bmcm

bmc messsysteme gmbh

LIBADX

Programmierschnittstelle (ActiveX Control) für
bmcm Messgerätetreiber

Installations- und Programmierhandbuch

Version 4.6

www.bmcm.de

bavarian measurement company munich

Inhaltsverzeichnis

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® XP/7/8 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 Beispielcodes 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 Entwicklungs-umgebungen 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.

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.

Stand: 09. Dezember 2014

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® 8**: Rechtsklick Bildschirmecke links unten (Tastatur "Windows+X") / Geräte-Manager
  - **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 beigefügten 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 das Installationsprogramm auf die Festplatte und starten Sie dieses dann 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 bmcm Hardware durch Auswahl des Eintrags "Produkte" und klicken Sie auf das von Ihnen verwendete Messsystem. Zur Programmierung unter Windows® XP/7/8 wählen Sie im Bereich "API (Programmierung)" die Produktbezeichnung "STR-LIBADX" um die Installation zu starten.

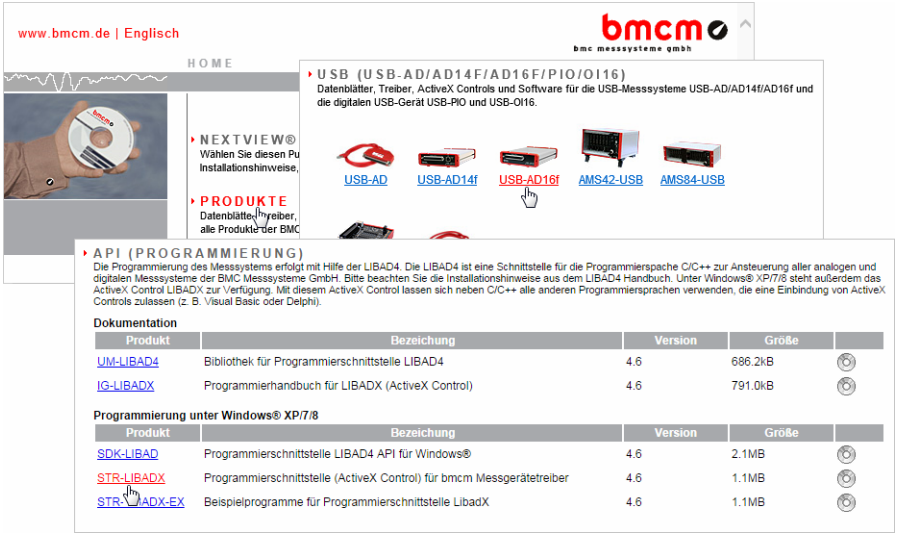

Abbildung 1

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.

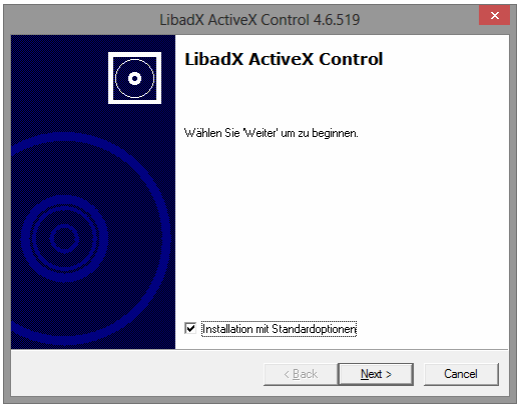

Abbildung 2

Sie werden schrittweise durch die Installation geführt. Die Schaltfläche "Weiter" führt in den nächsten Dialog, mit "Zurück" gehen Sie einen Schritt zurück. Die Installation kann mit "Abbrechen" stets vorzeitig beendet werden.

Wenn Sie die Installation nicht mit Standardoptionen durchführen wollen, deaktivieren Sie diese Option im ersten Fenster des Installationsprogramms und bestimmen, wohin **LibadX** installiert wird.

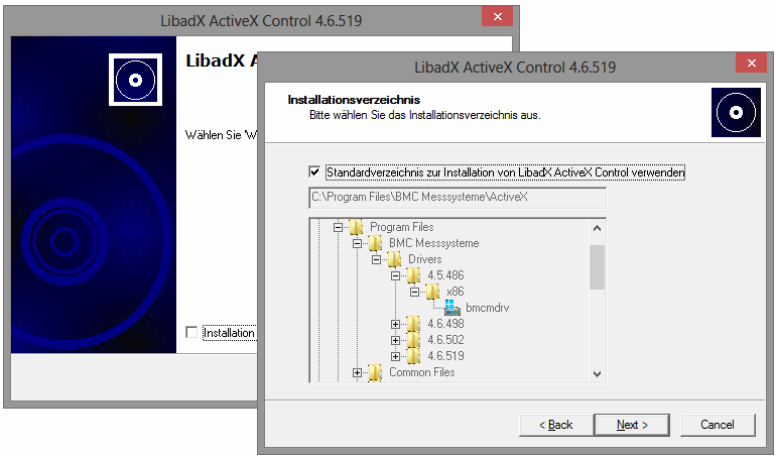

Abbildung 3

Das vorgeschlagene Standardverzeichnis lässt sich nach Belieben ändern. Zur Auswahl eines anderen Installationsverzeichnisses klickt man 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.

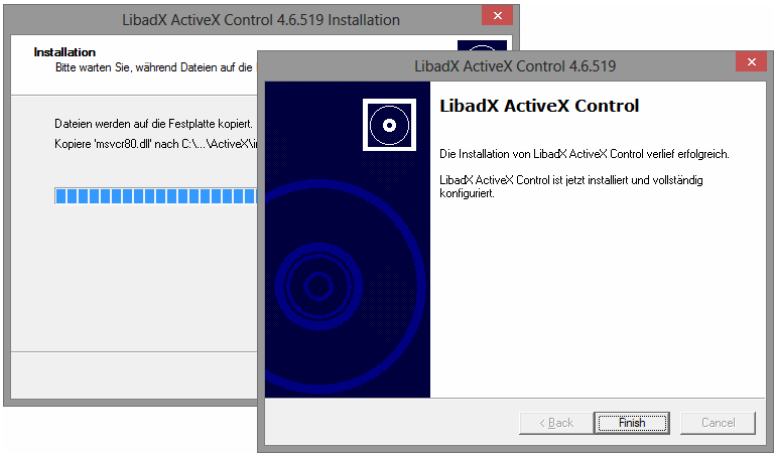

Abbildung 4

2.3 Einbindung in Programmiersprachen

2.3.1 Anbindung an Visual Basic® 4.0 - 6.0

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.

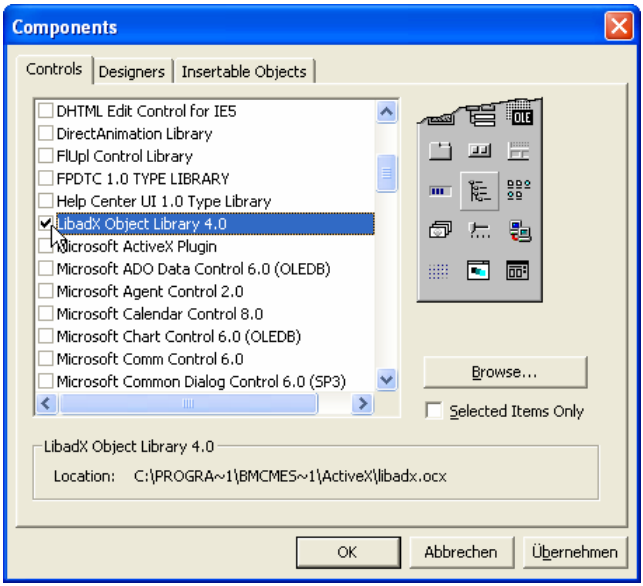

Abbildung 5

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 6

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()
            LIBADX1.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

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

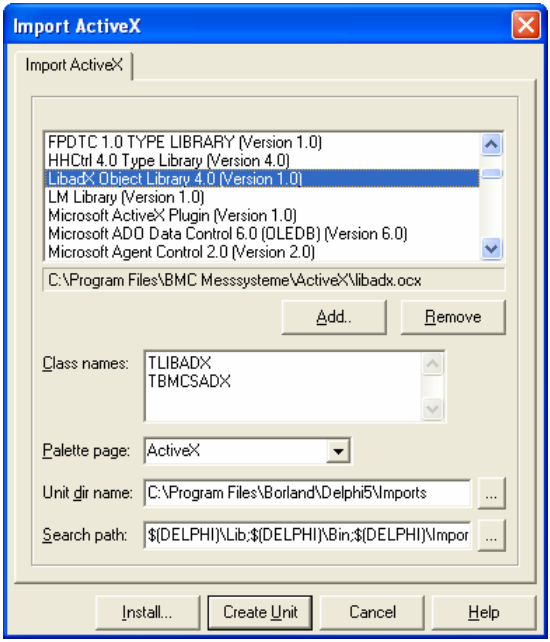

Abbildung 7

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 8

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

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  LibadX.AboutBox ();
end;
```

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.

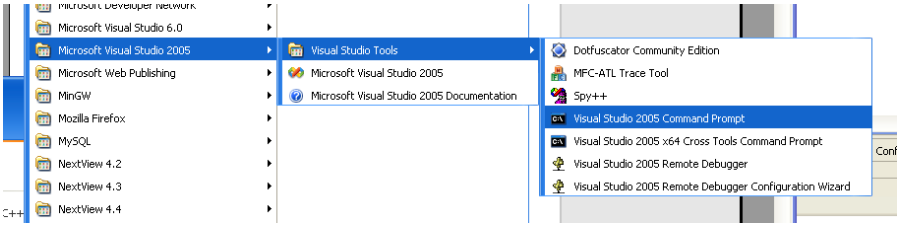

Abbildung 9

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 **`libadxTyleLib.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 Beispielprogrammen 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®)

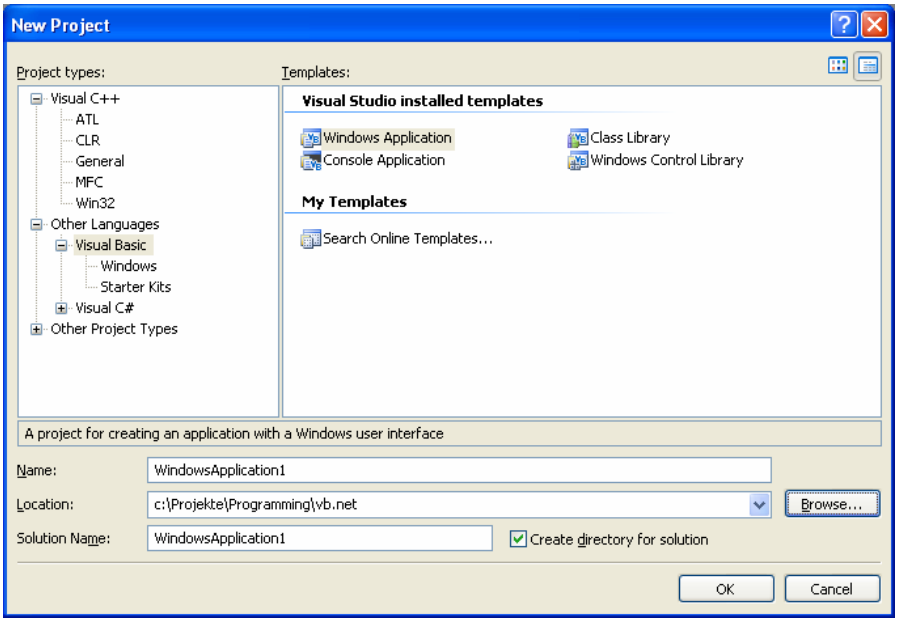

Abbildung 10

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 10).

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.

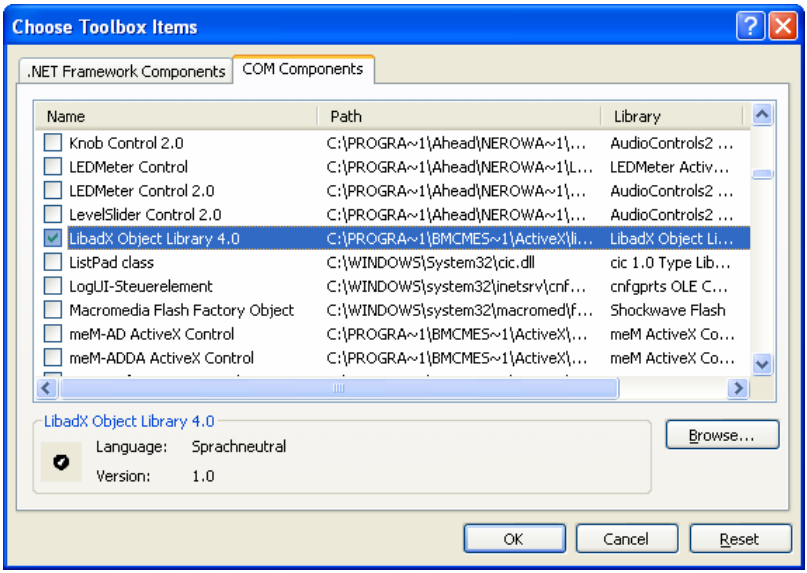

Abbildung 11

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.

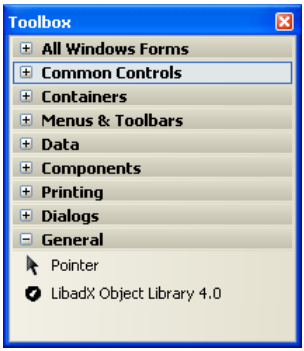

Abbildung 12

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® XP/7/8", der unter "API (Programmierung)" auf der Produktseite aufgelistet ist.

Abbildung 13

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

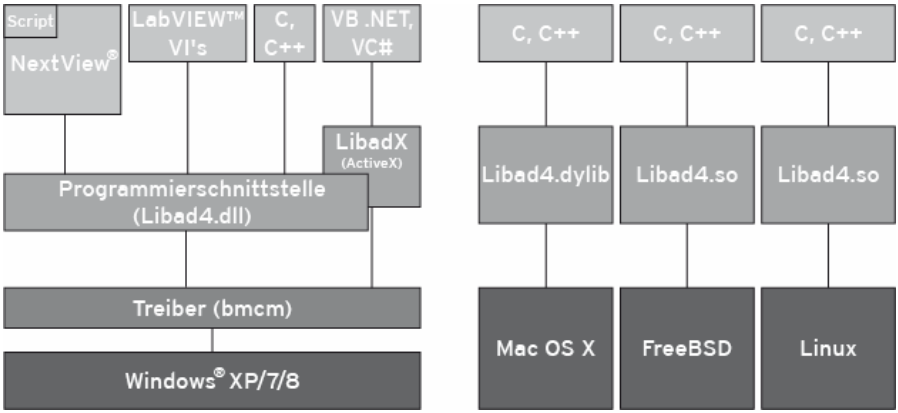

Abbildung 14

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 (LIBADX1.Open ("usb-pio"))
   ...
   LIBADX1.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 (LIBADX1.Open ("pcibase")
    AND LIBADX2.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-AD14f / 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	±10.24V ±5.12V	1 0	1: Ausgang (Bit 0..3)
iM-AD25	16 Eingänge	1..16	±5.12V	0	1: Ausgang (Bit 0..3)
iM3250T	32 Eingänge	17..48	±5.12V	0	-
iM3250	32 Eingänge	AnIn 1..16: 1..16 (bei 1 BPL) 17..32 (bei 2 BPL) AnIn 17..32: 33..48	±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-AD16fx / LAN-AD16f

Um ein LAN-Messsystem vom Typ LAN-AD16f(x) (auch: AMS42/84-LAN16f, AMS42/84-LAN16fx) 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(x). 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.

Mess-system	Analog	Kanal-nummer	range (Messber.)	range (Aus-gabebereich)	Digital	Richtung
LAN-AD16fx	16 Eingänge 2 Ausgänge	1..16 1 .. 2	0 (±1.024V) 1 (±2.048V) 2 (±5.12V) 3 (±10.24V)	0 (±10.24V)	2 Ports (je 16 Bit)	1: Port A 2: Port B
LAN-AD16f	16 Eingänge 2 Ausgänge	1..16 1 .. 2	0 (±1.024V) 1 (±2.048V) 2 (±5.12V) 3 (±10.24V)	0 (±10.24V)	2 Ports (je 16 Bit)	1: Eingang (Bit 0..15) 2: Ausgang (Bit 0..15)

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

Das LAN-AD16f(x) stellt zwei 16-Bit Digitalports zur Verfügung. Beim LAN-AD16fx ist die Richtung der digitalen Portleitungen in 8er Gruppen umschaltbar (s. "**`DigitalDirection`**", S. 55), beim LAN-AD16f sind die Leitungen hingegen fest verdrahtet. Nach dem Einschalten des LAN-AD16f(x) stehen die 16 Leitungen des ersten Ports (DIO1, Kanalnummer: 1) auf Eingang, die 16 Leitungen des zweiten Ports auf Ausgang (DIO2, Kanalnummer: 2).

Die Zähler des LAN-AD16f(x) sind 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 Kartennummer 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 (s. "**`DigitalDirection`**", S. 55). 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 (PCIe-BASE, PCI-BASEII, PCI-PIO) mit drei 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) 8 Eingänge (differentiell)	1..16 (se) 17..24 (diff)	±1.024V ±2.048V ±5.120V ±10.240V 0.06V..5.06V	0 1 2 3 4
MAD12a, MAD12f, MAD16a, MAD16f	16 Eingänge (single-ended) 8 Eingänge (differentiell)	1..16 (se) 17..24 (diff)	±1.024V ±2.048V ±5.120V ±10.240V	0 1 2 3
MAD12b, MAD16b	16 Eingänge (single-ended)	1..16	±1.024V ±2.048V ±5.120V ±10.240V	0 1 2 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 2 Ausgänge	1..16 1..2	0 (±1.024V) 1 (±2.048V) 2 (±5.120V) 3 (±10.240V)	0 (±10.24V)

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	±10.24V ±5.12V	0 1
MDA12-4	4 Ausgänge	1..4	±10.24V ±5.12V	0 1
MDA16-2i	2 Ausgänge	1..2	±10.24V	0
MDA16-4i	4 Ausgänge	1..4	±10.24V	0
MDA16-8i	8 Ausgänge	1..8	±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.

Mess-system	Analog	Kanal-nummer	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. 55). 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.

Mess-system	Analog	Kanal-nummer	range (Messber.)	range (Aus-gabebereich)	Digital	Richtung
USB-AD	16 Eingänge 1 Ausgang	1..16 1	0 (±5.12V)	0 (±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 Kompatibililtätsgründen 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 LIBADX1.Open("usb-ad:0") Then

    Dim tmp As Integer
    tmp = LIBADX1.DigitalIn(1)

    Dim bool As Boolean
    ' liest den Zustand der ersten Leitung des Ports 1
    bool = LIBADX1.DigitalInLine(1, 0)

    ' alle Leitungen erstmal löschen
    LIBADX1.DigitalOut(2) = 0
    ' Leitung 2 high
    LIBADX1.DigitalOutLine(2, 1) = True

    Dim val As Double
    ' liest den Analog Wert vom Analog In 1 im Messbereich 0
    val = LIBADX1.AnalogIn(1, 0)

    ' setze den Analog Out 1 auf 4.5 Volt
    LIBADX1.AnalogOut(1, 0) = 4.5

    LIBADX1.Close
  End If
```

3.2.8 USB-AD14f / USB-AD12f

Um ein USB-AD14f / USB-AD12f mit der **LibadX** zu öffnen, muss an `Open()` der String "`usbad14f`" bzw. "`usbad12f`" übergeben werden. Mehrere USB Messsysteme lassen sich durch Angabe der Gerätenummer öffnen (1. USB-AD14f mit "`usbad14f:0`", 2. USB-AD14f mit "`usbad14f: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-AD14f an und entfernt dann das 2. Gerät, sind die verbleibenden USB-AD14f mit "`usbad14f:0`" und "`usbad14f:2`" anzusprechen.

Um unabhängig von dieser Ansteckreihenfolge zu sein, kann ein Gerät auch mit einer bestimmten Seriennummer geöffnet werden. Das USB-AD14f mit der Seriennummer 157 lässt sich zum Beispiel mit "`usbad14f:@157`" ansprechen.

Mess-system	Analog	Kanal-nummer	range (Messber.)	range (Aus-gabebereich)	Digital	Richtung
USB-AD14f	16 Eingänge 1 Ausgang	1..16 1	0 (±10.24V)	0 (±5.12V)	2 Ports (je 8 Bit)	1: Eingang (Bit 0..7) 2: Ausgang (Bit 0..7)
USB-AD12f	16 Eingänge 1 Ausgang	1..16 1	0 (±10.24V)	0 (±5.12V)	2 Ports (je 4 Bit)	1: Eingang (Bit 0..3) 2: Ausgang (Bit 0..3)

Die 16 Analogeingänge eines USB-AD14f / USB-AD12f besitzen die Kanalnummern 1-16. Der Analogausgang hat die Kanalnummer 1.

Die Richtung der Digitalports ist nicht umschaltbar. Dabei stehen die 8 (USB-AD14f) bzw. 4 (USB-AD12f) Leitungen des ersten Ports (DIO1, Kanalnummer: 1) auf Eingang, die 8 (USB-AD14f) bzw. 4 (USB-AD12f) 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. 52), `AnalogOut` (s. S. 52) und `ScanAnalogIn` (s. S. 46) 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 (auch: AMS42/84-USB) 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.

Mess-system	Analog	Kanal-nummer	range (Messber.)	range (Ausgabebereich)	Digital	Richtung
USB-AD16f	16 Eingänge 2 Ausgänge	1..16 1 .. 2	0 (±1.024V) 1 (±2.048V) 2 (±5.12V) 3 (±10.24V)	0 (±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. 52), **AnalogOut** (s. S. 52) und

`ScanAnalogIn` (s. S. 46) 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-OI16

Um eine USB-OI16 mit der **LibadX** zu öffnen, muss an **`Open()`** der String "**`usb-oi16`**" übergeben werden. Mehrere USB-OI16 Geräte lassen sich durch Angabe der Gerätenummer öffnen (1. Gerät mit "**`usb-oi16:0`**", 2. Gerät mit "**`usb-oi16:1`**", usw.). Die Reihenfolge der Geräte wird durch das Anstecken bestimmt.

Da USB Geräte im laufenden Betrieb an- und wieder abgesteckt werden können, ist es möglich, dass die Gerätenummern nicht in aufsteigender Reihenfolge vergeben sind. Steckt man z. B. drei USB-OI16 an und entfernt dann das 2. Gerät, sind die verbleibenden USB-OI16 mit "**`usb-oi16:0`**" und "**`usb-oi16:2`**" 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-oi16:@157`**" ansprechen.

Messsystem	Digital	Kanalnummer
USB-OI16	2 Ports (je 16 Bit)	1: Eingang 2: Ausgang

Die USB-OI16 stellt zwei 16-Bit Digitalports zur Verfügung. Die Richtung der digitalen Portleitungen ist nicht umschaltbar. Dabei stehen die 16 Leitungen des ersten Ports (DIO1) auf Eingang, die 16 Leitungen des zweiten Ports (DIO2) auf Ausgang.

Die Zähler der USB-OI16 sind nur mit dem LIBAD4 SDK programmierbar.

3.2.11 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 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 "**`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. 55). 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 LIBADX1.Open("usb-pio:0") Then
  LIBADX1.DigitalDirection(1) = &H0   ' alle Output
  LIBADX1.DigitalDirection(2) = &HFF  ' alle Input
  LIBADX1.DigitalDirection(3) = &H0   ' alle Output

  Dim tmp As Integer
  ' Liest den Zustand aller Leitungen des Ports 2
  tmp = LIBADX1.DigitalIn(2)

  Dim bool As Boolean
  ' liest den Zustand der ersten Leitung des Ports 2
  bool = LIBADX1.DigitalInLine(2, 0)

  ' alle Leitungen erst mal löschen
  LIBADX1.DigitalOut(1) = 0
  ' Leitung 8 des Ports 1 high
  LIBADX1.DigitalOutLine(1, 7) = True

  ' Port 3 auf &H15 setzen = Leitung 1, 3, 5 high
  LIBADX1.DigitalOut(3) = &H15

  LIBADX1.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 letzen Fehlercode zurück
`LastErrorString`	liefert eine Beschreibung des letzen 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 Dateiobjekt 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++	`VARIANT_BOOL Open (_bstr_t path)`

BASIC	`Function Open (path As String) As Boolean`

Delphi	`function Open (const path: WideString): WordBool`

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. 41.

4.1.3 Close

C++	`HRESULT Close ()`
BASIC	`Sub Close ()`
Delphi	`procedure Close`

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

Eine Übersicht aller möglichen Befehle erhalten Sie im Kapitel "Überblick", S. 41.

4.1.4 GetVersion

C++	`long GetVersion ()`
BASIC	`Function GetVersion () As Long`
Delphi	`function GetVersion: Integer`

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. 41.

4.1.5 LastError

C++	`long LastError ()`
BASIC	`Function LastError () As Long`
Delphi	`function LastError: Integer`

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. 41.

4.1.6 LastErrorString

C++	`_bstr_t LastErrorString ()`
BASIC	`Function LastErrorString () As String`
Delphi	`function LastErrorString: WideString`

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. 41.

4.1.7 ScanPrepare

C++	`HRESULT ScanPrepare (float sample_rate, long samples)`

BASIC	`Sub ScanPrepare (sample_rate As Single, samples As Long)`

Delphi	`procedure ScanPrepare (sample_rate: Single; samples: Integer)`

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

```
' 1000 Messwerte, 100Hz (0.01 Sek.)
LIBADX1.ScanPrepare 0.01, 1000

' Kanal 1 & 2 speichern
LIBADX1.ScanAnalogIn 1, 0
LIBADX1.ScanAnalogIn 2, 0

' Zaehler 1 speichern
LIBADX1.ScanDigitalIn &h08000001

' Digital Port 1 speichern
LIBADX1.ScanDigitalIn 1

' Scan durchführen
LIBADX1.Scan

' Scan speichern
LIBADX1.ScanSave "scan.lfx"
```

Eine Übersicht aller möglichen Befehle erhalten Sie im Kapitel "Überblick", S. 41.

4.1.8 ScanAnalogIn

C++	`HRESULT ScanAnalogIn (long index, long range)`
BASIC	`Sub ScanAnalogIn (index as Long, range as Long)`
Delphi	`procedure ScanAnalogIn (index, range: Integer)`

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. 44) 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. 41.

4.1.9 ScanDigitalIn

C++	`HRESULT ScanDigitalIn (long index)`
BASIC	`Sub ScanDigitalIn (index as Long)`

Delphi	`procedure ScanDigitalIn (index: Integer)`

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. 44) 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. 41.

4.1.10 Scan

C++	`VARIANT_BOOL Scan ();`

BASIC	`Function Scan () As Boolean`

Delphi	`function Scan : WordBool`

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. 44) vorbereitet oder kein Kanal der Kanalliste hinzugefügt wurde.

Eine Übersicht aller möglichen Befehle erhalten Sie im Kapitel "Überblick", S. 41.

4.1.11 ScanSave

C++	`VARIANT_BOOL ScanSave (_bstr_t path);`
BASIC	`Function ScanSave (path As String) As Boolean`
Delphi	`function ScanSave (const path: WideString): WordBool`

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. 41.

4.1.12 FileOpen

C++	`INvxFilePtr FileOpen (_bstr_t path)`
BASIC	`Function FileOpen (path As String) As INvxFile`
Delphi	`function FileOpen (const path: WideString): INvxFile`

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. 41.

4.1.13 FileCreatePrepare

C++	`HRESULT FileCreatePrepare (long samples)`

BASIC	`Sub FileCreatePrepare (samples As Long)`

Delphi	`procedure FileCreatePrepare (samples: Integer)`

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

```
' 1000 Messwerte
LIBADX1.FileCreatePrepare 1000

' 2 analoge Kanaele
LIBADX1.FileCreateAnalogIn
LIBADX1.FileCreateAnalogIn

' 1 Zaehlerkanal
Const AD_CHA_TYPE_COUNTER as Integer = &h08000000
LIBADX1.FileCreateDigital AD_CHA_TYPE_COUNTER

' 1 Digitalkanal mit 16 Leitungen
LIBADX1.FileCreateDigital 16

' Datei anlegen
LIBADX1.FileCreate "scan.lfx"
```

Eine Übersicht aller möglichen Befehle erhalten Sie im Kapitel "Überblick", S. 41.

4.1.14 FileCreateAnalogIn

C++	`long FileCreateAnalogIn ()`
BASIC	`Function FileCreateAnalogIn () As Long`
Delphi	`function FileCreateAnalogIn: Integer;`

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. 48) vorbereitet wurde.

Eine Übersicht aller möglichen Befehle erhalten Sie im Kapitel "Überblick", S. 41.

4.1.15 FileCreateDigital

C++	`long FileCreateDigital (long lines)`
BASIC	`Function FileCreateDigital (lines As Long) As Long`
Delphi	`function FileCreateDigital(lines: Integer): Integer;`

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. 60) "yMax" (s. S. 68) und "yMin" (s. S. 67) unbedingt richtig gesetzt werden, da sonst diese nicht richtig geschrieben werden. Das y-Using (s. Kap. "ySetUsing", S. 70) sollte auch entsprechend angepasst werden.

Die Funktion löst eine Exception aus, wenn vorher keine Messdatei mit `FileCreatePrepare()` (s. S. 48) vorbereitet wurde.

Eine Übersicht aller möglichen Befehle erhalten Sie im Kapitel "Überblick", S. 41.

4.1.16 FileCreate

C++	`INvxFilePtr FileCreate (_bstr_t path)`
BASIC	`Function FileCreate (path As String) As INvxFile`
Delphi	`function FileCreate (const path: WideString): INvxFile`

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. 48) vorbereitet oder kein Kanal der Kanalliste hinzugefügt wurde.

Eine Übersicht aller möglichen Befehle erhalten Sie im Kapitel "Überblick", S. 41.

4.1.17 AnalogIn

C++	`__declspec(property(get=GetAnalogIn))` `  float AnalogIn[][]`
BASIC	`Property AnalogIn (index As Long, range as Long) As Single`
Delphi	`property AnalogIn [index, range: Integer]: Single readonly`

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. 41.

4.1.18 AnalogOut

C++	`__declspec(property(get=GetAnalogOut,put=PutAnalogOut))` `  float AnalogOut[][]`
BASIC	`Property AnalogOut (index As Long, range as Long) As Single`
Delphi	`property AnalogOut [index, range: Integer]: Single`

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. 41.

4.1.19 DigitalIn

C++	`__declspec(property(get=GetDigitalIn))` `  long DigitalIn[]`

BASIC	`Property DigitalIn (index As Long) As Long`

Delphi	`property DigitalIn [index: Integer]: Integer readonly`

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. 41.

4.1.20 DigitalOut

C++	`__declspec(property(get=GetDigitalOut,put=PutDigitalOut))` `  long DigitalOut[];`
BASIC	`Property DigitalOut (index As Long) As Long`
Delphi	`property DigitalOut [index: Integer]: Integer`

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. 41.

4.1.21 DigitalInLine

C++	`__declspec(property(get=GetDigitalInLine))` `  VARIANT_BOOL DigitalInLine[][];`
BASIC	`Property DigitalInLine (index As Long, line As Long)` `  As Boolean`
Delphi	`property DigitalInLine [index, line: Integer]:` `  WordBool readonly`

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. 41.

4.1.22 DigitalOutLine

C++	`__declspec(property(get=GetDigitalOutLine, put=PutDigitalOutLine)) VARIANT_BOOL DigitalOutLine[][];`
BASIC	`Property DigitalOutLine (index As Long, line As Long) As Boolean`
Delphi	`property DigitalOutLine [index, line: Integer]: WordBool`

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. 41.

4.1.23 DigitalDirection

C++	`__declspec(property(get=GetDigitalDirection, put=PutDigitalDirection)) long DigitalDirection[];`
BASIC	`Property DigitalDirection (index As Long) As Long`
Delphi	`property DigitalDirection [index: Integer]: Integer`

Legt die Ein-/Ausgaberichtung 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. 41.

4.1.24 Sample

C++	`__declspec(property(get=GetSample,put=PutSample))` `  float Sample[][]`
BASIC	`Property Sample (index As Long, pos As Long) As Single`
Delphi	`property Sample [index, pos: Integer]: Single`

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. 41.

4.1.25 AboutBox

C++	`HRESULT AboutBox ()`
BASIC	`Sub AboutBox ()`
Delphi	`procedure AboutBox`

Zeigt die AboutBox von **LibadX**.

Eine Übersicht aller möglichen Befehle erhalten Sie im Kapitel "Überblick", S. 41.

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++	`HRESULT Open(_bstr_t fileName);`
BASIC	`Sub Open(fileName As String)`
Delphi	`procedure Open(const fileName: WideString);`

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. 57.

4.2.3 Create

C++	`HRESULT Create(_bstr_t fileName,`
	`long signalCount,`
	`long sampleCount);`

BASIC	`Sub Create(fileName As String,`
	`signalCount As Long,`
	`sampleCount As Long)`

Delphi	`procedure Create(const fileName: WideString;`
	`signalCount: Integer;`
	`sampleCount: Integer);`

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. 57.

4.2.4 Close

C++	`HRESULT Close();`
BASIC	`Sub Close()`
Delphi	`procedure Close;`

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

Eine Übersicht aller möglichen Befehle erhalten Sie im Kapitel "Überblick", S. 57.

4.2.5 SignalCount

C++	`long SignalCount();`
BASIC	`Function SignalCount() As Long`
Delphi	`function SignalCount: Integer;`

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. 57.

4.2.6 Signal

C++	`INvxSignalPtr Signal(long index);`

BASIC	`Function Signal(index As Long) As INvxSignal`

Delphi	`function Signal(index: Integer): INvxSignal;`

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. 57.

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++	`__declspec(property(get=GetName,put=PutName))` `  _bstr_t Name;`

BASIC	`Property Name As String`

Delphi	`property Name: WideString` `  read Get_Name write Set_Name;`

Liefert den Namen des Signals.

Eine Übersicht aller möglichen Befehle erhalten Sie im Kapitel "Überblick", S. 60.

4.3.3 GroupName

C++	`__declspec(property(get=GetName,put=PutName))` `  _bstr_t Name;`

BASIC	`Property GroupName As String`

Delphi	`property GroupName: WideString` `  read Get_GroupName write Set_GroupName;`

Liefert den Gruppennamen des Signals.

Eine Übersicht aller möglichen Befehle erhalten Sie im Kapitel "Überblick", S. 60.

4.3.4 Comment

C++	`__declspec(property(get=GetComment,put=PutComment))` `  _bstr_t Comment;`
BASIC	`Property Comment As String`
Delphi	`property Comment: WideString` `  read Get_Comment write Set_Comment;`

Liefert den Kommentar des Signals.

Eine Übersicht aller möglichen Befehle erhalten Sie im Kapitel "Überblick", S. 60.

4.3.5 xStart

C++	`__declspec(property(get=GetxStart,put=PutxStart))` `  double xStart;`
BASIC	`Property xStart As Double`
Delphi	`property xStart: Double` `  read Get_xStart write Set_xStart;`

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. 60.

4.3.6 xEnd

C++	`__declspec(property(get=GetxEnd,put=PutxEnd))` `  double xEnd;`

BASIC	`Property xEnd As Double`

Delphi	`property xEnd: Double` `  read Get_xEnd write Set_xEnd;`

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. 60.

4.3.7 xDelta

C++	`__declspec(property(get=GetxDelta,put=PutxDelta))` `  double xDelta;`

BASIC	`Property xDelta As Double`

Delphi	`property xDelta: Double` `  read Get_xDelta write Set_xDelta;`

Liefert die Abtastzeit des Signals in Sekunden.

Eine Übersicht aller möglichen Befehle erhalten Sie im Kapitel "Überblick", S. 60.

4.3.8 xUnit

C++	`__declspec(property(get=GetxUnit,put=PutxUnit))` `  _bstr_t xUnit;`
BASIC	`Property xUnit As String`
Delphi	`property xUnit: WideString` `  read Get_xUnit write Set_xUnit;`

Liefert die Einheit der x-Achse.

Eine Übersicht aller möglichen Befehle erhalten Sie im Kapitel "Überblick", S. 60.

4.3.9 xSetUsing

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

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

Delphi	`procedure xSetUsing(format: Integer;`
	`  width: Integer;`
	`  frac: Integer;`
	`  opt: Integer);`

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. 60.

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	`procedure xGetUsing(var format: Integer;`
	`var width: Integer;`
	`var frac: Integer;`
	`var opt: Integer);`

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. 65 beschrieben.

Eine Übersicht aller möglichen Befehle erhalten Sie im Kapitel "Überblick", S. 60.

4.3.11 yMin

C++	`__declspec(property(get=GetyMin,put=PutyMin))` `double yMin;`

BASIC	`Property yMin As Double`

Delphi	`property yMin: Double` `read Get_yMin write Set_yMin;`

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

Eine Übersicht aller möglichen Befehle erhalten Sie im Kapitel "Überblick", S. 60.

4.3.12 yMax

C++	`__declspec(property(get=GetyMax,put=PutyMax))` `  double yMax;`
BASIC	`Property yMax As Double`
Delphi	`property yMax: Double` `  read Get_yMax write Set_yMax;`

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

Eine Übersicht aller möglichen Befehle erhalten Sie im Kapitel "Überblick", S. 60.

4.3.13 yDefaultMin

C++	`__declspec(property(get=GetyDefaultMin,put=PutyDefaultMin))` `  double yDefaultMin;`
BASIC	`Property yDefaultMin As Double`
Delphi	`property yDefaultMin: Double` `  read Get_yDefaultMin write Set_yDefaultMin;`

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. 60.

4.3.14 yDefaultMax

C++	`__declspec(property(get=GetyDefaultMax,put=PutyDefaultMax))` `  double yDefaultMax;`
BASIC	`Property yDefaultMax As Double`
Delphi	`property yDefaultMax: Double` `  read Get_yDefaultMax write Set_yDefaultMax;`

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. 60.

4.3.15 yDelta

C++	`__declspec(property(get=GetyDelta,put=PutyDelta))` `  double yDelta;`
BASIC	`Property yDelta As Double`
Delphi	`property yDelta: Double` `  read Get_yDelta write Set_yDelta;`

Liefert die Auflösung der Messwerte des Signals.

Eine Übersicht aller möglichen Befehle erhalten Sie im Kapitel "Überblick", S. 60.

4.3.16 yUnit

C++	`__declspec(property(get=GetyUnit,put=PutyUnit))` `  _bstr_t yUnit;`
BASIC	`Property yUnit As String`
Delphi	`property yUnit: WideString` `  read Get_yUnit write Set_yUnit;`

Liefert die Einheit der y-Achse.

Eine Übersicht aller möglichen Befehle erhalten Sie im Kapitel "Überblick", S. 60.

4.3.17 ySetUsing

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

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

Delphi	`procedure ySetUsing(format: Integer;`
	`  width: Integer;`
	`  frac: Integer;`
	`  opt: Integer);`

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. 60.

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	`procedure yGetUsing(var format: Integer;`
	`var width: Integer;`
	`var frac: Integer;`
	`var opt: Integer);`

Liefert die Einstellungen für das aktuelle Zahlenformat der Messwerte des Signals. Die Bedeutung der einzelnen Parameter ist im Kapitel "ySetUsing", S. 70 beschrieben.

Eine Übersicht aller möglichen Befehle erhalten Sie im Kapitel "Überblick", S. 60.

4.3.19 ScanStart

C++	`__declspec(property(get=GetScanStart,put=PutScanStart))` `double ScanStart;`

BASIC	`Property ScanStart As Double`

Delphi	`property ScanStart: Double` `read Get_ScanStart write Set_ScanStart;`

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. 60.

4.3.20 SampleCount

C++	`long SampleCount();`
BASIC	`Function SampleCount() As Long`
Delphi	`function SampleCount: Integer;`

Liefert die Anzahl der Messwerte des Signals.

Eine Übersicht aller möglichen Befehle erhalten Sie im Kapitel "Überblick", S. 60.

4.3.21 ScaleX

C++	`HRESULT ScaleX(double xStart,`
	`double xEnd,`
	`long px);`

BASIC	`Sub ScaleX(xStart As Double,`
	`xEnd As Double,`
	`px As Long)`

Delphi	`procedure ScaleX(xStart: Double;`
	`xEnd: Double;`
	`px: Integer);`

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. 60.

4.3.22 ScaleY

C++	`HRESULT ScaleY(double yMin,`
	`  double yMax,`
	`  long py);`

BASIC	`Sub ScaleY(yMin As Double,`
	`  yMax As Double,`
	`  py As Long)`

Delphi	`procedure ScaleY(yMin: Double;`
	`  yMax: Double;`
	`  py: Integer);`

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. 60.

4.3.23 ResetDataPosition

C++	`HRESULT ResetDataPosition();`
BASIC	`Sub ResetDataPosition()`
Delphi	`procedure ResetDataPosition;`

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. 60.

4.3.24 GetNextScaled

C++	`VARIANT_BOOL GetNextScaled(long *min, long *max);`
BASIC	`Function GetNextScaled(min As Long, max As Long) As Boolean`
Delphi	`function GetNextScaled(out min: Integer; out max: Integer): WordBool;`

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. 60.

4.3.25 GetNextScaledDigital

C++	`VARIANT_BOOL GetNextScaledDigital(long *min, long *max);`
BASIC	`Function GetNextScaledDigital(min As Long, max As Long) As Boolean`
Delphi	`function GetNextScaledDigital(out min: Integer; out max: Integer): WordBool;`

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. 60.

4.3.26 Unscale

C++	`HRESULT Unscale();`
BASIC	`Sub Unscale()`
Delphi	`procedure Unscale;`

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. 60.

4.3.27 NextSample

C++	`__declspec(property(get=GetNextSample,put=PutNextSample)` `  double NextSample;`
BASIC	`Property NextSample As Double`
Delphi	`property NextSample: Double` `  read Get_NextSample write Set_NextSample`

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. 60.

4.3.28 NextDigitalSample

C++	`__declspec(property(get=GetNextDigitalSample, put=PutNextDigitalSample) long NextDigitalSample;`
BASIC	`Property NextDigitalSample As Long`
Delphi	`property NextDigitalSample: Long read Get_NextDigitalSample write Set_NextDigitalSample`

Liefert den nächsten Wert eines Digitalsignals.

Eine Übersicht aller möglichen Befehle erhalten Sie im Kapitel "Überblick", S. 60.

4.3.29 GetSampleAt

C++	`double GetSampleAt(double time);`
BASIC	`Function GetSampleAt(time As Double) As Double`
Delphi	`function GetSampleAt(time: Double): Double;`

Liefert einen Messwert zu einer bestimmten Zeit innerhalb des Signals.

Eine Übersicht aller möglichen Befehle erhalten Sie im Kapitel "Überblick", S. 60.

4.3.30 GetSampleAtOffset

C++	`double GetSampleAtOffset(long offset);`
BASIC	`Function GetSampleAtOffset(offset As Long) As Double`
Delphi	`function GetSampleAt(offset: Integer): Double;`

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. 60.

4.3.31 IsAnalog

C++	`VARIANT_BOOL IsAnalog();`
BASIC	`Function IsAnalog() As Boolean`
Delphi	`function IsAnalog: WordBool;`

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

Eine Übersicht aller möglichen Befehle erhalten Sie im Kapitel "Überblick", S. 60.

4.3.32 IsDigital

C++	`VARIANT_BOOL IsDigital();`
BASIC	`Function IsDigital() As Boolean`
Delphi	`function IsDigital: WordBool;`

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

Eine Übersicht aller möglichen Befehle erhalten Sie im Kapitel "Überblick", S. 60.

5 Index

G

I

K

L

M

N

O

P

R

S

T

U

V

X

Y

Z