Laufwerk geben.

Die Programmierschnittstelle **LibadX** ist als *ActiveX Control* realisiert, das vom Installationsprogramm im System registriert wird. Die Registrierung im System allein genügt allerdings nicht, um die Funktionen aus **LibadX** in den meisten Entwicklungsumgebungen verwenden zu können. Die folgenden Kapitel geben einen Überblick über die nötige Anbindung für einige ausgewählte Entwicklungsumgebungen. Genaue Informationen zur Einbindung eines ActiveX Controls entnehmen Sie bitte der Dokumentation Ihrer Entwicklungsumgebung.

2.2 LibadX Installation

Beim Einlegen der "Software Collection"-CD wird bei aktivierter AutoPlay-Funktion des CD-ROM automatisch ein CD-Starter geöffnet (andernfalls **openhtml.exe** starten).

Öffnen Sie zuerst die Produktübersicht der bmc Hardware durch Auswahl des Eintrags "Produkte" und klicken Sie auf das von Ihnen verwendete Messsystem. Zur Programmierung unter Windows® 7/XP wählen Sie im Bereich "API (Programmierung)" die Produktbezeichnung "STR-LIBADX" um die Installation zu starten.

www.bmc.de | Englisch

HOME

► USB
Datenblätter, Treiber, ActiveX Controls und Software für die USB-Messsysteme USB-AD/AD12f/AD16f und meM-AD/ADDA/ADf/ADfo, die digitalen USB-Geräte USB-PIO und meM-PIO/INC, sowie den Datenlogger meM-LOG(-SE).

► NEXTVIEW
Wählen Sie dies Sie auch Installieren Sie NextView®.

► PRODUKT
Datenblätter, Treiber, Programmierschnittstelle GmbH.

USB-AD USB-AD12f USB-AD16f AMS42-USB AMS84-USB

meM-AD meM-ADDA meM-ADf meM-ADfo meM-LOG(-SE)

► API (PROGRAMMIERUNG)
Die Programmierung des Messsystems erfolgt mit Hilfe der LIBAD4. Die LIBAD4 ist eine Schnittstelle für die Programmiersprache C/C++ zur Ansteuerung aller analogen und digitalen Messsysteme der BMC Messsysteme GmbH. Bitte beachten Sie die Installationshinweise aus dem LIBAD4 Handbuch. Unter Windows® 7/XP steht außerdem das ActiveX Control LIBADX zur Verfügung. Mit diesem ActiveX Control lassen sich neben C/C++ alle anderen Programmiersprachen verwenden, die eine Einbindung von ActiveX Controls zulassen (z. B. Visual Basic oder Delphi).

Dokumentation				
Produkt	Bezeichnung	Version	Größe	
UM-LIBAD4	Bibliothek für Programmierschnittstelle LIBAD4	4.5	661.3KB	
IG-LIBADX	Programmierhandbuch für LIBADX (ActiveX Control)	4.5	713.1KB	

Programmierung unter Windows® 7/XP

Produkt	Bezeichnung	Version	Größe	
SDK-LIBAD	Programmierschnittstelle LIBAD4 API für Windows®	4.5.469	1.9MB	
STR-LIBADX	Programmierschnittstelle (ActiveX Control) für bmc Messgerätetreiber	4.5.469	1.9MB	
STR-LIBADX-EX	Beispielprogramme für Programmierschnittstelle LibadX	4.5.469	374.2KB	

Abbildung 1

Bei Verwendung des CD-Starters im HTML-Format, wird zusätzlich gefragt, ob das Installationsprogramm direkt ausgeführt oder ob es gespeichert werden soll. Möglich sind beide Optionen.

Bei manchen Browsern ist die vorherige Speicherung der Installationsdatei auf die Festplatte jedoch zwingend erforderlich. In diesem Fall muss das Installationsprogramm **libad-actx.exe** nach dem Kopiervorgang explizit gestartet werden.

Ein Assistent führt in überschaubaren Dialogen durch die Installation. Dabei können die Angaben jederzeit geändert werden. Die Schaltfläche "Weiter" führt in den nächsten Dialog, mit "Zurück" gehen Sie einen Schritt zurück. Die Installation kann mit "Abbrechen" stets vorzeitig beendet werden.

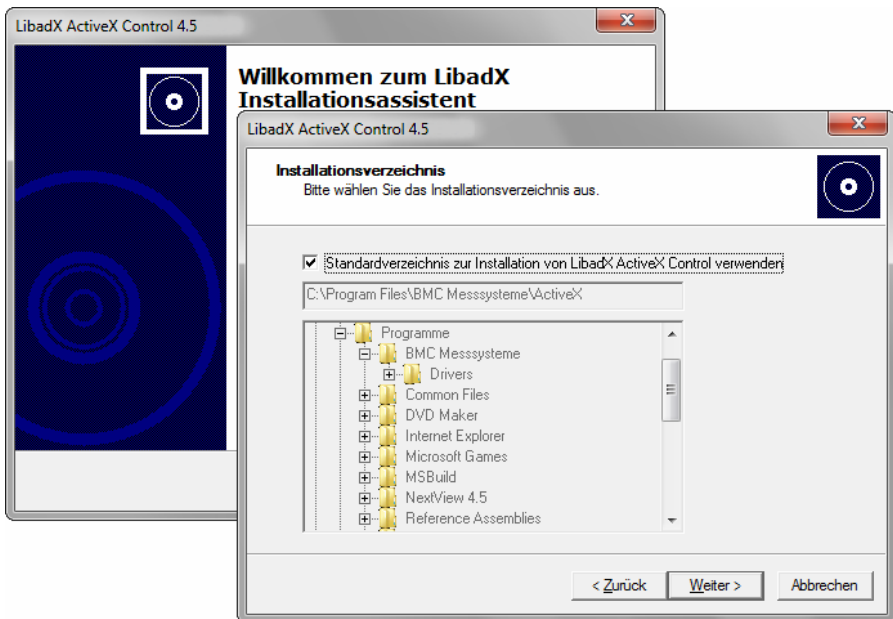


Abbildung 2

Das vorgeschlagene Standardverzeichnis lässt sich nach Belieben ändern. Zur Auswahl eines anderen Installationsverzeichnisses klickt man dazu das Häkchen in der Optionsbox weg, so dass die unteren Eingabefelder eingeblendet werden.

Nachdem alle Informationen gegeben sind, wird die Größe des zur Verfügung stehenden Speicherplatzes ermittelt und die zur Installation des **LibadX** ActiveX Controls benötigten Dateien auf Festplatte kopiert.

Starten Sie, falls erforderlich den Rechner neu, damit die Änderungen vom System erkannt werden.

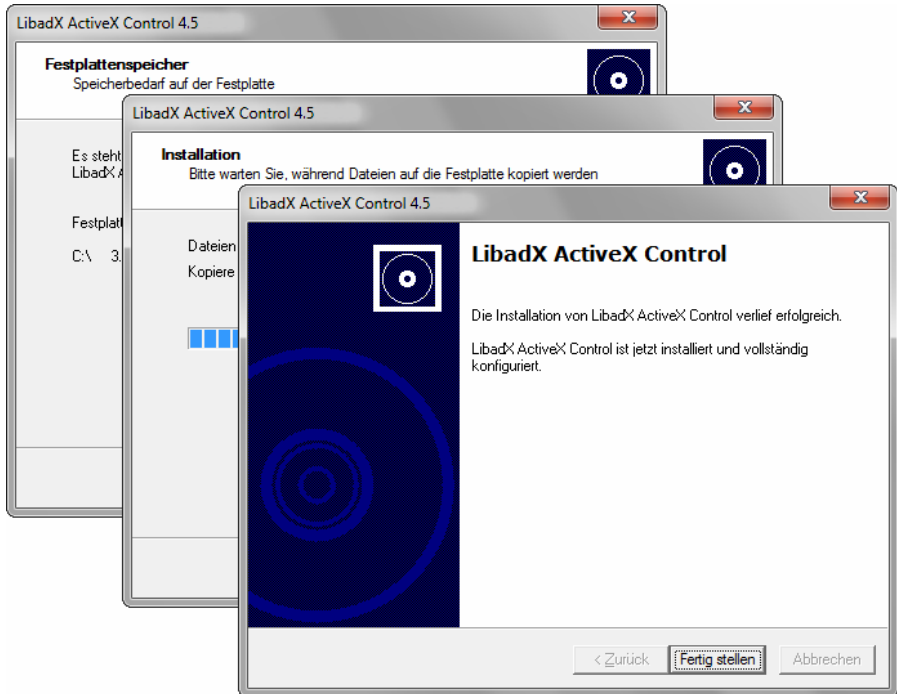


Abbildung 3

2.3 Einbindung in Programmiersprachen

2.3.1 Anbindung an Visual Basic® 4.0 - 6.0



Standard EXE

Starten Sie Visual Basic® und wählen Sie die Option "Standard EXE" aus dem Anfangsdialog (bzw. Menüpunkt "Datei / Neues Projekt").

LibadX wird wie jedes ActiveX Control in Visual Basic® über das Menü "Projekt" und den Eintrag "Komponenten" eingebunden. Im darauf folgenden Dialog "Komponenten" muss der Eintrag "LibadX Object Library 4.0" aktiviert werden.

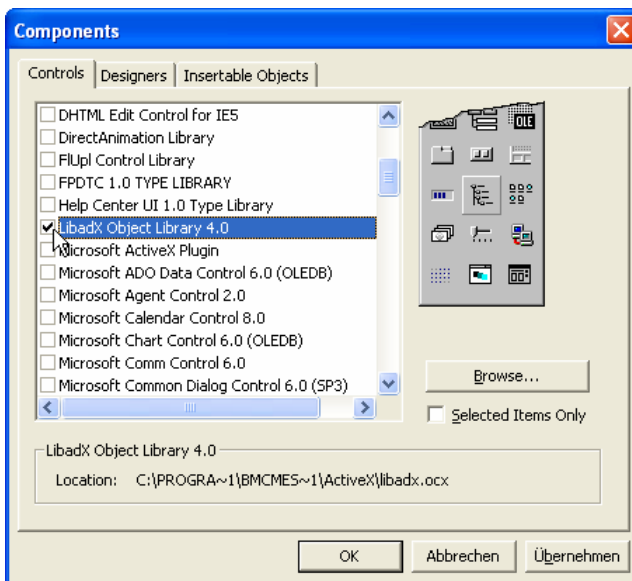


Abbildung 4

Danach steht das **LibadX** Icon in der Werkzeugleiste von Visual Basic® zur Verfügung und kann in eine Form eingefügt werden. Wie der Timer ist es während der Ausführung des Programms unsichtbar.

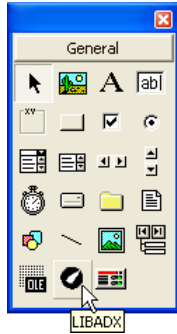


Abbildung 5

Klicken Sie wie gewohnt auf das Icon und ziehen Sie einen Rahmen in der Form des neu angelegten Projekts auf. Dieser Rahmen wird nach Einfügen des Objekts wieder auf die feste Größe des Icons reduziert.

Erstellen Sie nun die folgende Routine Form_Load() im Codefenster des Projekts:

```
VB      Private Sub Form_Load()
           LIBADIX1.AboutBox
       End Sub
```

Um die korrekte Installation der **LibadX** und ihre Erreichbarkeit in Visual Basic[®] zu überprüfen, empfiehlt es sich dieses Programm zu starten. Das Programm muss die Form ohne Fehlermeldung auf dem Bildschirm anzeigen.



- **Aus Kompatibilitätsgründen befindet sich das Icon der ehemaligen Programmierschnittstelle BMCSAD ebenfalls in der Toolbar. Als Benutzer der LibadX benötigen Sie weder dieses Icon noch den BMCSAD.**
- **Bitte beachten Sie, dass alle Beispielcodes hier aus Gründen der Einfachheit bewusst auf eine Fehlerbehandlung verzichten. Selbstverständlich muss diese in selbst geschriebenen Programmen realisiert werden.**
- **Weitere Beispielprogramme mit Sourcecode können von der LibadX Produktseite der "Software Collection"-CD installiert werden.**

2.3.2 Anbindung an Delphi® 3.01 - 5.0



Application

Starten Sie Delphi® und erzeugen Sie ggf. ein neues Projekt (Menüpunkt "Datei / Neue Anwendung").

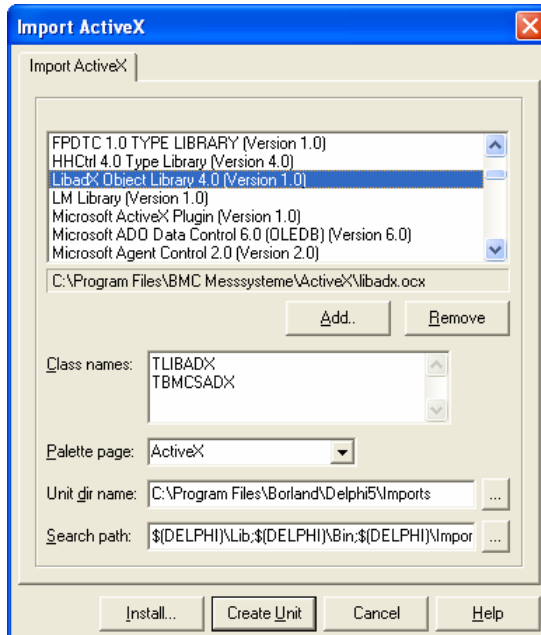


Abbildung 6

Wählen Sie nun im Menü "Komponente" den Eintrag "ActiveX importieren..." aus. Im darauf folgenden Dialog "ActiveX-Element importieren" markieren Sie den Eintrag "LibadX Object Library 4.0". Betätigt man die Taste "Installieren...", wird **LibadX** von Delphi® importiert und durch die nächsten Schritte in Delphi® als Komponente registriert.

Im automatisch folgenden Dialog "Installieren" wählen Sie das Package aus, in das die neue Komponente installiert werden soll. Bestätigen Sie diesen Dialog mit "OK", um das **LibadX** ActiveX Control in das Package "Delphi Benutzerkompo-

nenten" aufzunehmen. Delphi® wird daraufhin das Package neu übersetzen. Bestätigen Sie dazu die diesbezügliche Frage fortzufahren mit "Ja".

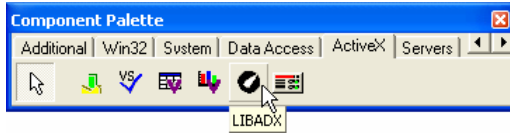


Abbildung 7

Danach steht das **LibadX** - Icon im Tab "ActiveX" der Werkzeugleiste von Delphi® zur Verfügung. Fügen Sie dieses Icon in die Form des neuen Projekts ein.

Erstellen Sie dann einen Event Handler für das **OnCreate()** Ereignis der Form und führen Sie den Event Handler folgendermaßen aus:

Delphi	<pre> procedure TForm1.FormCreate(Sender: TObject); begin LibadX.AboutBox (); end; </pre>
---------------	---

Um die korrekte Installation der **LibadX** und ihre Erreichbarkeit in Delphi® zu überprüfen, empfiehlt es sich dieses Programm zu starten. Das Programm muss die Form ohne Fehlermeldung auf dem Bildschirm anzeigen.



- **Aus Kompatibilitätsgründen befindet sich das Icon der ehemaligen Programmierschnittstelle BMCSAD ebenfalls in der Toolbar. Als Benutzer der LibadX benötigen Sie weder dieses Icon noch den BMCSAD.**
 - **Bitte beachten Sie, dass alle Beispielcodes hier aus Gründen der Einfachheit bewusst auf eine Fehlerbehandlung verzichten. Selbstverständlich muss diese in selbst geschriebenen Programmen realisiert werden.**
 - **Weitere Beispielprogramme mit Sourcecode können von der LibadX Produktseite der "Software Collection"-CD installiert werden.**
-
-

2.3.3 Anbindung an 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. Folgender Beispielcode demonstriert das Vorgehen:

```
C++
#include <windows.h>
#import "c:\LibadX\LibadX.ocx"

LIBADX::_DLibadXPtr libadx;

int
main (int argc, char **argv)
{
    HRESULT result = CoInitialize (NULL);
    if (FAILED (result))
        return FALSE;

    libadx.CreateInstance (__uuidof(LIBADX::LIBADX));
    libadx->AboutBox ();

    return 0;
}
```



- Nähere Hinweise zu **#import**, **__uuidof()** und den **Compiler Support Klassen** für COM entnehmen Sie bitte dem Artikel "Microsoft Visual C++[®] Compiler Native COM Support" von Microsoft[®], sowie der entsprechenden Compilerdokumentation von Microsoft[®].
- Bitte beachten Sie, dass alle Beispielcodes hier aus Gründen der Einfachheit bewusst auf eine Fehlerbehandlung verzichten. Selbstverständlich muss diese in selbst geschriebenen Programmen realisiert werden.
- Weitere Beispielprogramme mit Sourcecode können von der LibadX Produktseite der "Software Collection"-CD installiert werden.

2.3.4 Anbindung an Visual C#[®].NET



Der "managed code" eines .NET Programms enthält keine direkte Unterstützung für den Aufruf von ActiveX Controls. Aus diesem Grund muss vor Verwendung eines ActiveX Controls eine DLL erzeugt werden, die als "Brücke" zwischen "managed code" und ActiveX Control funktioniert. Dem Visual C#[®] Programm wird dann lediglich eine Referenz auf diese "Brücke" übergeben.

Obwohl Visual Studio[®] den automatischen Import von ActiveX Controls unterstützt, ist diese Vorgehensweise mit Einschränkungen verbunden (siehe MSDN Dokumentation). Deshalb sollte die entsprechende "Brücke" durch den Aufruf des Programms **tlbimp** aus dem .NET SDK erzeugt werden.

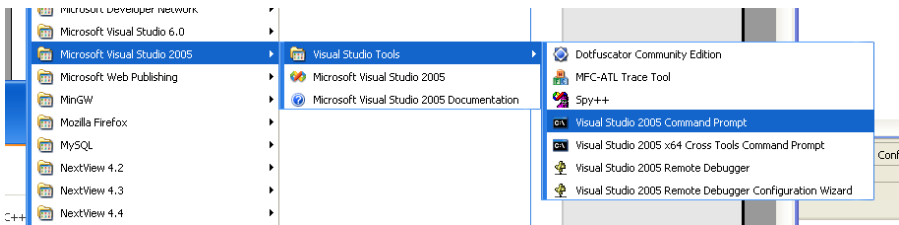


Abbildung 8

Starten Sie dazu ein "Microsoft Visual Studio Command Prompt" und geben Sie folgenden Befehl ein (dabei muss **libadx.ocx** durch den vollständigen Pfad zur ActiveX Komponente ersetzt werden).

```
tlbimp libadx.ocx /out:libadxTypeLib.dll /namespace:LIBADX
```

Dann kann jedem .NET Programm eine Referenz auf **libadxTypeLib.dll** hinzugefügt werden. Damit steht die Funktionalität des ActiveX Controls für Visual C#[®] zur Verfügung.

Ein Batchfile zum Erzeugen der "Brücke" und Übersetzen eines Visual C#[®] Programms findet sich in den Beispielpogrammen für das LIBADX ActiveX Control.

Die genauen Aufrufkonventionen der generierten "Brücken"-DLL lassen sich mit dem Object Browser des Visual Studios[®] einsehen.

2.3.5 Anbindung an VB.NET (Microsoft®)

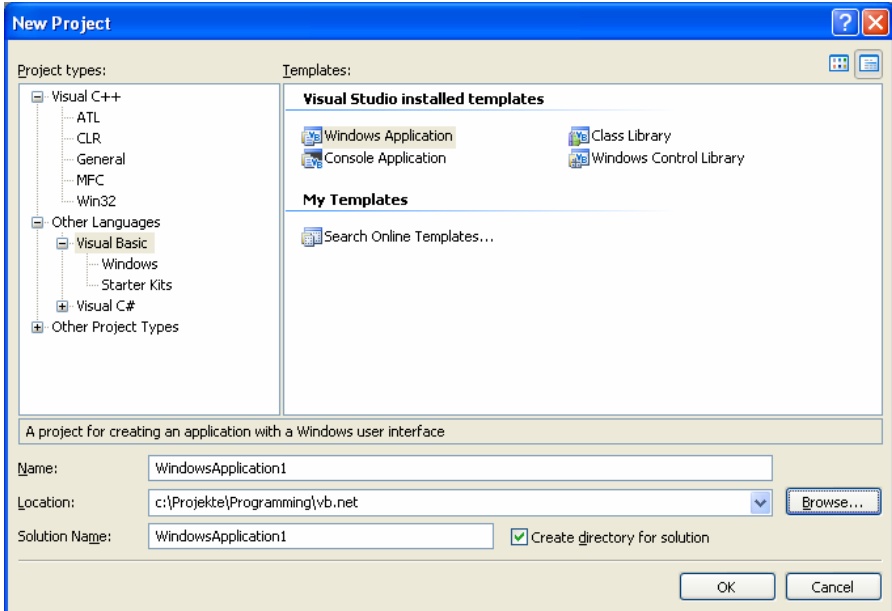


Abbildung 9

Starten Sie Visual Studio® und erzeugen Sie in Visual Basic® ein neues Projekt (z. B. über den Menüpunkt "Datei / Neues Projekt") als Windows® Applikation (s. Abbildung 9).

Öffnen Sie durch einen Rechtsklick auf die Toolbox das Kontextmenü und wählen Sie daraus den Befehl "Choose Items...".

Aktivieren Sie die COM-Komponente "LibadX Object Library 4.0", um das **LibadX** ActiveX Control in die Programmierumgebung einzubinden.

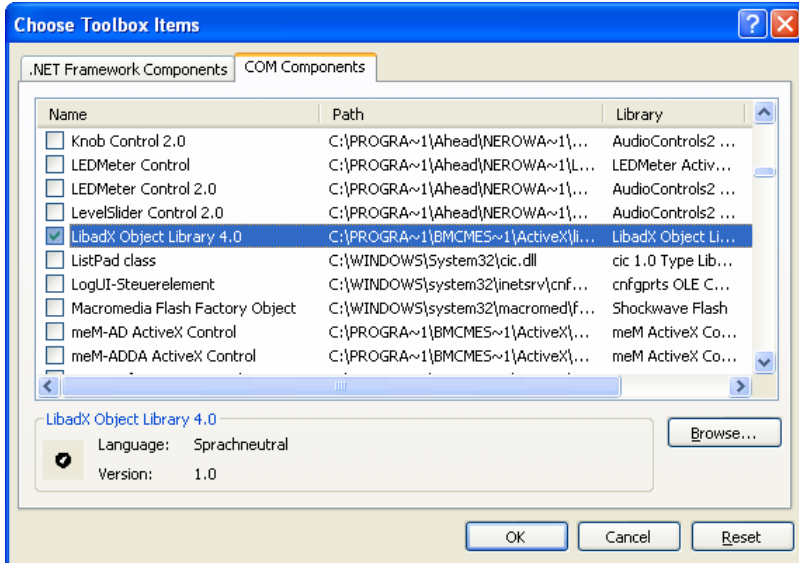


Abbildung 10

Danach steht das **LibadX** Icon in der Werkzeugleiste von Visual Basic® zur Verfügung und kann in eine Form eingefügt werden.

Klicken Sie wie gewohnt auf das Icon und ziehen Sie einen Rahmen in der Form des neu angelegten Projekts auf. Dieser Rahmen wird nach Einfügen des Objekts wieder auf die feste Größe des Icons reduziert.

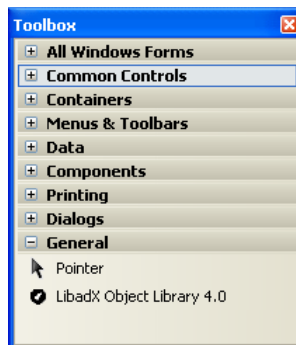


Abbildung 11

2.4 Beispielprogramme

Die "Software Collection"-CD enthält Beispielprogramme, die die Verwendung des **LibadX** ActiveX Controls demonstrieren. Diese können von der jeweiligen Produktseite des verwendeten Messsystems installiert werden.

Wählen Sie dazu den Eintrag "STR-LIBADX-EX" im Abschnitt "Programmierung unter Windows® 7/XP", der unter "API (Programmierung)" auf der Produktseite aufgelistet ist.

The screenshot shows the BMC Messsysteme website. At the top, there is a navigation bar with 'Homepage | English' and the BMC logo. Below the navigation bar, there are images of a hand holding a CD and a USB device. The main content area is divided into several sections:

- USB-AD12F**: Das USB-AD12F ist ein 4 digitale Ein-/Ausgang...
- Datenblätter**: A table with columns 'Produkt' and 'Datei'. It lists 'USB-AD12F' and 'USB-AD12F'.
- TREIBER**: Installation der Hardware, Installation der passenden Treiber, Anwendungsorte.
- Treiber für Windows**: A table with columns 'Produkt', 'Bezeichnung', 'Version', 'Größe', and 'Icon'. It lists 'BMC-M-DR-IG', 'BMC-M-DR', and 'STR-LIBADX'.
- NEXTVIEW**: NextView® 4 ist ein Softwarepaket für die Erfassung und Auswertung digitaler Signale.
- API (PROGRAMMIERUNG)**: Die Programmierung des Messsystems erfolgt mit Hilfe der LIBAD4. Die LIBAD4 ist eine Schnittstelle für die Programmiersprache C/C++ zur Ansteuerung aller analogen und digitalen Messsysteme der BMC Messsysteme GmbH. Bitte beachten Sie die Installationshinweise aus dem LIBAD4 Handbuch. Unter Windows® 7/XP steht außerdem das ActiveX Control LIBADX zur Verfügung. Mit diesem ActiveX Control lassen sich neben C/C++ alle anderen Programmiersprachen verwenden, die eine Einbindung von ActiveX Controls zulassen (z. B. Visual Basic oder Delphi).
- Dokumentation**: A table with columns 'Produkt', 'Bezeichnung', 'Version', 'Größe', and 'Icon'. It lists 'UM-LIBAD4', 'IG-LIBADX', and 'STR-LIBADX'.
- Programmierung unter Windows® 7/XP**: A table with columns 'Produkt', 'Bezeichnung', 'Version', 'Größe', and 'Icon'. It lists 'SDK-LIBAD', 'STR-LIBADX', and 'STR-LIBADX-EX'.

Abbildung 12

Die Beispielprogramme befinden sich im während der Installation ausgewählten Verzeichnis (z. B. "Programme \ BMC Messsysteme \ ActiveX \ LibadX Examples") jeweils nach Programmiersprache getrennt.

Programmiersprache	Ordner
Visual Basic®	vb
Delphi®	delphi
Visual C++®	vc5
Visual C#®	.net



Bitte beachten Sie, dass alle Beispielprogramme möglichst einfach ausgeführt worden sind, keine Fehlerbehandlung enthalten und aus diesem Grund keine vollständige Applikation darstellen.

3 Grundlagen

3.1 Allgemeines

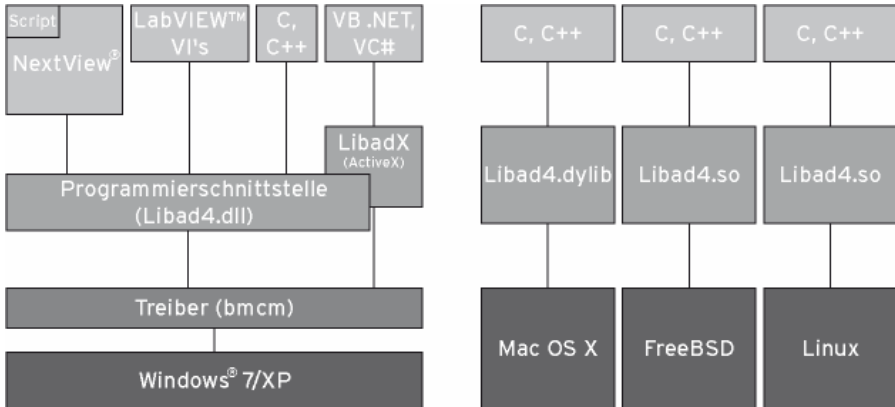


Abbildung 13

Das **LibadX** ActiveX Control ist die Programmierschnittstelle zur **LIBAD4** Bibliothek. Diese ist eine Schnittstelle zu allen Messsystemen der BMC Messsysteme GmbH. Sie erlaubt das Lesen und Schreiben von Einzelwerten, wie das Einlesen eines Analogeingangs oder das Ausgeben eines Werts an einen Analogausgang.

Neben der Ein-/Ausgabe von Einzelwerten kann mit der **LibadX** eine Messung durchgeführt werden. Das Scannen der Eingangskanäle findet im entsprechenden Treiber statt und ist aus diesem Grund zeitlich von der Applikation entkoppelt. Schnell und ohne jeglichen Verlust von Messwerten lassen sich damit die Eingangskanäle abtasten.

Zusätzlich kann man mit der LibadX Programmierschnittstelle auf Messdateien der Messdatenerfassungs- und Analysesoftware **NextView®4** zugreifen.

3.2 Verbindung zum Messsystem

Das **LibadX** ActiveX Control stellt zwei Funktionen zur Verfügung, mit denen eine Verbindung zu einem Messsystem geöffnet bzw. wieder geschlossen werden kann.

Mit der Funktion **Open()** wird ein Messsystem geöffnet, mit **Close()** wieder geschlossen. Folgendes Beispiel demonstriert das prinzipielle Vorgehen:

```
if (LIBADXL.Open ("usb-pio"))
    ...
    LIBADXL.Close
else
    MsgBox "Konnte USB-PIO Gerät nicht öffnen"
```

Der Funktion **Open()** wird der Name des Messsystems übergeben. Der übergebene String wird ohne Berücksichtigung von Groß- und Kleinschreibung verwendet, das heißt "usb-pio" und "USB-PIO" öffnen beide die USB-PIO / USB-PIO-OEM. Wurde eine Verbindung mit dem Messsystem geöffnet, gibt **Open()** den Wert **TRUE** zurück, beim Auftritt eines Fehlers **FALSE**.

Dabei ist es nicht möglich, mehrere Geräte mit einem Objekt gleichzeitig zu öffnen. Es ist aber durchaus möglich, mehrere (verschiedene) Messsysteme zu öffnen, wenn mehrere Objekte benutzt werden. Folgendes Beispiel öffnet eine PCIe-BASE / PCI-BASEII/300/1000 / PCI-PIO und eine USB-PIO / USB-PIO-OEM:

```
if (LIBADXL.Open ("pcibase")
    AND LIBADXL2.Open ("usb-pio"))
    ...
endif
```

3.2.1 Kanalnummern und Messbereiche

Ein- bzw. Ausgangskanäle werden in **LibadX** durch Kanalnummern spezifiziert. Die verwendeten Kanalnummern sind abhängig vom eingesetzten Messsystem und in den entsprechenden Kapiteln dokumentiert. Beispielsweise ist der erste Analogeingang eines USB-AD12f der Kanal 1.

Analoge Kanäle erwarten neben der Kanalnummer noch die Angabe eines Messbereichs (bzw. eines Ausgabebereichs), in dem gemessen (bzw. ausgegeben) werden soll. Dieser Messbereich ist wie die Kanalnummer vom Messsystem abhängig und in den folgenden Kapiteln dokumentiert.

3.2.2 iM-AD25a / iM-AD25 / iM3250T / iM3250

Um ein iM-AD25a, iM-AD25, iM3250T oder iM2350 mit **LibadX** zu öffnen, muss an **Open()** der String "**im:<ip-addr>**" übergeben werden. Dabei muss **<ip-addr>** durch die entsprechende IP-Adresse ersetzt werden. Beispielsweise öffnet der String "**im:192.168.1.1**" das iM-Gerät mit der IP Adresse 192.168.1.1. Beim Öffnen des Treibers wird nicht zwischen den iM-Gerätetypen unterschieden.

Messsystem	Analog	Kanalnummer	Messbereich	range	Digital
iM-AD25a	16 Eingänge	1..16	$\pm 10.24V$ $\pm 5.12V$	1 0	1: Ausgang (Bit 0..3)
iM-AD25	16 Eingänge	1..16	$\pm 5.12V$	0	1: Ausgang (Bit 0..3)
iM3250T	32 Eingänge	17..48	$\pm 5.12V$	0	-
iM3250	32 Eingänge	AnIn 1..16: 1..16 (bei 1 BPL) 17..32 (bei 2 BPL) AnIn 17..32: 33..48	$\pm 5.00V$	0	-



Bitte beachten Sie, dass sich beim iM3250T durch einen eventuell gesteckten MAL-Messverstärker der Messbereich des entsprechenden Kanals verändern kann.

3.2.3 LAN-AD16f

Um ein LAN-Messsystem vom Typ LAN-AD16f mit der **LibadX** zu öffnen, wird an `Open()` der String "`lanbase:<ip-addr>`" oder "`lanbase:@<sn>`" übergeben. Dabei muss `<ip-addr>` durch die entsprechende IP-Adresse ersetzt werden oder `<sn>` durch die Seriennummer des LAN-AD16f. Der String "`lanbase:192.168.1.1`" öffnet z. B. das LAN-Gerät mit der IP Adresse 192.168.1.1 und der String "`lanbase:@157`" öffnet das LAN-Gerät mit der Seriennummer 157.

Messsystem	Analog	Kanalnummer	range (Messber.)	range (Ausgabebereich)	Digital	Richtung
LAN-AD16f	16 Eingänge 2 Ausgänge	1..16 1 .. 2	0 ($\pm 1.024V$) 1 ($\pm 2.048V$) 2 ($\pm 5.12V$) 3 ($\pm 10.24V$)	0 ($\pm 10.24V$)	2 Ports (je 16 Bit)	1: Eingang (Bit 0..15) 2: Ausgang (Bit 0..15)

Die 16 Analogeingänge eines LAN-AD16f besitzen die Kanalnummern 1-16. Die beiden Analogausgänge haben die Kanalnummern 1 und 2.

Die Richtung der Digitalports ist nicht umschaltbar. Dabei stehen die 16 Leitungen des ersten Ports (DIO1, Kanalnummer: 1) auf Eingang, die 16 Leitungen des zweiten Ports (DIO2, Kanalnummer: 2) auf Ausgang.



Der Zähler des LAN-AD16f ist nur mit dem LIBAD4 SDK programmierbar.

3.2.4 PCIe-BASE / PCI-BASEII/300/1000 / PCI-PIO

Um eine PCIe-BASE, PCI-BASEII, PCI-BASE300, PCI-BASE1000 oder PCI-PIO mit **LibadX** zu öffnen, muss an **Open()** der String "**pcibase**" (oder "**pci300**") übergeben werden. Beim Öffnen des Treibers wird nicht zwischen verschiedenen Versionen der PCI(e)-Messkarte unterschieden.

Mehrere Karten lassen sich durch Angabe der Kartenummer unterscheiden (1. Karte mit "**pcibase:0**", 2. Karte mit "**pcibase:1**", usw.).

Eine Messkarte kann auch direkt unter Angabe ihrer Seriennummer geöffnet werden. Die Karte mit der Seriennummer 157 lässt sich zum Beispiel mit "**pcibase:@157**" ansprechen.

3.2.4.1 Digitalports und Zähler

Die PCIe-BASE / PCI-BASEII/300/1000 / PCI-PIO stellt zwei 16-Bit Digitalports zur Verfügung.

Bei der PCIe-BASE, PCI-BASEII und PCI-PIO ist die Richtung der Digitalleitungen in 8-er Gruppen umschaltbar. Nach dem Einschalten steht der erste Port auf Eingang, der zweite auf Ausgang.

Bei der PCI-BASE300/1000 hingegen sind die Ports in ihrer Richtung fest verdrahtet, der erste Port steht auf Eingang, der zweite Port auf Ausgang.

Zusätzlich sind einige Versionen mit einem (PCIe-BASE) bzw. drei (PCI-BASEII, PCI-PIO) 32-Bit Zählern ausgestattet.



Die Zähler der PCIe-BASE, PCI-BASEII und PCI-PIO sind nur mit dem LIBAD4 SDK programmierbar.

3.2.4.2 MAD12/12a/12b/12f/16/16a/16b/16f

Der erste analoge Eingangskanal eines MAD12/12a/12b/12f/16/16a/16b/16f beginnt bei 1. Befindet sich ein zweites analoges Eingangsmodul auf der PCI(e)-Multifunktionskarte (nicht: PCI-PIO), wird der erste Eingang des zweiten Moduls unter der Nummer 257 (0x100+1) angesprochen.

Selbstverständlich ist es möglich, ein Eingangsmodul differentiell (nicht MAD12b/16b) und das zweite single-ended zu betreiben, so dass dann 24 Eingangskanäle zur Verfügung stehen.

Die Messbereiche der Eingänge sind modulabhängig. Sind verschiedene Analogeingangsmodule auf die PCI(e)-Messkarte (nicht PCI-PIO) gesteckt, können sich die Messbereiche der Kanäle 1..16 von den Messbereichen der Kanäle 17..32 unterscheiden.

Modul	Analog	Kanalnummer	Messbereich	range
MAD12, MAD16	16 Eingänge (single-ended)	1..16 (se)	$\pm 1.024\text{V}$	0
			$\pm 2.048\text{V}$	1
	8 Eingänge (differentiell)	17..24 (diff)	$\pm 5.120\text{V}$	2
			$\pm 10.240\text{V}$	3
			0.06V..5.06V	4
MAD12a, MAD12f, MAD16a, MAD16f	16 Eingänge (single-ended)	1..16 (se)	$\pm 1.024\text{V}$	0
			$\pm 2.048\text{V}$	1
	8 Eingänge (differentiell)	17..24 (diff)	$\pm 5.120\text{V}$	2
			$\pm 10.240\text{V}$	3
MAD12b, MAD16b	16 Eingänge (single-ended)	1..16	$\pm 1.024\text{V}$	0
			$\pm 2.048\text{V}$	1
			$\pm 5.120\text{V}$	2
			$\pm 10.240\text{V}$	3

3.2.4.3 MADDA16/16n

Der erste analoge Eingangs- oder Ausgangskanal eines MADDA16/16n beginnt bei 1. Befindet sich ein zweites Analogmodul auf der PCI(e)-Multifunktionskarte (nicht: PCI-PIO), wird der erste Eingang bzw. Ausgang des zweiten Moduls unter der Nummer 257 (0x100+1) angesprochen.

Die Messbereiche der Eingänge sind modulabhängig. Sind verschiedene Analogeingangsmodule auf die PCI(e)-Messkarte (nicht PCI-PIO) gesteckt, können sich die Messbereiche der Kanäle 1..16 von den Messbereichen der Kanäle 17..32 unterscheiden.

Modul	Analog	Kanalnummer	range (Messbereich)	range (Ausgabebereich)
MADDA16, MADDA16n	16 Eingänge	1..16	0 ($\pm 1.024V$)	0
	2 Ausgänge	1..2	1 ($\pm 2.048V$)	1
			2 ($\pm 5.120V$)	2
			3 ($\pm 10.240V$)	3

3.2.4.4 MDA12/12-4/16/16-2i/16-4i/16-8i

Ebenso wie beim MAD12/12a/12b/12f/16/16a/16b/16f werden die Kanäle eines zweiten analogen Ausgangsmoduls ab der Nummer 257 (0x100+1) angesprochen.

Modul	Analog	Kanalnummer	Ausgabebereich	range
MDA12, MDA16	2 Ausgänge	1..2	$\pm 10.24V$	0
			$\pm 5.12V$	1
MDA12-4	4 Ausgänge	1..4	$\pm 10.24V$	0
			$\pm 5.12V$	1
MDA16-2i	2 Ausgänge	1..2	$\pm 10.24V$	0
MDA16-4i	4 Ausgänge	1..4	$\pm 10.24V$	0
MDA16-8i	8 Ausgänge	1..8	$\pm 10.24V$	0

Die Ausgabebereiche der Ausgangsmodule MDA12/MDA12-4 und MDA16 werden hardwaremäßig am Modul konfiguriert. Der Aufrufer muss sicherstellen, dass der übergebene Messbereich mit dem konfigurierten Messbereich des Moduls übereinstimmt.

3.2.5 meM-AD /-ADDA /-ADf / -ADfo

Um ein meM-AD/-ADDA/-ADf/-ADfo mit **LibadX** zu öffnen, muss an **Open()** der String **"memadusb"** (meM-AD), **"memaddausb"** (meM-ADDA), **"memadfusb"** (meM-ADf) bzw. **"memadfpusb"** (meM-ADfo) übergeben werden. Mehrere USB Messsysteme lassen sich durch Angabe der Gerätenummer öffnen (z. B. 1. Gerät mit **"memadusb:0"**, 2. Gerät mit **"memadusb:1"**, usw.). Die Reihenfolge der Geräte wird durch das Anstecken bestimmt.

Da USB Messsysteme im Betrieb an- und abgesteckt werden können, ist es möglich, dass die Gerätenummern nicht aufeinander folgend vergeben sind. Steckt man z. B. drei meM-ADDA an und entfernt dann das 2. Gerät, sind die verbleibenden meM-ADDA mit **"memaddausb:0"** und **"memaddausb:2"** anzusprechen.

Um unabhängig von dieser Ansteckreihenfolge zu sein, kann ein Gerät auch mit einer bestimmten Seriennummer geöffnet werden. Das Gerät mit der Seriennummer 157 lässt sich zum Beispiel mit **"memadfpusb:@157"** ansprechen.

Messsystem	Analog	Kanalnummer	Eing./Ausg.-bereich	range	Digital	Kanalnummer
meM-AD	16 Eingänge	1..16	±5.12V	0	-	-
meM-ADDA, meM-ADf	16 Eingänge 1 Ausgang	1..16 1	±5.12V	0	2 Ports (je 4 Bit)	1: Eingang (Bit 0..3) 2: Ausgang (Bit 0..3)
meM-ADfo	16 Eingänge 1 Ausgang	1..16 1	±5.12V	0	2 Ports (je 8 Bit)	1: Eingang (Bit 0..7) 2: Ausgang (Bit 0..7)

Die 16 Analogeingänge eines meM-AD/-ADDA/-ADf/-ADfo besitzen die Kanalnummern 1-16. Der Analogausgang hat die Kanalnummer 1.

Die Richtung der Digitalports ist nicht umschaltbar. Dabei stehen die 4 (meM-ADfo: 8) Leitungen des ersten Ports (DIO1, Kanalnummer: 1) auf Eingang, die 4 (meM-ADfo: 8) Leitungen des zweiten Ports (DIO2, Kanalnummer: 2) auf Ausgang.

3.2.6 meM-PIO / meM-PIO-OEM

Um eine meM-PIO/meM-PIO-OEM mit **LibadX** zu öffnen, muss an **Open()** der String "**mempiousb**" übergeben werden. Mehrere USB Messsysteme lassen sich durch Angabe der Gerätenummer öffnen (1. Gerät mit "**mempiousb:0**", 2. Gerät mit "**mempiousb:1**", usw.). Die Reihenfolge der Geräte wird durch das Anstecken bestimmt.

Da die USB Messsysteme im laufenden Betrieb an- und wieder abgesteckt werden können, ist es möglich, dass die Gerätenummern nicht in aufsteigender Reihenfolge vergeben sind. Werden beispielsweise drei Geräte angesteckt und dann das zweite Gerät wieder abgesteckt, sind die beiden verbleibenden Geräte mit "**mempiousb:0**" und "**mempiousb:2**" anzusprechen.

Um unabhängig von dieser Ansteckreihenfolge zu sein, kann ein Gerät auch mit einer bestimmten Seriennummer geöffnet werden. Das Gerät mit der Seriennummer 157 lässt sich zum Beispiel durch Angabe von "**mempiousb:@157**" ansprechen.

Messsystem	Digital	Kanalnummer
meM-PIO, meM-PIO-OEM	3 Ports (je 8 Bit)	1..3 (Bit 0..7)

Die Richtung der Leitungen ist für jeden Port getrennt einstellbar. Die Umstellung erfolgt portweise (s. "**DigitalDirection**", S. 54). Dabei besitzt der erste Port (DIO1) die Kanalnummer 1, der zweite Port (DIO2) die Kanalnummer 2 und der dritte Port (DIO3) die Kanalnummer 3.

3.2.7 USB-AD

Um ein USB-AD mit der **LibadX** zu öffnen, muss an **Open()** der String "**usb-ad**" übergeben werden. Mehrere USB Messsysteme lassen sich durch Angabe der Gerätenummer öffnen (1. Gerät mit "**usb-ad:0**", 2. Gerät mit "**usb-ad:1**", usw.). Die Reihenfolge der Geräte wird durch das Anstecken bestimmt.

Da USB Messsysteme im Betrieb an- und abgesteckt werden können, ist es möglich, dass die Gerätenummern nicht aufeinander folgend vergeben sind. Steckt man z. B. drei USB-AD an und entfernt dann das 2. Gerät, sind die verbleibenden USB-AD mit "**usb-ad:0**" und "**usb-ad:2**" anzusprechen.

Um unabhängig von dieser Ansteckreihenfolge zu sein, kann ein Gerät auch mit einer bestimmten Seriennummer geöffnet werden. Das Gerät mit der Seriennummer 157 lässt sich zum Beispiel mit "**usb-ad:@157**" ansprechen.

Messsystem	Analog	Kanalnummer	range (Messber.)	range (Ausgabebereich)	Digital	Richtung
USB-AD	16 Eingänge 1 Ausgang	1..16 1	0 ($\pm 5.12V$)	0 ($\pm 5.12V$)	2 Ports (je 4 Bit)	1: Eingang (Bit 0..3) 2: Ausgang (Bit 0..3)

Die 16 Analogeingänge eines USB-AD besitzen die Kanalnummern 1-16. Der Analogausgang hat die Kanalnummer 1.



Aus Kompatibilitätsgründen lässt sich alternativ für die analogen Eingänge auch der Messbereich 33 und für den analogen Ausgang der Ausgabebereich 1 angeben.

Die Richtung der Digitalports ist nicht umschaltbar. Dabei stehen die 4 Leitungen des ersten Ports (DIO1, Kanalnummer: 1) auf Eingang, die 4 Leitungen des zweiten Ports (DIO2, Kanalnummer: 2) auf Ausgang.

Beispiel:

```
VB      If LIBADXL.Open("usb-ad:0") Then

          Dim tmp As Integer
          tmp = LIBADXL.DigitalIn(1)

          Dim bool As Boolean
          ' liest den Zustand der ersten Leitung des Ports 1
          bool = LIBADXL.DigitalInLine(1, 0)

          ' alle Leitungen erstmal löschen
          LIBADXL.DigitalOut(2) = 0
          ' Leitung 2 high
          LIBADXL.DigitalOutLine(2, 1) = True

          Dim val As Double
          ' liest den Analog Wert vom Analog In 1 im Messbereich 0
          val = LIBADXL.AnalogIn(1, 0)

          ' setze den Analog Out 1 auf 4.5 Volt
          LIBADXL.AnalogOut(1, 0) = 4.5

          LIBADXL.Close
      End If
```

3.2.8 USB-AD12f

Um ein USB-AD12f mit der **LibadX** zu öffnen, muss an **Open()** der String "**usbad12f**" übergeben werden. Mehrere USB Messsysteme lassen sich durch Angabe der Gerätenummer öffnen (1. Gerät mit "**usbad12f:0**", 2. Gerät mit "**usbad12f:1**", usw.). Die Reihenfolge der Geräte wird durch das Anstecken bestimmt.

Da USB Messsysteme im Betrieb an- und abgesteckt werden können, ist es möglich, dass die Gerätenummern nicht aufeinander folgend vergeben sind. Steckt man z. B. drei USB-AD12f an und entfernt dann das 2. Gerät, sind die verbleibenden USB-AD12f mit "**usbad12f:0**" und "**usbad12f:2**" anzusprechen.

Um unabhängig von dieser Ansteckreihenfolge zu sein, kann ein Gerät auch mit einer bestimmten Seriennummer geöffnet werden. Das Gerät mit der Seriennummer 157 lässt sich zum Beispiel mit "**usbad12f:@157**" ansprechen.

Messsystem	Analog	Kanalnummer	range (Messber.)	range (Ausgabebereich)	Digital	Richtung
USB-AD12f	16 Eingänge 1 Ausgang	1..16 1	0 ($\pm 10.24V$)	0 ($\pm 5.12V$)	2 Ports (je 4 Bit)	1: Eingang (Bit 0..3) 2: Ausgang (Bit 0..3)

Die 16 Analogeingänge eines USB-AD12f besitzen die Kanalnummern 1-16. Der Analogausgang hat die Kanalnummer 1.

Die Richtung der Digitalports ist nicht umschaltbar. Dabei stehen die 4 Leitungen des ersten Ports (DIO1, Kanalnummer: 1) auf Eingang, die 4 Leitungen des zweiten Ports (DIO2, Kanalnummer: 2) auf Ausgang.

Der erste Digitaleingang (Bit 1) ist als 16-Bit Zähler verwendbar. Dieser wird von der **LibadX** wie ein Analogkanal behandelt. Dazu muss die Kanalnummer des Zählers in den Analogroutinen **AnalogIn** (s. S. 51), **AnalogOut** (s. S. 51) und **ScanAnalogIn** (s. S. 45) um den Zählerkanaltyp (**hex 0x08000000**) erweitert werden, so dass sich für den Zähler 1 die Kanalnummer **0x08000001** in hexadezimaler Schreibweise ergibt. Der zu übergebende Rangeparameter ist immer '0'. Ein Zurücksetzen des Zählers erfolgt über den Befehl **AnalogOut**, dem der Wert 0 übergeben werden muss.

3.2.9 USB-AD16f

Um ein USB-AD16f mit der **LibadX** zu öffnen, muss an **Open()** der String "**usbbase**" übergeben werden. Mehrere USB-AD16f Messsysteme lassen sich durch Angabe der Gerätenummer öffnen (1. Gerät mit "**usbbase:0**", 2. Gerät mit "**usbbase:1**", usw.). Die Reihenfolge der Geräte wird durch das Anstecken bestimmt.

Da USB Messsysteme im Betrieb an- und abgesteckt werden können, ist es möglich, dass die Gerätenummern nicht aufeinander folgend vergeben sind. Steckt man z. B. drei USB-AD16f an und entfernt dann das 2. Gerät, sind die verbleibenden USB-AD16f mit "**usbbase:0**" und "**usbbase:2**" anzusprechen.

Um unabhängig von dieser Ansteckreihenfolge zu sein, kann ein Gerät auch mit einer bestimmten Seriennummer geöffnet werden. Das Gerät mit der Seriennummer 157 lässt sich zum Beispiel mit "**usbbase:@157**" ansprechen.

Messsystem	Analog	Kanalnummer	range (Messber.)	range (Ausgabebereich)	Digital	Richtung
USB-AD16f	16 Eingänge 2 Ausgänge	1..16 1 .. 2	0 ($\pm 1.024V$) 1 ($\pm 2.048V$) 2 ($\pm 5.12V$) 3 ($\pm 10.24V$)	0 ($\pm 10.24V$)	2 Ports (je 4 Bit)	1: Eingang (Bit 0..3) 2: Ausgang (Bit 0..3)

Die 16 Analogeingänge eines USB-AD16f besitzen die Kanalnummern 1-16. Die beiden Analogausgänge haben die Kanalnummern 1 und 2.

Die Richtung der Digitalports ist nicht umschaltbar. Dabei stehen die 4 Leitungen des ersten Ports (DIO1, Kanalnummer: 1) auf Eingang, die 4 Leitungen des zweiten Ports (DIO2, Kanalnummer: 2) auf Ausgang.

Außerdem besitzt das USB-AD16f einen Zählereingang, der von der **LibadX** wie ein Analogkanal behandelt wird. Dazu muss die Kanalnummer des Zählers in den Analogroutinen **AnalogIn** (s. S. 51), **AnalogOut** (s. S. 51) und **ScanAnalogIn** (s. S. 45) um den Zählerkanaltyp (**hex 0x08000000**) erweitert werden, so dass sich für den Zähler 1 die Kanalnummer **0x08000001** in hexadezimaler Schreibweise ergibt. Der zu übergebende Rangeparameter ist immer '0'. Ein Zurücksetzen des Zählers erfolgt über den Befehl **AnalogOut**, dem der Wert 0 übergeben werden muss.

3.2.10 USB-PIO / USB-PIO-OEM

Um eine USB-PIO / USB-PIO-OEM mit der **LibadX** zu öffnen, muss an **Open()** der String "**usb-pio**" übergeben werden. Mehrere USB Messsysteme lassen sich durch Angabe der Gerätenummer öffnen (1. Gerät mit "**usb-pio:0**", 2. Gerät mit "**usb-pio:1**", usw.). Die Reihenfolge der Geräte wird durch das Anstecken bestimmt.

Da die USB Messsysteme im laufenden Betrieb an- und wieder abgesteckt werden können, ist es möglich, dass die Geräteummern nicht in aufsteigender Reihenfolge vergeben sind. Werden beispielsweise drei Geräte angesteckt und dann das zweite Gerät wieder abgesteckt, sind die beiden verbleibenden Geräte mit "**usb-pio:0**" und "**usb-pio:2**" anzusprechen.

Um unabhängig von dieser Ansteckreihenfolge zu sein, kann ein Gerät auch mit einer bestimmten Seriennummer geöffnet werden. Das Gerät mit der Seriennummer 157 lässt sich zum Beispiel durch Angabe von "**usb-pio:@157**" ansprechen.

Messsystem	Digital	Kanalnummer
USB-PIO, USB-PIO-OEM	3 Ports (je 8 Bit)	1..3 (Bit 0..7)

Die Richtung der Leitungen ist für jeden Port getrennt einstellbar. Die Umstellung erfolgt portweise (s. "**DigitalDirection**", S. 54). Dabei besitzt der erste Port (DIO1) die Kanalnummer 1, der zweite Port (DIO2) die Kanalnummer 2 und der dritte Port (DIO3) die Kanalnummer 3.

Beispiel:

VB

```
If LIBADXL.Open("usb-pio:0") Then
LIBADXL.DigitalDirection(1) = &H0 ' alle Output
LIBADXL.DigitalDirection(2) = &HFF ' alle Input
LIBADXL.DigitalDirection(3) = &H0 ' alle Output

Dim tmp As Integer
' Liest den Zustand aller Leitungen des Ports 2
tmp = LIBADXL.DigitalIn(2)

Dim bool As Boolean
' liest den Zustand der ersten Leitung des Ports 2
bool = LIBADXL.DigitalInLine(2, 0)

' alle Leitungen erst mal löschen
LIBADXL.DigitalOut(1) = 0
' Leitung 8 des Ports 1 high
LIBADXL.DigitalOutLine(1, 7) = True

' Port 3 auf &H15 setzen = Leitung 1, 3, 5 high
LIBADXL.DigitalOut(3) = &H15

LIBADXL.Close
End If
```

4 Schnittstellen und Funktionen

4.1 Die Schnittstelle LibadX

Die Schnittstelle **LibadX** wird direkt vom LibadX ActiveX Control exportiert. Sie dient dazu, die Verbindung zum Messdatenserver herzustellen.

4.1.1 Überblick

Funktion	Beschreibung
Open	öffnet eine Verbindung zu einem Messsystem
Close	schließt die Verbindung zu einem Messsystem
GetVersion	liefert die Versionsnummer der LIBAD4.dll
LastError	liefert den letzten Fehlercode zurück
LastErrorString	liefert eine Beschreibung des letzten Fehlers
ScanPrepare	bereitet einen Scan vor
ScanAnalogIn	fügt einen analogen Eingang der Scanliste hinzu
ScanDigitalIn	fügt einen digitalen Eingang der Scanliste hinzu
Scan	startet einen vorbereiteten Scan
ScanSave	speichert einen durchgeführten Scan
FileOpen	legt ein Dateiojekt an, mit dem auf gespeicherte Messdateien zugegriffen werden kann
FileCreatePrepare	bereitet das Anlegen einer Scandatei vor
FileCreateAnalogIn	fügt der Kanalliste einen Analogkanal hinzu
FileCreateDigital	fügt der Kanalliste einen Digitalkanal hinzu
FileCreate	erzeugt eine vorbereitete Scandatei
AnalogIn	liefert den aktuellen Wert eines Analogeingangs
AnalogOut	liefert den aktuellen Wert eines Analogausgangs
DigitalIn	liefert den aktuellen Wert eines Digitaleingangs

DigitalOut	liefert den aktuellen Wert eines Digitalausgangs
DigitalInLine	liefert den aktuellen Wert einer digitalen Eingangsleitung
DigitalOutLine	liefert aktuellen Wert einer digitalen Ausgangsleitung
DigitalDirection	setzt/liefert die eingestellte Richtung eines Digitalkanals
Sample	liest den Wert eines Samples in einem Scan
AboutBox	zeigt die AboutBox von LibadX

4.1.2 Open

C++	<code>VARIANT_BOOL Open (_bstr_t path)</code>
------------	---

BASIC	<code>Function Open (path As String) As Boolean</code>
--------------	--

Delphi	<code>function Open (const path: WideString): WordBool</code>
---------------	---

Die Funktion **Open()** stellt eine Verbindung zum Messsystem her. Es wird der Name des Messsystems übergeben. Der übergebene String wird ohne Berücksichtigung von Groß- und Kleinschreibung verwendet, das heißt "**pcibase**" und "**Pcibase**" öffnen beide die PCIe-BASE / PCI-BASEII/300/1000 / PCI-PIO.

Wurde eine Verbindung mit dem Messsystem geöffnet, gibt **Open** den Wert **TRUE** zurück, beim einem Fehler **FALSE**. Eine ausführliche Beschreibung des Befehls **Open()** finden Sie im Kapitel "Verbindung zum Messsystem", S. 26.

Eine Übersicht aller möglichen Befehle erhalten Sie im Kapitel "Überblick", S. 40.

4.1.3 Close

C++	<code>HRESULT Close ()</code>
------------	-------------------------------

BASIC	<code>Sub Close ()</code>
--------------	---------------------------

Delphi	<code>procedure Close</code>
---------------	------------------------------

Die Funktion **Close()** schließt eine Verbindung zum Messsystem.

Eine Übersicht aller möglichen Befehle erhalten Sie im Kapitel "Überblick", S. 40.

4.1.4 GetVersion

C++	<code>long GetVersion ()</code>
------------	---------------------------------

BASIC	<code>Function GetVersion () As Long</code>
--------------	---

Delphi	<code>function GetVersion: Integer</code>
---------------	---

Die Funktion **GetVersion()** liefert die Version der LIBAD4.dll zurück, die von **LibadX** benutzt wird.

Eine Übersicht aller möglichen Befehle erhalten Sie im Kapitel "Überblick", S. 40.

4.1.5 LastError

C++	<code>long GetLastError ()</code>
------------	-----------------------------------

BASIC	<code>Function GetLastError () As Long</code>
--------------	---

Delphi	<code>function GetLastError: Integer</code>
---------------	---

Gibt die Nummer des zuletzt aufgetretenen Fehlers zurück. Ist kein Fehler aufgetreten, gibt die Funktion 0 zurück.

Eine Übersicht aller möglichen Befehle erhalten Sie im Kapitel "Überblick", S. 40.

4.1.6 LastErrorString

C++	<code>_bstr_t GetLastErrorString ()</code>
------------	--

BASIC	<code>Function GetLastErrorString () As String</code>
--------------	---

Delphi	<code>function GetLastErrorString: WideString</code>
---------------	--

Gibt eine Beschreibung des zuletzt aufgetretenen Fehlers zurück. Ist kein Fehler aufgetreten, gibt die Funktion "" zurück.

Eine Übersicht aller möglichen Befehle erhalten Sie im Kapitel "Überblick", S. 40.

4.1.7 ScanPrepare

C++	<code>HRESULT ScanPrepare (float sample_rate, long samples)</code>
------------	--

BASIC	<code>Sub ScanPrepare (sample_rate As Single, samples As Long)</code>
--------------	---

Delphi	<code>procedure ScanPrepare (sample_rate: Single; samples: Integer)</code>
---------------	--

Um einen Scan zu starten, muss zuerst **ScanPrepare()** aufgerufen werden. Dadurch wird die **LibadX** auf einen Scan vorbereitet und die Abtastrate auf **sample_rate** und die Anzahl der zu speichernden Werte auf **samples** gestellt.

Um einen Kanal der Kanalliste für den Scan hinzuzufügen, ruft man **ScanAnalogIn()** bzw. **ScanDigitalIn()** auf. Der Start des Scans erfolgt durch einen Aufruf der Funktion **Scan()**.

Folgender Visual Basic® Beispielcode zeigt die Funktionsweise:

VB	<pre>' 1000 Messwerte, 100Hz (0.01 Sek.) LIBADXL.ScanPrepare 0.01, 1000 ' Kanal 1 & 2 speichern LIBADXL.ScanAnalogIn 1, 0 LIBADXL.ScanAnalogIn 2, 0 ' Zaehler 1 speichern LIBADXL.ScanDigitalIn &h08000001 ' Digital Port 1 speichern LIBADXL.ScanDigitalIn 1 ' Scan durchführen LIBADXL.Scan ' Scan speichern LIBADXL.ScanSave "scan.lfx"</pre>
-----------	---

Eine Übersicht aller möglichen Befehle erhalten Sie im Kapitel "Überblick", S. 40.

4.1.8 ScanAnalogIn

C++	<code>HRESULT ScanAnalogIn (long index, long range)</code>
------------	--

BASIC	<code>Sub ScanAnalogIn (index as Long, range as Long)</code>
--------------	--

Delphi	<code>procedure ScanAnalogIn (index, range: Integer)</code>
---------------	---

Mit `ScanAnalogIn()` fügt man den analogen Kanal oder Zähler mit der Nummer **index** und dem Bereich **range** der Kanalliste des Scans hinzu. Die Funktion löst eine Exception aus, wenn vorher kein Scan mit `ScanPrepare()` (s. S. 43) vorbereitet wurde.



- **Aufgrund von Einschränkungen bei den meisten Messkarten müssen die Eingangskanäle unbedingt in aufsteigender Reihenfolge der Kanalliste hinzugefügt werden! Werden sowohl analoge Eingänge als auch Zähler oder digitale Eingänge abgetastet, müssen zuerst alle analogen, dann die Zähler und schließlich die digitalen Kanäle angegeben werden!**
 - **Bei Zählern muss die Indexnummer um den Zählerkanaltyp (hex 0x08000000) erweitert werden. Für den Zähler 1 ergibt dies beispielsweise die Indexnummer 0x08000001 in hexadezimaler Schreibweise.**
-

Eine Übersicht aller möglichen Befehle erhalten Sie im Kapitel "Überblick", S. 40.

4.1.9 ScanDigitalIn

C++	<code>HRESULT ScanDigitalIn (long index)</code>
------------	---

BASIC	<code>Sub ScanDigitalIn (index as Long)</code>
--------------	--

Delphi	<code>procedure ScanDigitalIn (index: Integer)</code>
---------------	---

Mit `ScanDigitalIn()` fügt man digitalen Kanal mit der Nummer **index** der Kanalliste des Scans hinzu. Die Funktion löst eine Exception aus, wenn vorher kein Scan mit `ScanPrepare()` (s. S. 43) vorbereitet wurde.



Aufgrund von Einschränkungen bei den meisten Messkarten müssen die Eingangskanäle unbedingt in aufsteigender Reihenfolge der Kanalliste hinzugefügt werden! Werden sowohl analoge Eingänge als auch Zähler oder digitale Eingänge abgetastet, müssen zuerst alle analogen, dann die Zähler und schließlich die digitalen Kanäle angegeben werden!

Eine Übersicht aller möglichen Befehle erhalten Sie im Kapitel "Überblick", S. 40.

4.1.10 Scan

C++	<code>VARIANT_BOOL Scan ();</code>
------------	------------------------------------

BASIC	<code>Function Scan () As Boolean</code>
--------------	--

Delphi	<code>function Scan : WordBool</code>
---------------	---------------------------------------

Mit `Scan()` wird ein mit `ScanPrepare()`, `ScanAnalogIn()` und `ScanDigitalIn()` vorbereiteter Scan gestartet. Während des Scans blockiert das aufrufende Programm bzw. der aufrufende Thread.

Die Funktion löst eine Exception aus, wenn vorher kein Scan mit `ScanPrepare()` (s. S. 43) vorbereitet oder kein Kanal der Kanalliste hinzugefügt wurde.

Eine Übersicht aller möglichen Befehle erhalten Sie im Kapitel "Überblick", S. 40.

4.1.11 ScanSave

C++	<code>VARIANT_BOOL ScanSave (_bstr_t path);</code>
------------	--

BASIC	<code>Function ScanSave (path As String) As Boolean</code>
--------------	--

Delphi	<code>function ScanSave (const path: WideString): WordBool</code>
---------------	---

Mit **ScanSave()** wird ein mit **Scan()** durchgeführter Scan gespeichert.

Die Funktion löst eine Exception aus, wenn vorher kein Scan mit **Scan()** durchgeführt wurde.

Eine Übersicht aller möglichen Befehle erhalten Sie im Kapitel "Überblick", S. 40.

4.1.12 FileOpen

C++	<code>INvxFilePtr FileOpen (_bstr_t path)</code>
------------	--

BASIC	<code>Function FileOpen (path As String) As INvxFile</code>
--------------	---

Delphi	<code>function FileOpen (const path: WideString): INvxFile</code>
---------------	---

Öffnet das angegebene Messdatenfile. Falls das File nicht existiert oder nicht geöffnet werden kann, löst die Funktion eine Exception aus.

Eine Übersicht aller möglichen Befehle erhalten Sie im Kapitel "Überblick", S. 40.

4.1.13 FileCreatePrepare

C++	<code>HRESULT FileCreatePrepare (long samples)</code>
------------	---

BASIC	<code>Sub FileCreatePrepare (samples As Long)</code>
--------------	--

Delphi	<code>procedure FileCreatePrepare (samples: Integer)</code>
---------------	---

Die Erstellung einer Messdatei erfolgt wie bei einem Scan. Dazu muss zuerst **FileCreatePrepare()** mit der Anzahl der zu speichernden Werte aufgerufen werden.

Um der Kanalliste einer Datei einen Kanal hinzuzufügen, ruft man **FileCreateAnalogIn()** bzw. **FileCreateDigital()** auf. Die Datei wird durch einen Aufruf von **FileCreate()** schließlich erzeugt.

Folgender Visual Basic® Beispielcode demonstriert die Funktionsweise:

VB	<pre>' 1000 Messwerte LIBADXL.FileCreatePrepare 1000 ' 2 analoge Kanäle LIBADXL.FileCreateAnalogIn LIBADXL.FileCreateAnalogIn ' 1 Zählerkanal Const AD_CHA_TYPE_COUNTER as Integer = &h08000000 LIBADXL.FileCreateDigital AD_CHA_TYPE_COUNTER ' 1 Digitalkanal mit 16 Leitungen LIBADXL.FileCreateDigital 16 ' Datei anlegen LIBADXL.FileCreate "scan.lfx"</pre>
-----------	--

Eine Übersicht aller möglichen Befehle erhalten Sie im Kapitel "Überblick", S. 40.

4.1.14 FileCreateAnalogIn

C++	<code>long FileCreateAnalogIn ()</code>
------------	--

BASIC	<code>Function FileCreateAnalogIn () As Long</code>
--------------	--

Delphi	<code>function FileCreateAnalogIn: Integer;</code>
---------------	--

Mit **FileCreateAnalogIn()** fügt man einen analogen Kanal oder Zähler der Kanalliste einer zu erstellenden Datei hinzu. Als Rückgabewert erhält man den Index des Kanals in der Datei. Die Funktion löst eine Exception aus, wenn vorher keine Messdatei mit **FileCreatePrepare()** (s. S. 47) vorbereitet wurde.

Eine Übersicht aller möglichen Befehle erhalten Sie im Kapitel "Überblick", S. 40.

4.1.15 FileCreateDigital

C++	<code>long FileCreateDigital (long lines)</code>
------------	--

BASIC	<code>Function FileCreateDigital (lines As Long) As Long</code>
--------------	---

Delphi	<code>function FileCreateDigital(lines: Integer): Integer;</code>
---------------	---

Mit **FileCreateDigital()** fügt man einen digitalen Kanal der Kanalliste einer zu erstellenden Datei hinzu.

Bei Digitalkanälen ist **lines** die Anzahl der zu speichernden Leitungen. Diese darf 32 nicht überschreiten. Als Rückgabewert erhält man den Index des Kanals in der Datei.



Vor dem Schreiben von Daten in die Datei müssen die Signalparameter (s. Kap. "Die Schnittstelle INvxSignal", S. 59) "yMax" (s. S. 67) und "yMin" (s. S. 66) unbedingt richtig gesetzt werden, da sonst diese nicht richtig geschrieben werden. Das y-Using (s. Kap. "ySetUsing", S. 69) sollte auch entsprechend angepasst werden.

Die Funktion löst eine Exception aus, wenn vorher keine Messdatei mit `FileCreatePrepare()` (s. S. 47) vorbereitet wurde.

Eine Übersicht aller möglichen Befehle erhalten Sie im Kapitel "Überblick", S. 40.

4.1.16 FileCreate

C++	<code>INvxFilePtr FileCreate (_bstr_t path)</code>
------------	--

BASIC	<code>Function FileCreate (path As String) As INvxFile</code>
--------------	---

Delphi	<code>function FileCreate (const path: WideString): INvxFile</code>
---------------	---

Mit `FileCreate()` wird ein mit `FileCreatePrepare()`, `FileCreateAnalogIn()` und `FileCreateDigital()` vorbereitete Messdatei erzeugt. Die Funktion löst eine Exception aus, wenn vorher keine Messdatei mit `FileCreatePrepare()` (s. S. 47) vorbereitet oder kein Kanal der Kanalliste hinzugefügt wurde.

Eine Übersicht aller möglichen Befehle erhalten Sie im Kapitel "Überblick", S. 40.

4.1.17 AnalogIn

C++	<code>__declspec(property(get=GetAnalogIn)) float AnalogIn[[]]</code>
------------	---

BASIC	<code>Property AnalogIn (index As Long, range as Long) As Single</code>
--------------	---

Delphi	<code>property AnalogIn [index, range: Integer]: Single readonly</code>
---------------	---

Liefert für den analogen Eingangskanal mit der Nummer **index** den aktuell gemessenen Wert im Messbereich **range** dieses Kanals. Der Wert dieses Properties ist nur lesbar.

Bei Zählern muss die Indexnummer um den Zählerkanaltyp (hex 0x08000000) erweitert werden. Für den Zähler 1 ergibt dies beispielsweise die Indexnummer 0x08000001 in hexadezimaler Schreibweise.

Die Funktion löst eine Exception aus, wenn vorher keine Verbindung mit einem Messsystem durch **Open ()** hergestellt wurde.

Eine Übersicht aller möglichen Befehle erhalten Sie im Kapitel "Überblick", S. 40.

4.1.18 AnalogOut

C++	<code>__declspec(property(get=GetAnalogOut,put=PutAnalogOut)) float AnalogOut[[]]</code>
------------	--

BASIC	<code>Property AnalogOut (index As Long, range as Long) As Single</code>
--------------	--

Delphi	<code>property AnalogOut [index, range: Integer]: Single</code>
---------------	---

Legt für den Ausgangskanal mit der Nummer **index** den aktuellen Wert im Ausgabebereich **range** dieses Kanals fest oder gibt diesen zurück.

Bei Zählern muss die Indexnummer um den Zählerkanaltyp (hex 0x08000000) erweitert werden. Für den Zähler 1 ergibt dies beispielsweise die Indexnummer 0x08000001 in hexadezimaler Schreibweise.

Die Funktion löst eine Exception aus, wenn vorher keine Verbindung mit einem Messsystem durch **Open()** hergestellt wurde.

Eine Übersicht aller möglichen Befehle erhalten Sie im Kapitel "Überblick", S. 40.

4.1.19 DigitalIn

C++	<code>__declspec(property(get=GetDigitalIn)) long DigitalIn[]</code>
------------	--

BASIC	<code>Property DigitalIn (index As Long) As Long</code>
--------------	---

Delphi	<code>property DigitalIn [index: Integer]: Integer readonly</code>
---------------	--

Liefert für den digitalen Eingangskanal mit der Nummer **index** den aktuell gemessenen Wert. Der Wert dieses Properties ist nur lesbar.

Die Funktion löst eine Exception aus, wenn vorher keine Verbindung mit einem Messsystem durch **Open()** hergestellt wurde.

Eine Übersicht aller möglichen Befehle erhalten Sie im Kapitel "Überblick", S. 40.

4.1.20 DigitalOut

C++	<code>__declspec(property(get=GetDigitalOut,put=PutDigitalOut)) long DigitalOut[];</code>
------------	---

BASIC	<code>Property DigitalOut (index As Long) As Long</code>
--------------	--

Delphi	<code>property DigitalOut [index: Integer]: Integer</code>
---------------	--

Legt für den Ausgangskanal mit der Nummer **index** den aktuellen Wert dieses Kanals fest oder gibt diesen zurück.

Die Funktion löst eine Exception aus, wenn vorher keine Verbindung mit einem Messsystem durch **Open()** hergestellt wurde.

Eine Übersicht aller möglichen Befehle erhalten Sie im Kapitel "Überblick", S. 40.

4.1.21 DigitalInLine

C++	<code>__declspec(property(get=GetDigitalInLine)) VARIANT_BOOL DigitalInLine[][];</code>
------------	---

BASIC	<code>Property DigitalInLine (index As Long, line As Long) As Boolean</code>
--------------	--

Delphi	<code>property DigitalInLine [index, line: Integer]: WordBool readonly</code>
---------------	---

Liefert für die Leitung Nummer **line** des digitalen Eingangskanals mit der Nummer **index** den aktuell gemessenen Wert. Der Wert dieses Properties ist nur lesbar.

Die Funktion löst eine Exception aus, wenn vorher keine Verbindung mit einem Messsystem durch **Open()** hergestellt wurde.

Eine Übersicht aller möglichen Befehle erhalten Sie im Kapitel "Überblick", S. 40.

4.1.22 DigitalOutLine

C++	<code>__declspec(property(get=GetDigitalOutLine, put=PutDigitalOutLine)) VARIANT_BOOL DigitalOutLine[[]];</code>
------------	--

BASIC	<code>Property DigitalOutLine (index As Long, line As Long) As Boolean</code>
--------------	---

Delphi	<code>property DigitalOutLine [index, line: Integer]: WordBool</code>
---------------	---

Legt für die Leitung Nummer **line** des digitalen Ausgangskanals mit der Nummer **index** den aktuell gesetzten Wert fest oder gibt diesen zurück.

Die Funktion löst eine Exception aus, wenn vorher keine Verbindung mit einem Messsystem durch **Open()** hergestellt wurde.

Eine Übersicht aller möglichen Befehle erhalten Sie im Kapitel "Überblick", S. 40.

4.1.23 DigitalDirection

C++	<code>__declspec(property(get=GetDigitalDirection, put=PutDigitalDirection)) long DigitalDirection[];</code>
------------	--

BASIC	<code>Property DigitalDirection (index As Long) As Long</code>
--------------	--

Delphi	<code>property DigitalDirection [index: Integer]: Integer</code>
---------------	--

Legt die Ein-/Ausgaberrichtung aller Leitungen des Digitalkanals mit der Nummer **index** fest oder gibt diesen zurück. Dabei wird eine Bitmaske übergeben, die die Richtung der Digitalleitung beschreibt. Jedes gesetzte Bit definiert eine Eingangs-

leitung, jedes gelöschte Bit eine Ausgangsleitung. Das Bit #0 legt die Richtung der ersten Leitung des Digitalports fest.

Die Funktion löst eine Exception aus, wenn vorher keine Verbindung mit einem Messsystem durch `Open()` hergestellt wurde.

Eine Übersicht aller möglichen Befehle erhalten Sie im Kapitel "Überblick", S. 40.

4.1.24 Sample

C++	<code>__declspec(property(get=GetSample,put=PutSample)) float Sample[[]]</code>
------------	---

BASIC	<code>Property Sample (index As Long, pos As Long) As Single</code>
--------------	---

Delphi	<code>property Sample [index, pos: Integer]: Single</code>
---------------	--

Legt das Sample des Kanals **index** an der Position **pos** des durchgeführten Scans fest oder gibt diesen zurück.

Die Funktion löst eine Exception aus, wenn vorher kein Scan durchgeführt wurde oder **index** bzw. **pos** ungültig sind.



Bei 32-Bit Zählern geht bei hohen Zählerständen durch die Single Floating Point Verwendung Information verloren. Es steht nur ein Wertebereich von ± 16777216 zur Verfügung.

Eine Übersicht aller möglichen Befehle erhalten Sie im Kapitel "Überblick", S. 40.

4.1.25 AboutBox

C++	<code>HRESULT AboutBox ()</code>
BASIC	<code>Sub AboutBox ()</code>
Delphi	<code>procedure AboutBox</code>

Zeigt die AboutBox von **LibadX**.

Eine Übersicht aller möglichen Befehle erhalten Sie im Kapitel "Überblick", S. 40.

4.2 Die Schnittstelle INvxFile

Die Schnittstelle **INvxFile** ermöglicht den Zugriff auf gespeicherte Messdaten.

4.2.1 Überblick

Funktion	Beschreibung
Open	öffnet ein Messdatenfile
Create	legt ein neues Messdatenfile an
Close	schließt das Messdatenfile wieder
SignalCount	liefert die Anzahl der Signale im Messdatenfile
Signal	gibt die Schnittstelle zu einem Signal im Messdatenfile zurück

4.2.2 Open

C++	<code>HRESULT Open(_bstr_t fileName);</code>
------------	--

BASIC	<code>Sub Open(fileName As String)</code>
--------------	---

Delphi	<code>procedure Open(const fileName: WideString);</code>
---------------	--

Öffnet das angegebene Messdatenfile. Falls das File nicht existiert oder nicht geöffnet werden kann, löst die Funktion eine Exception aus.

Eine Übersicht aller möglichen Befehle erhalten Sie im Kapitel "Überblick", S. 56.

4.2.3 Create

C++	<code>HRESULT Create(_bstr_t fileName, long signalCount, long sampleCount);</code>
------------	--

BASIC	<code>Sub Create(fileName As String, signalCount As Long, sampleCount As Long)</code>
--------------	---

Delphi	<code>procedure Create(const fileName: WideString; signalCount: Integer; sampleCount: Integer);</code>
---------------	--

Erzeugt ein neues Messdatenfile. In der Datei werden **signalCount** Signale angelegt, wobei jedes Signal **sampleCount** Messwerte speichern kann.

Eine Übersicht aller möglichen Befehle erhalten Sie im Kapitel "Überblick", S. 56.

4.2.4 Close

C++	<code>HRESULT Close();</code>
------------	-------------------------------

BASIC	<code>Sub Close()</code>
--------------	--------------------------

Delphi	<code>procedure Close;</code>
---------------	-------------------------------

Schließt eine Messdatei, die mit **Open()** geöffnet oder **Create()** erzeugt wurde.

Eine Übersicht aller möglichen Befehle erhalten Sie im Kapitel "Überblick", S. 56.

4.2.5 SignalCount

C++	<code>long SignalCount();</code>
------------	----------------------------------

BASIC	<code>Function SignalCount() As Long</code>
--------------	---

Delphi	<code>function SignalCount: Integer;</code>
---------------	---

Liefert die Signalanzahl im Messdatenfile. Die Funktion löst eine Exception aus, wenn vorher keine Messdatei mit **Open()** geöffnet oder mit **Create()** angelegt worden ist.

Eine Übersicht aller möglichen Befehle erhalten Sie im Kapitel "Überblick", S. 56.

4.2.6 Signal

C++	<code>INvxSignalPtr Signal(long index);</code>
------------	--

BASIC	<code>Function Signal(index As Long) As INvxSignal</code>
--------------	---

Delphi	<code>function Signal(index: Integer): INvxSignal;</code>
---------------	---

Liefert ein Signal aus dem Messdatenfile. Das erste Signal im File hat den Index 1.

Eine Übersicht aller möglichen Befehle erhalten Sie im Kapitel "Überblick", S. 56.

4.3 Die Schnittstelle INvxSignal

Die Schnittstelle **INvxSignal** ermöglicht den Zugriff auf ein einzelnes Signal aus einem Messdatenfile.

4.3.1 Überblick

Funktion	Beschreibung
Name	Name des Signals
GroupName	Gruppenname des Signals
Comment	Kommentar des Signals
xStart	Startzeitpunkt des Signals
xEnd	Endzeitpunkt des Signals
xDelta	Abtastzeit des Signals
xUnit	Einheit der x-Achse
xSetUsing	legt das Zahlenformat der x-Achse fest
xGetUsing	liefert das Zahlenformat der x-Achse

yMin	untere Messbereichsgrenze
yMax	obere Messbereichsgrenze
yDefaultMin	untere Grenze des Standardanzeigebereichs
yDefaultMax	obere Grenze des Standardanzeigebereichs
yDelta	Auflösung des Signals
yUnit	y-Achseneinheit
ySetUsing	legt das Zahlenformat der y-Achse fest
yGetUsing	liefert das Zahlenformat der y-Achse
ScanStart	Datum zu Beginn der Messung
SampleCount	Anzahl der Messwerte im Signal
ScaleX	Skalierung der x-Achse
ScaleY	Skalierung der y-Achse
ResetDataPosition	Zurücksetzen der aktuellen Position im Signal
GetNextScaled	liefert das nächste skalierte Wertepaar
GetNextScaledDigital	liefert das nächste skalierte Wertepaar eines Digitalsignals
Unscale	hebt die Skalierung des Signals auf
NextSample	liefert den nächsten Messwert an der aktuellen Position im Signal
NextDigitalSample	liefert den nächsten Wert an der aktuellen Position im Digitalsignal
GetSampleAt	liefert einen Messwert an einer bestimmten zeitlichen Position im Signal
GetSampleAtOffset	liefert einen Messwert an einem bestimmten Offset im Signal
IsAnalog	überprüft, ob das Signal analoge Messwerte enthält
IsDigital	überprüft, ob das Signal digitale oder Zähler-Werte enthält

4.3.2 Name

C++	<code>__declspec(property(get=GetName,put=PutName)) _bstr_t Name;</code>
------------	--

BASIC	<code>Property Name As String</code>
--------------	--------------------------------------

Delphi	<code>property Name: WideString read Get_Name write Set_Name;</code>
---------------	--

Liefert den Namen des Signals.

Eine Übersicht aller möglichen Befehle erhalten Sie im Kapitel "Überblick", S. 59.

4.3.3 GroupName

C++	<code>__declspec(property(get=GetName,put=PutName)) _bstr_t Name;</code>
------------	--

BASIC	<code>Property GroupName As String</code>
--------------	---

Delphi	<code>property GroupName: WideString read Get_GroupName write Set_GroupName;</code>
---------------	---

Liefert den Gruppennamen des Signals.

Eine Übersicht aller möglichen Befehle erhalten Sie im Kapitel "Überblick", S. 59.

4.3.4 Comment

C++	<code>__declspec(property(get=GetComment,put=PutComment)) _bstr_t Comment;</code>
------------	---

BASIC	<code>Property Comment As String</code>
--------------	---

Delphi	<code>property Comment: WideString read Get_Comment write Set_Comment;</code>
---------------	---

Liefert den Kommentar des Signals.

Eine Übersicht aller möglichen Befehle erhalten Sie im Kapitel "Überblick", S. 59.

4.3.5 xStart

C++	<code>__declspec(property(get=GetxStart,put=PutxStart)) double xStart;</code>
------------	---

BASIC	<code>Property xStart As Double</code>
--------------	--

Delphi	<code>property xStart: Double read Get_xStart write Set_xStart;</code>
---------------	--

Liefert den Startzeitpunkt des Signals in Sekunden. Dieser Wert steht im Normalfall auf 0.0s. Nur Messungen mit Trigger geben hier als negativen Wert die Dauer der Vorgeschichte zurück.

Eine Übersicht aller möglichen Befehle erhalten Sie im Kapitel "Überblick", S. 59.

4.3.6 xEnd

C++	<code>__declspec(property(get=GetXEnd,put=PutxEnd)) double xEnd;</code>
------------	---

BASIC	<code>Property xEnd As Double</code>
--------------	--------------------------------------

Delphi	<code>property xEnd: Double read Get_xEnd write Set_xEnd;</code>
---------------	--

Liefert den Endzeitpunkt des Signals. Bitte beachten Sie, dass sich die Gesamtdauer des Signals vom Endzeitpunkt unterscheiden kann. Die Gesamtdauer lässt sich mit **xEnd-xStart** berechnen.

Eine Übersicht aller möglichen Befehle erhalten Sie im Kapitel "Überblick", S. 59.

4.3.7 xDelta

C++	<code>__declspec(property(get=GetXDelta,put=PutxDelta)) double xDelta;</code>
------------	---

BASIC	<code>Property xDelta As Double</code>
--------------	--

Delphi	<code>property xDelta: Double read Get_xDelta write Set_xDelta;</code>
---------------	--

Liefert die Abtastzeit des Signals in Sekunden.

Eine Übersicht aller möglichen Befehle erhalten Sie im Kapitel "Überblick", S. 59.

4.3.8 xUnit

C++	<code>__declspec(property(get=GetXUnit,put=PutxUnit)) _bstr_t xUnit;</code>
------------	---

BASIC	<code>Property xUnit As String</code>
--------------	---------------------------------------

Delphi	<code>property xUnit: WideString read Get_xUnit write Set_xUnit;</code>
---------------	---

Liefert die Einheit der x-Achse.

Eine Übersicht aller möglichen Befehle erhalten Sie im Kapitel "Überblick", S. 59.

4.3.9 xSetUsing

C++	<code>HRESULT xSetUsing(long format, long width, long frac, long opt);</code>
------------	---

BASIC	<code>Sub xSetUsing(format As Long, width As Long, frac As Long, opt As Long)</code>
--------------	--

Delphi	<code>procedure xSetUsing(format: Integer; width: Integer; frac: Integer; opt: Integer);</code>
---------------	---

Legt das Zahlenformat für die Ausgabe der Werte an der x-Achse dieses Signals fest. Dabei legt **format** das Ausgabeformat fest, **width** die Breite des Zahlenfeldes und **frac** die Anzahl der Ziffern nach der Kommastelle. Das Argument **opt** wird nur für die feste wissenschaftliche Darstellung verwendet und gibt dort die verwendete Zehnerpotenz an (siehe nachfolgende Tabelle).

Für **format** können folgende Zahlenwerte übergeben werden, alle anderen führen zu dem Fehlercode **E_INVALIDARG**:

Wert	Bedeutung	Beispiel: 17336,78
0	gibt die Messwerte als ganze Zahlen aus	17336
3	Der Messwert wird ohne Angabe einer Zehnerpotenz mit der Anzahl der Kommastellen laut frac ausgegeben.	17336,780
4	exponentielle Schreibweise E+xxx mit Zehnerpotenzangabe	1,734E+004
5	Wissenschaftliche Darstellung: Die Zehnerpotenz des Messwerts wird mit metrischen Einheiten dargestellt: p (10^{-12}), n (10^{-9}), μ (10^{-6}), m (10^{-3}), k (10^3), M (10^6), G (10^9)	17,337k
6	Feste wissenschaftliche Darstellung: Die Zehnerpotenz wird fest über den Parameter opt vorgegeben. Dabei können für opt folgende Werte vorkommen: 0: p (10^{-12}) 3: m (10^{-3}) 6: M (10^6) 1: n (10^{-9}) 4: (10^0) 7: G (10^9) 2: μ (10^{-6}) 5: k (10^3)	0,017M

Eine Übersicht aller möglichen Befehle erhalten Sie im Kapitel "Überblick", S. 59.

4.3.10 xGetUsing

```
C++      HRESULT xGetUsing(long *format,
                         long *width,
                         long *frac,
                         long *opt);
```

```
BASIC    Sub xGetUsing(format As Long,
                         width As Long,
                         frac As Long,
                         opt As Long)
```

Delphi	<pre>procedure xGetUsing(var format: Integer; var width: Integer; var frac: Integer; var opt: Integer);</pre>
---------------	--

Liefert die Einstellungen für das aktuelle Zahlenformat der Werte an der x-Achse dieses Signals. Die Bedeutung der einzelnen Parameter ist im Kapitel "xSetUsing", S. 64 beschrieben.

Eine Übersicht aller möglichen Befehle erhalten Sie im Kapitel "Überblick", S. 59.

4.3.11 yMin

C++	<pre>__declspec(property(get=GetYMin,put=PutyMin)) double yMin;</pre>
------------	---

BASIC	Property yMin As Double
--------------	-------------------------

Delphi	<pre>property yMin: Double read Get_yMin write Set_yMin;</pre>
---------------	--

Liefert die untere Grenze des Messbereichs, in dem das Signal aufgezeichnet worden ist.

Eine Übersicht aller möglichen Befehle erhalten Sie im Kapitel "Überblick", S. 59.

4.3.12 yMax

C++	<code>__declspec(property(get=GetYMax,put=PutyMax)) double yMax;</code>
------------	---

BASIC	<code>Property yMax As Double</code>
--------------	--------------------------------------

Delphi	<code>property yMax: Double read Get_yMax write Set_yMax;</code>
---------------	--

Liefert die obere Messbereichsgrenze, in dem das Signal aufgezeichnet worden ist.

Eine Übersicht aller möglichen Befehle erhalten Sie im Kapitel "Überblick", S. 59.

4.3.13 yDefaultMin

C++	<code>__declspec(property(get=GetYDefaultMin,put=PutyDefaultMin)) double yDefaultMin;</code>
------------	--

BASIC	<code>Property yDefaultMin As Double</code>
--------------	---

Delphi	<code>property yDefaultMin: Double read Get_yDefaultMin write Set_yDefaultMin;</code>
---------------	---

Liefert die untere Grenze des Darstellungsbereichs, in dem das Signal standardmäßig gezeichnet werden soll.

Eine Übersicht aller möglichen Befehle erhalten Sie im Kapitel "Überblick", S. 59.

4.3.14 yDefaultMax

C++	<code>__declspec(property(get=GetYDefaultMax,put=PutyDefaultMax)) double yDefaultMax;</code>
------------	--

BASIC	<code>Property yDefaultMax As Double</code>
--------------	---

Delphi	<code>property yDefaultMax: Double read Get_yDefaultMax write Set_yDefaultMax;</code>
---------------	---

Liefert die obere Grenze des Darstellungsbereichs, in dem das Signal standardmäßig gezeichnet werden soll.

Eine Übersicht aller möglichen Befehle erhalten Sie im Kapitel "Überblick", S. 59.

4.3.15 yDelta

C++	<code>__declspec(property(get=GetYDelta,put=PutyDelta)) double yDelta;</code>
------------	---

BASIC	<code>Property yDelta As Double</code>
--------------	--

Delphi	<code>property yDelta: Double read Get_yDelta write Set_yDelta;</code>
---------------	--

Liefert die Auflösung der Messwerte des Signals.

Eine Übersicht aller möglichen Befehle erhalten Sie im Kapitel "Überblick", S. 59.

4.3.16 yUnit

C++	<code>__declspec(property(get=GetYUnit,put=PutyUnit)) _bstr_t yUnit;</code>
------------	---

BASIC	<code>Property yUnit As String</code>
--------------	---------------------------------------

Delphi	<code>property yUnit: WideString read Get_yUnit write Set_yUnit;</code>
---------------	---

Liefert die Einheit der y-Achse.

Eine Übersicht aller möglichen Befehle erhalten Sie im Kapitel "Überblick", S. 59.

4.3.17 ySetUsing

C++	<code>HRESULT ySetUsing(long format, long width, long frac, long opt);</code>
------------	---

BASIC	<code>Sub ySetUsing(format As Long, width As Long, frac As Long, opt As Long)</code>
--------------	--

Delphi	<code>procedure ySetUsing(format: Integer; width: Integer; frac: Integer; opt: Integer);</code>
---------------	---

Legt das Zahlenformat für die Ausgabe der Messwerte des Signals fest. Dabei legt **format** das Ausgabeformat fest, **width** die Breite des Zahlenfeldes und **frac** die Anzahl Ziffern nach der Kommastelle. Das Argument **opt** wird nur für die feste wissenschaftliche Darstellung verwendet und gibt dort die verwendete Zehnerpotenz an (siehe nachfolgende Tabelle).

Für **format** können folgende Zahlenwerte übergeben werden, alle anderen führen zu dem Fehlercode **E_INVALIDARG**:

Wert	Bedeutung	Beispiel: 17336,78
0	gibt die Messwerte dezimal aus	17336
3	Der Messwert wird ohne Angabe einer Zehnerpotenz mit der Anzahl der Kommastellen laut frac ausgegeben.	17336,780
4	Messwerte werden unter Angabe der Zehnerpotenz in der Schreibweise E+xxx ausgegeben.	1,734E+004
5	Wissenschaftliche Darstellung: Die Zehnerpotenz des Messwerts wird mit wissenschaftlichen Kürzeln dargestellt p (10 ⁻¹²), n (10 ⁻⁹), μ (10 ⁻⁶), m (10 ⁻³), k (10 ³), M (10 ⁶), G (10 ⁹)	17,337k
6	Feste wissenschaftliche Darstellung: In diesem Fall wird die Zehnerpotenz fest über den Parameter opt vorgegeben. Dabei können für opt folgende Werte vorkommen: 0: p (10 ⁻¹²) 3: m (10 ⁻³) 6: M (10 ⁶) 1: n (10 ⁻⁹) 4: (10 ⁰) 7: G (10 ⁹) 2: μ (10 ⁻⁶) 5: k (10 ³)	0,017M

Eine Übersicht aller möglichen Befehle erhalten Sie im Kapitel "Überblick", S. 59.

4.3.18 yGetUsing

```
C++      HRESULT yGetUsing(long *format,
                    long *width,
                    long *frac,
                    long *opt);
```

```
BASIC    Sub yGetUsing(format As Long,
                    width As Long,
                    frac As Long,
                    opt As Long)
```

Delphi	<pre>procedure yGetUsing(var format: Integer; var width: Integer; var frac: Integer; var opt: Integer);</pre>
---------------	---

Liefert die Einstellungen für das aktuelle Zahlenformat der Messwerte des Signals. Die Bedeutung der einzelnen Parameter ist im Kapitel "ySetUsing", S. 69 beschrieben.

Eine Übersicht aller möglichen Befehle erhalten Sie im Kapitel "Überblick", S. 59.

4.3.19 ScanStart

C++	<pre>__declspec(property(get=GetScanStart,put=PutScanStart)) double ScanStart;</pre>
------------	--

BASIC	Property ScanStart As Double
--------------	------------------------------

Delphi	<pre>property ScanStart: Double read Get_ScanStart write Set_ScanStart;</pre>
---------------	---

Liefert das Datum des Starts der Messung (d. h. den Zeitpunkt, zu dem der erste Messwert des Signals aufgenommen worden ist). Das Datum wird in Sekunden seit dem 1. Januar 1970 übergeben.

Eine Übersicht aller möglichen Befehle erhalten Sie im Kapitel "Überblick", S. 59.

4.3.20 SampleCount

C++	<code>long SampleCount();</code>
------------	----------------------------------

BASIC	<code>Function SampleCount() As Long</code>
--------------	---

Delphi	<code>function SampleCount: Integer;</code>
---------------	---

Liefert die Anzahl der Messwerte des Signals.

Eine Übersicht aller möglichen Befehle erhalten Sie im Kapitel "Überblick", S. 59.

4.3.21 ScaleX

C++	<code>HRESULT ScaleX(double xStart, double xEnd, long px);</code>
------------	---

BASIC	<code>Sub ScaleX(xStart As Double, xEnd As Double, px As Long)</code>
--------------	---

Delphi	<code>procedure ScaleX(xStart: Double; xEnd: Double; px: Integer);</code>
---------------	---

Skaliert den x-Bereich eines Signals so, dass von **GetNextScaled** die Messwerte zwischen **xStart** und **xEnd** übergeben werden. Dabei muss die Routine **GetNextScaled** **px**-mal aufgerufen werden, um den gesamten Kurvenzug zu zeichnen.

Eine Übersicht aller möglichen Befehle erhalten Sie im Kapitel "Überblick", S. 59.

4.3.22 ScaleY

C++	<pre>HRESULT ScaleY(double yMin, double yMax, long py);</pre>
------------	---

BASIC	<pre>Sub ScaleY(yMin As Double, yMax As Double, py As Long)</pre>
--------------	---

Delphi	<pre>procedure ScaleY(yMin: Double; yMax: Double; py: Integer);</pre>
---------------	---

Skaliert den y-Bereich eines Signals so, dass die Messwerte zwischen **yMin** und **yMax** auf die Integerwerte 0 bis **py** abgebildet werden.

Eine Übersicht aller möglichen Befehle erhalten Sie im Kapitel "Überblick", S. 59.

4.3.23 ResetDataPosition

C++	<pre>HRESULT ResetDataPosition();</pre>
------------	---

BASIC	<pre>Sub ResetDataPosition()</pre>
--------------	------------------------------------

Delphi	<pre>procedure ResetDataPosition;</pre>
---------------	---

Setzt den internen Zähler des Signals zurück, so dass der nächste Aufruf von **GetNextScaled** das erste Minimum/Maximum-Paar zurückliefert (bzw. **NextSample** liefert den ersten Messwert des Signals).

Eine Übersicht aller möglichen Befehle erhalten Sie im Kapitel "Überblick", S. 59.

4.3.24 GetNextScaled

C++	<code>VARIANT_BOOL GetNextScaled(long *min, long *max);</code>
------------	--

BASIC	<code>Function GetNextScaled(min As Long, max As Long) As Boolean</code>
--------------	--

Delphi	<code>function GetNextScaled(out min: Integer; out max: Integer): WordBool;</code>
---------------	--

Liefert das nächste Minimum/Maximum-Paar des Signals entsprechend den Skalierungen durch `ScaleX()` und `ScaleY()`.

Eine Übersicht aller möglichen Befehle erhalten Sie im Kapitel "Überblick", S. 59.

4.3.25 GetNextScaledDigital

C++	<code>VARIANT_BOOL GetNextScaledDigital(long *min, long *max);</code>
------------	---

BASIC	<code>Function GetNextScaledDigital(min As Long, max As Long) As Boolean</code>
--------------	---

Delphi	<code>function GetNextScaledDigital(out min: Integer; out max: Integer): WordBool;</code>
---------------	---

Liefert das nächste Minimum/Maximum-Paar des Signals entsprechend den Skalierungen durch `ScaleX()` als Digitalwert. Diese Funktion berücksichtigt die Einstellungen von `ScaleY()` nicht.

Eine Übersicht aller möglichen Befehle erhalten Sie im Kapitel "Überblick", S. 59.

4.3.26 Unscale

C++	<code>HRESULT Unscale();</code>
------------	---------------------------------

BASIC	<code>Sub Unscale()</code>
--------------	----------------------------

Delphi	<code>procedure Unscale;</code>
---------------	---------------------------------

Schaltet die Skalierung des Signals ab, so dass alle Messwerte des Signals mit Hilfe der Funktion **NextSample()** abgefragt werden können.

Eine Übersicht aller möglichen Befehle erhalten Sie im Kapitel "Überblick", S. 59.

4.3.27 NextSample

C++	<code>__declspec(property(get=GetNextSample,put=PutNextSample)) double NextSample;</code>
------------	---

BASIC	<code>Property NextSample As Double</code>
--------------	--

Delphi	<code>property NextSample: Double read Get_NextSample write Set_NextSample</code>
---------------	---

Liefert den nächsten Messwert des Signals. Diese Funktion liefert nur dann sinnvolle Werte, wenn die Skalierung des Signals vorher mit **Unscale()** abgeschaltet wurde.

Eine Übersicht aller möglichen Befehle erhalten Sie im Kapitel "Überblick", S. 59.

4.3.28 NextDigitalSample

C++	<code>__declspec(property(get=GetNextDigitalSample, put=PutNextDigitalSample)) long NextDigitalSample;</code>
------------	---

BASIC	<code>Property NextDigitalSample As Long</code>
--------------	---

Delphi	<code>property NextDigitalSample: Long read Get_NextDigitalSample write Set_NextDigitalSample</code>
---------------	--

Liefert den nächsten Wert eines Digitalsignals.

Eine Übersicht aller möglichen Befehle erhalten Sie im Kapitel "Überblick", S. 59.

4.3.29 GetSampleAt

C++	<code>double GetSampleAt(double time);</code>
------------	---

BASIC	<code>Function GetSampleAt(time As Double) As Double</code>
--------------	---

Delphi	<code>function GetSampleAt(time: Double): Double;</code>
---------------	--

Liefert einen Messwert zu einer bestimmten Zeit innerhalb des Signals.

Eine Übersicht aller möglichen Befehle erhalten Sie im Kapitel "Überblick", S. 59.

4.3.30 GetSampleAtOffset

C++	<code>double GetSampleAtOffset(long offset);</code>
------------	---

BASIC	<code>Function GetSampleAtOffset(offset As Long) As Double</code>
--------------	---

Delphi	<code>function GetSampleAt(offset: Integer): Double;</code>
---------------	---

Liefert einen Messwert an einem bestimmten Offset innerhalb des Signals. Dabei muss **offset** innerhalb von 0 und **sampleCount** liegen.

Eine Übersicht aller möglichen Befehle erhalten Sie im Kapitel "Überblick", S. 59.

4.3.31 IsAnalog

C++	<code>VARIANT_BOOL IsAnalog();</code>
------------	---------------------------------------

BASIC	<code>Function IsAnalog() As Boolean</code>
--------------	---

Delphi	<code>function IsAnalog: WordBool;</code>
---------------	---

Liefert **TRUE** zurück, wenn im Signal analoge Werte gespeichert sind.

Eine Übersicht aller möglichen Befehle erhalten Sie im Kapitel "Überblick", S. 59.

4.3.32 IsDigital

C++	<code>VARIANT_BOOL IsDigital();</code>
------------	--

BASIC	<code>Function IsDigital() As Boolean</code>
--------------	--

Delphi	<code>function IsDigital: WordBool;</code>
---------------	--

Liefert **TRUE** zurück, wenn im Signal digitale Werte gespeichert sind.

Eine Übersicht aller möglichen Befehle erhalten Sie im Kapitel "Überblick", S. 59.

5 Index

3

32-Bit 7

6

64-Bit 7

A

AboutBox 56

Abtaste 44

Abtastzeit 63

ActiveX Control 11

Analogausgang
aktueller Wert 51

Analogeingang
aktueller Wert 51

AnalogIn 51

AnalogOut 51

Anzahl der Messwerte 44, 72

Anzeigebereich
obere Grenze 68
untere Grenze 67

Auflösung 68

Ausgabebereich 26

B

Beispielprogramme 16, 18, 23

BMCSAD 16, 18

C

C++[®] 7

Close 42, 58

Comment 62

Create 57

D

Datum 71

Delphi[®] 7, 17

Digitalausgang
aktueller Wert 53

Digitalausgangsleitung
aktueller Wert 54

DigitalDirection 54

Digitaleingang
aktueller Wert 52

Digitaleingangsleitung
aktueller Wert 53

DigitalIn 52

DigitalInLine 53

Digitalkanal
Richtung 54

DigitalOut 53

DigitalOutLine 54

Digitalport
Richtung 54

Digitalsignal
nächster Wert 76

E

E_INVALIDARG 65, 70

Einbindung in Programmiersprachen 11

Einheit
x-Achse 64
y-Achse 69

F

Fehlermeldung 43

Fehlernummer 43

FileCreate 50

FileCreateAnalogIn 49

FileCreateDigital 49

FileCreatePrepare 48

FileOpen 47

G

Gerätekonflikte 10

Geräte-Manager 10

GetNextScaled 74

GetNextScaledDigital 74
 GetSampleAt 76
 GetSampleAtOffset 77
 GetVersion 42
 Grenze
 oben 67, 68
 unten 66, 67
 Groß-/Kleinschreibung 26, 41
 GroupName 61
 Grundlagen 25
 Gruppenname 61

I

iM-3250 27
 iM-3250T 27
 iM-AD25 27
 iM-AD25a 27
 Installation 10, 12
 Internetadresse 8
 INvxFile 56
 INvxSignal 59
 IsAnalog 77
 IsDigital 78

K

Kanalliste 44, 46, 48, 49, 50
 Analogkanal hinzufügen 45
 Digitalkanal hinzufügen 46
 Reihenfolge der Kanäle 45, 46
 Zähler hinzufügen 45
 Kanalnummer 26
 Kommentar 62

L

LAN-AD16f 28
 Digitalports 28
 Zähler 28
 LastError 43
 LastErrorString 43
 LIBAD4 25
 LibadX 40

M

MAD12 30
 MAD12a 30
 MAD12b 30
 MAD12f 30
 MAD16 30
 MAD16a 30
 MAD16b 30
 MAD16f 30
 MADDA16 30
 MADDA16n 30
 Maximum 74
 MDA12 31
 MDA12-4 31
 MDA16 31
 MDA16-2i 31
 MDA16-4i 31
 MDA16-8i 31
 meM-AD 32
 meM-ADDA 32
 meM-ADf 32
 meM-ADfo 32
 meM-Geräte
 Digitalports 32, 33
 Reihenfolge 32, 33
 Seriennummer 32, 33
 meM-PIO 33
 meM-PIO-OEM 33
 Messbereich 26
 obere Grenze 67
 untere Grenze 66
 Messdatei
 Analogeingang hinzufügen 49
 Anzahl Signale 58
 Digitalkanal hinzufügen 49
 erzeugen 48, 50, 57
 öffnen 47, 57
 schließen 58
 Signal ausgeben 59
 vorbereiten 48
 Zähler hinzufügen 49
 Messsystem
 öffnen 26, 41
 schließen 26, 42
 Messwert
 abfragen 76
 an Offset abfragen 77

Minimum 74

N

nächster Digitalwert 76

nächster Wert 75

Name 61

NextDigitalSample 76

NextSample 75

NextView@4 25

O

Offset 77

Open 26, 41, 57

P

PCI-BASE1000 29

Digitalports 29

PCI-BASE300 29

Digitalports 29

PCI-BASEII 29

Digitalports 29

PCIe-BASE 29

Digitalports 29

PCIe-Karten

Seriennummer 29

PCI-Karten

Seriennummer 29

PCI-PIO 29

Digitalports 29

R

ResetDataPosition 73

Richtung 54

S

Sample 55

SampleCount 72

ScaleX 72

ScaleY 73

Scan 46

speichern 47

starten 46

vorbereiten 44

ScanAnalogIn 45

ScanDigitalIn 46

ScanPrepare 44

ScanSave 47

Scanstart 44

Datum 71

Zeit 62

ScanStart 71

Schnittstelle

INvxFile 56

INvxSignal 59

LibadX 40

Seriennummer 29, 32, 33, 34, 36, 37, 38

Signal 59

analog 77

Anzahl der Messwerte 72

Datenposition zurücksetzen 73

digital 78

nächster Wert 75

SignalCount 58

Signaldauer 63

Signalende 63

Signalname 61

Signalstart 62

Skalierung 72, 73

ausschalten 75

Software Collection-CD 10, 11, 12, 16,

18, 23

Speicherplatz 13

Standardverzeichnis 13

T

Trigger 62

U

Unscale 75

Urheberrechte 9

USB-AD 34

Digitalports 34

Reihenfolge 34

Seriennummer 34

USB-AD12f 36

Digitalports 36

Reihenfolge 36

Seriennummer 36

- Zähler 36
- USB-AD16f 37
 - Digitalports 37
 - Reihenfolge 37
 - Seriennummer 37
 - Zähler 37

- USB-PIO 38
 - Digitalports 38
 - Reihenfolge 38
 - Seriennummer 38
- USB-PIO-OEM 38
 - Digitalports 38
 - Reihenfolge 38
 - Seriennummer 38

V

- VB .NET 21
- Version 42
- Visual Basic® 7, 15
- Visual Basic® .NET 7
- Visual C#® 7, 20
- Visual C++® 7, 19
- Vorgeschichte 62

X

- x-Achse
 - Einheit 64
 - Skalierung 72
 - Zahlenformat 64
- xDelta 63

- xEnd 63
- xGetUsing 66
- xSetUsing 64
- xStart 62
- xUnit 64

Y

- y-Achse
 - Einheit 69
 - Skalierung 73
 - Zahlenformat 69, 71
- yDefaultMax 68
- yDefaultMin 67
- yDelta 68
- yGetUsing 71
- yMax 67
- yMin 66
- ySetUsing 69
- yUnit 69

Z

- Zahlenformat 66
 - x-Achse 64, 66
 - y-Achse 69, 71
- Zähler 36, 37, 49
 - aktueller Wert 51