TestPoint ä
Neue Funktionen in Version 3.0

Limited Warranty

Capital Equipment Corporation (CEC) warrants the physical diskettes and documentation enclosed herein to be free of defects in materials and workmanship for a period of sixty days from the purchase date. The entire and exclusive liability and remedy for breach of this Limited Warranty shall be limited to replacement of defective diskettes or documentation, and shall not include or extend to any claim for or right to recover any other damages, including but not limited to, loss of profit, data, or use of the software, or special, incidental, or consequential damages or other similar claims. In no event shall CEC's liability exceed the purchase price of the product.

TestPoint v3 Features
Part number 04000-90100, vol. 3
First edition
99 98 97 96
10 9 8 7 6 5 4 3 2 1

Capital Equipment Corporation
900 Middlesex Turnpike Building 2.
Billerica, Massachusetts 01821
(508) 663-2002

Kapitel 1. Übersicht

Diese Beschreibung ist als Zusatz zum TestPoint Handbuch gedacht.

TestPoint Version 3.0 enthält viele neue Funktionen, bei einigen handelt es sich um neue Objekte, andere sind Erweiterungen oder „plus“ Versionen bestehender Objekte.

Neue Objekte

OLE2 - Die Object Linking and Embedding Unterstützung wurde auf den OLE 2.0 Standard erweitert. Das neue OLE2 Objekt arbeitet schneller und bietet die Möglichkeit der ruckfreien Aktualisierung von Anzeigen.

OCX, or Active-X, Custom Controls - Die neue Version der Custom Controls wird jetzt voll von TestPoint unterstützt. TestPoint v3.0 unterstützt beide Formen, die älteren VBX und die neuen OCX Controls, die wie die bereits vorhandenen Objekte in TestPoint eingesetzt werden können. Hunderte von unterschiedlichen OCX Controls sind von verschiedenen Firmen erhältlich. Die Funktionalität reicht von simplen Benutzerschnittstellen bis zu Objekten zur Datenbankanbindung und speziellen Graphikanwendungen.

OLE Automation - TestPoint bietet nun auch den Zugriff auf die brandneue Windows Technologie der OLE Automation, damit ist eine komplette Kontrolle anderer Anwendungen wie Microsoft Excel möglich. OLE Automa-
tion erlaubt Ihnen, Objekte zu erzeugen und zu bearbeiten, die von anderen Anwendungen bereitgestellt werden (z. B. Excel Charts)

Erweiterte Objekte

plus Das Analog/Digital Objekt hat nun eine eingebaute Funktion, die eine direkte Umrechnung der Eingangsspannungen erlaubt. Die Skalierung von Sensoren (z. B. Thermoelementen) kann für jeden Kanal getrennt vorgenommen werden. Außerdem besitzt das Objekt jetzt eine sehr schnelle log-to-disk Funktion, mit der Daten direkt auf ein Laufwerk geschrieben werden können.

plus Das Data-Entry Objekt bietet als Neuerung eine Multi-Zeilen Funktion (inkl. Scrollen), mit der auch Texte über mehrere Zeilen angezeigt und bearbeitet werden können. Es kann auch benutzt werden, um Daten über mehrere Zeilen darzustellen, wobei jeder Wert einer neuen Zeile zugeordnet wird.

plus Mit dem Code Objekt können unter **Windows 95** oder **NT** jetzt auch **32-bit DLLs** eingebunden werden. TestPoint ist das einzige Entwicklungssystem, das Ihnen die Möglichkeit gibt, aus einer **einzigen Anwendung 16-bit und 32-bit DLLs** einzubinden.

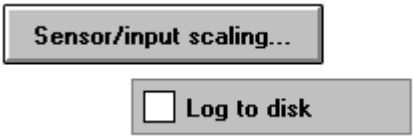
Neu: **Show dialog** action - Sie können die Datei Dialog Box jetzt auch aus Ihrem Programm heraus öffnen, genauso wie Sie bisher den File Button gedrückt haben.

Es wurde die Funktion randomize() hinzugefügt, die eine erneute Initialisierung des Zufallszahlen-Generators erlaubt.

Die neue "Other D/A commands" Aktion erlaubt Ihnen den Zugriff auf hardware-spezifische Funktionen von D/A Karten.

Kapitel 2. A/D „plus“

Dem A/D Objekt wurden neue Funktionen hinzugefügt.

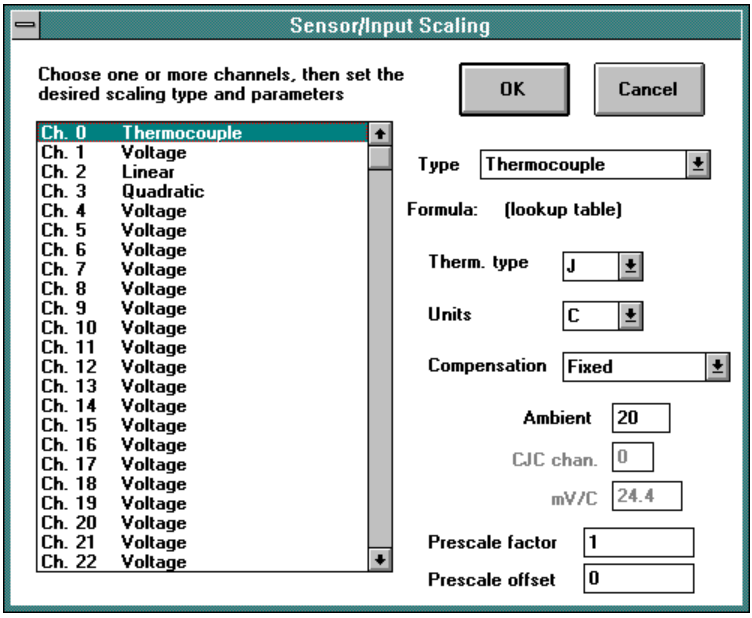


Alle A/D Objekte in existierenden TestPoint Applikationen werden nach dem Einladen in Version 3.0 automatisch durch A/D „plus“ ersetzt.

Das A/D Objekt hat neue Settings, die eine direkte mathematische Bearbeitung der Daten im Objekt erlauben, ohne daß zusätzliche Objekte oder Zeilen in einer Action List erforderlich sind:

Sensor/Eingangs Umrechnung

Wenn Sie den entsprechenden Knopf in den Settings des A/D Objektes betätigen, öffnet sich ein Dialog Fenster, in dem Sie für jeden Kanal getrennt eine Skalierung für die jeweilige Eingangsspannung festlegen können:



Sie können einen oder mehrere Kanäle durch Anklicken in der Liste auswählen (gleichzeitiges Drücken der Ctrl Taste markiert mehrere) und dann die dazugehörige Skalierung auswählen.
Wenn Sie die Apply Scaling Einstellung im Settings Fenster aktivieren, werden die eingelesenen Spannungen automatisch mit den von Ihnen vorgegebenen Parametern umgerechnet. Oben sehen Sie ein Beispiel, bei dem die gemessenen Spannungswerte des 1. Kanals unverändert übernommen werden und für die des 2. Kanals eine lineare Skalierung erfolgt.
Die „raw data“ und „input scaling“ Einstellungen sind exklusiv, es

können nicht beide gleichzeitig aktiviert werden.

Voltage scaling

„Voltage scaling“ bedeutet, daß die für diesen Kanal gemessenen Werte unverändert von der Hardware als Spannungswerte übernommen werden.

Linear scaling

„Linear scaling“ bewirkt eine Umrechnung nach folgender Gleichung, die Parameter A, B sind dann in der Dialog Box einzugeben.

A * x + B

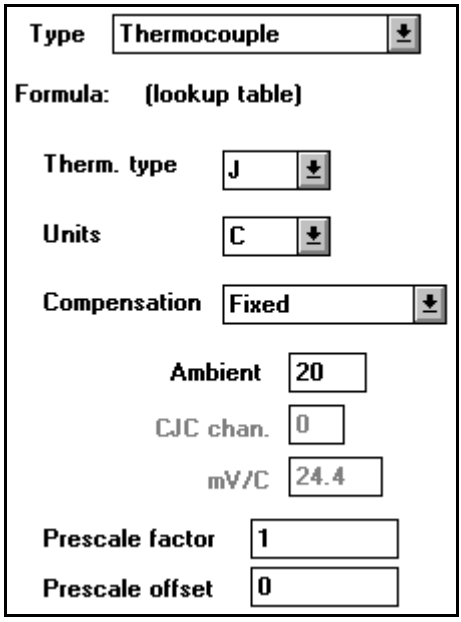
Quadratic scaling

„Quadratic scaling“ benutzt die unten stehende Gleichung, die Parameter A, B, C müssen in der Dialog Box eingetragen werden.

A * x^2 + B * x + C

Thermocouple scaling

„Thermocouple scaling“ berechnet aus der Eingangsspannung die dazugehörige Temperatur. Sie müssen eine Anzahl von Parametern eingeben, die den Typ des Thermoelements und dessen Kompensation definieren.



Das Thermoelement kann vom Typ B, E, J, K, R, S, oder T seien. Die Linearisierungstabellen sind intern in TestPoint enthalten.

Als Einheit kann Grad in C, F, oder K gewählt werden.

Die **„Compensation“** Einstellung ist davon abhängig, wie Sie die Thermoelemente angeschlossen haben, denn es besteht nicht nur eine bi-metall Verbindung im Thermoelement, sondern auch an der Anschlußstelle der A/D Karte. Die gemessene Spannung muß durch eine Gleichspannung kompensiert werden, die von der Raumtemperatur abhängig ist.
Wenn Sie keinen Sensor für die Kaltstellenkompensation haben, können Sie mit „fixed“ einen konstanten Wert vorgeben, in dem Sie eine Temperatur in °C vorgeben.
Viele Anschlußboxen für Thermoelemente werden mit einem isothermalen Block ausgeliefert, der über einen CJC (cold-junction compensation) Sensor verfügt. Sie sollten dann in den Einstellungen „CJC Sensor“ wählen und den Kanal angeben, an den der Sensor angeschlossen ist, sowie den Umrechnungsfaktor in mV/°C.

Der „prescale factor/offset“ wird auf die Meßwerte vor der Temperaturberechnung angewendet. Die Einstellungen werden benötigt, wenn Sie externe Vorverstärker für Thermoelemente einsetzen.
Wenn der externe Vorverstärker zum Beispiel einen Verstärkungsfaktor von 50 besitzt, sollten Sie für den „Prescale factor“ 0.02 eintragen.

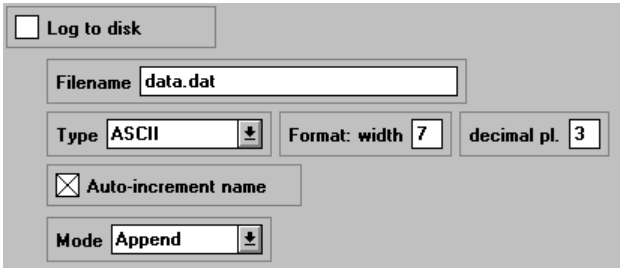
Daten speichern

Die „Log to disk“ Funktion dient dazu, Meßwerte direkt auf ein Laufwerk zu speichern. In den vorherigen TestPoint Versionen war das nur über das File Objekt und einigen zusätzlichen Zeilen in der Action List möglich. Diese Möglichkeit besteht auch weiterhin und sollte bei unüblichen Datenformaten eingesetzt werden.

Bei bestehenden TestPoint Programmen wird die „Log to disk“ Funktion in bestehende A/D Objekte eingefügt. Die „Log to disk“ Funktion ist deutlich schneller als der Weg über das File Objekt.

Achtung: Die „Log to disk“ Funktion steht nur bei den Aktionen „Start A/D“ und „Acquire A/D“ zur Verfügung, nicht bei der Erfassung von Daten mit „Sample A/D“.

Die Settings des A/D Objektes für Datenspeicherung:



Aktivieren Sie einfach „Log to disk“ um die Datenspeicherung einzuschalten.

Filename - Hier können Sie den gewünschten Dateinamen eingeben, eine komplette Angabe des Pfad ist optional (z. B. „c:\daten\data.dat“). Beachten Sie auch die folgende Beschreibung für „Auto-increment name“.

Wenn Sie den Dateinamen während des Ablaufs des Programms eingeben möchten, können Sie das mit wenigen Zeilen in der Action List realisieren. Ziehen Sie einfach die entsprechenden Actions und Settings in die Action List. Ein Beispiel finden Sie weiter unten. Die Action List öffnet ein Datei Dialog Fenster, in dem ein Name eingegeben werden kann, unter dem die Daten dann gespeichert werden:

1)	Show dialog	File1	
2)	Get filename	File1	
3)	Set	A/D1(Filename)	to File1
4)	Acquire A/D	A/D1	# samples=1000, rate=10000 Hz, channel(s)=0

Type - Sie können entweder ASCII (Text) oder ein binär Format für Ihre Datei wählen. Das binäre Datenformat verbraucht weniger Speicherplatz, und es ist später einfacher, Teile daraus wieder zu laden, da jeder Wert exakt 8 Byte benötigt. Das ASCII Format ist besser dafür geeignet, die Daten in andere Anwendungen zu übernehmen, wie z. B. Tabellenkalkulationsprogramme.

Format - Wenn Sie ASCII Format wählen, können Sie mit dieser Einstellung die Anzahl der Zeichen pro Wert angeben, und wie viele Nachkommastellen verwendet werden sollen. Wenn die eingetragene Anzahl der Zeichen zu klein ist, wird der Wert mit der kleinstmöglichen erforderlichen Anzahl von Zeichen heraus geschrieben. Wenn ein Wert größer als erforderlich angegeben wird, werden entsprechend Leerzeichen eingefügt (so daß die Werte später in der Datei in Spalten genau untereinander stehen).

Auto-increment name - Wenn diese Einstellung aktiviert wird, dient der gewählte Dateiname als Basis für die Berechnung weiterer Dateinamen. Eine fortlaufend ansteigende Nummer wird dem vorgegebenem Dateinamen angefügt, so daß jede Datei einen neuen Namen erhält (z. B. DATA0001.DAT, DATA0002.DAT, ...). Ein

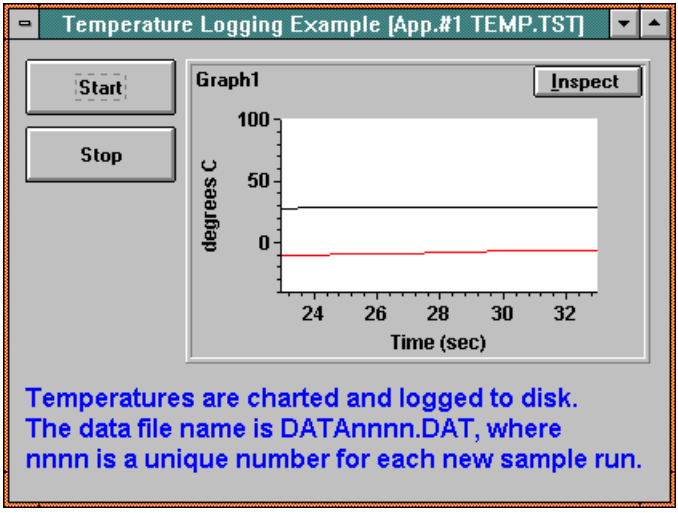
neuer Dateiname wird für jeden „Start A/D“ oder „Acquire A/D) Aufruf erzeugt. Diese Option garantiert, daß für jeden Datensatz eine neue Datei angelegt wird.

Der „auto-incrementing“ Algorithmus funktioniert wie folgt: finde alle Dateinamen in dem vorgegebenem Verzeichnis, die das Format „baseNNN.ext“ haben, wobei „base“ und „ext“ die gewählten Dateinamen sind (im vorherigen Beispiel „DATA“ und „DAT“) und NNNN eine Nummer ist. Dann finde die größte dem entsprechende Dateinummer und erhöhe sie um Eins. Wenn der Dateiname bereits 8 Zeichen lang ist oder es besteht schon eine Datei mit 9999 (keine weitere Nummer möglich), dann gebe eine Fehlermeldung aus.

Mode - Wenn „Auto-incrementing name“ nicht aktiviert ist, können Sie mit dieser Einstellung wählen, ob die neuen Daten an die bestehenden angehängt werden oder sie ersetzen (überschreiben) sollen.

Beispiele

Anzeigen und Speichern von Temperaturwerten



Die Action List des Start-Knopfes:

1)	Clear	Graph1	
2)	Start A/D	A/D1	#samples="continuous", rate=1Hz,
	channel(s)="0,1",		event after 1 sample

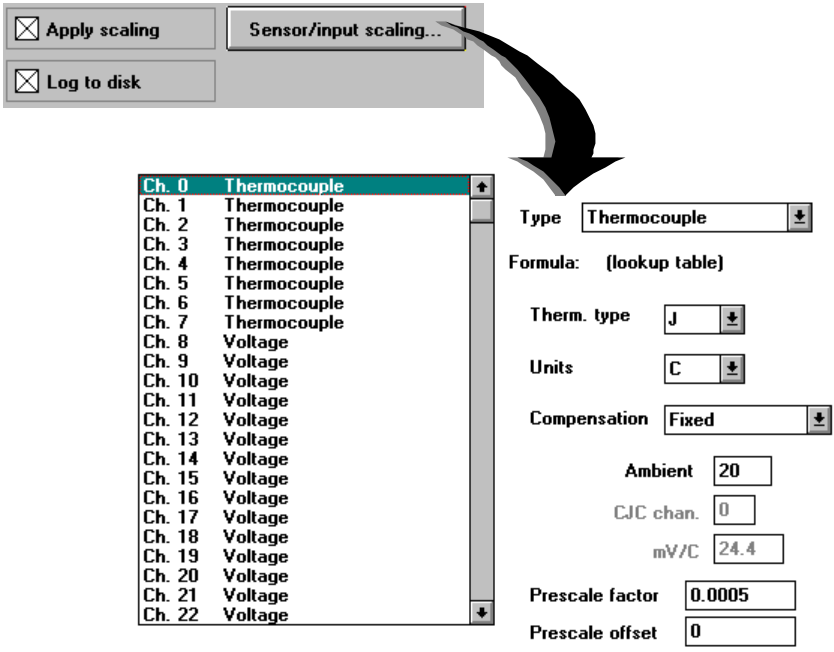
Die Action List des A/D Objektes:

1)	Add point(s) to	Graph1	from A/D1, max. #pts=480

Die Action List des Stop-Knopfes:

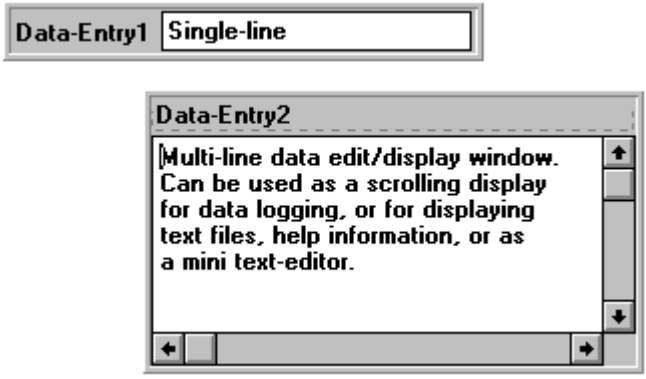
1)	Stop A/D	A/D1

Die Umrechnung und Speicherung der Daten erfordert keine weiteren Zeilen in einer Action List, beides ist ein Bestandteil des A/D Objektes:



Kapitel 3. Data-Entry „plus“

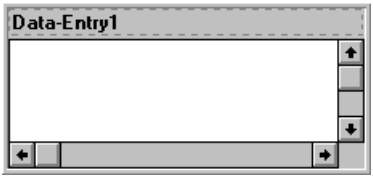
Dem Data-Entry (Daten-Eingabe/Anzeige) Objekt wurden neue Funktionen hinzugefügt.



Alle Data-Entry Objekte in existierenden TestPoint Applikationen werden nach dem Einladen in Version 3.0 automatisch durch Data-Entry „plus“ ersetzt.

Neue Einstellungen

Multi-line - Hiermit aktivieren Sie die Möglichkeit, mehrere Zeilen im Data-Entry Fenster einzugeben und anzuzeigen, anstatt nur einer einzigen Zeile. Vertikale und horizontale Scroll Bars erscheinen automatisch:



Wenn Sie die „Enable“ Setting deaktivieren, können Sie mehrzeilige Texte anzeigen, dem Benutzer wird aber nicht die Möglichkeit gegeben, den Text zu verändern. Beachten Sie, daß die maximale Anzahl von darstellbaren Zeichen in einem Data-Entry Objekt 32K beträgt.

Neue Aktionen

Append to - Diese Action erlaubt Ihnen, an einen existierenden Text im Data-Entry Objekt, neue Teile (Text oder auch numerisch) anzuhängen. Diese Action ist sinnvoll, Anzeigen zur fortlaufenden Datenaufzeichnung zu erstellen.

Achtung: Sie können die Daten, die Sie anfügen, auch formatieren. Als Beispiel sehen Sie hier eine Action List Zeile:

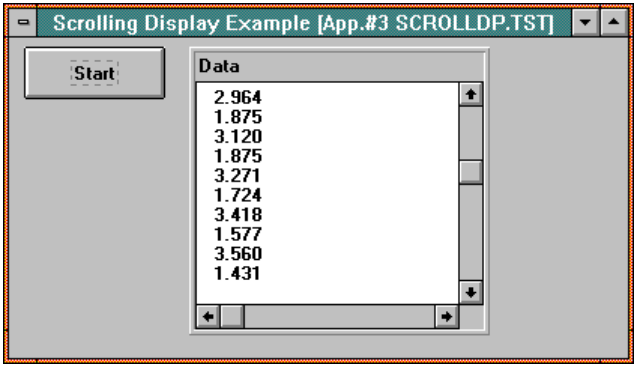
1) Append to Data-Entry1 new line(s)=A/D1

Sie können durch Doppelklicken auf A/D1 in der Zeile den Datentypen spezifizieren und formatieren.

Wenn der Text 32K überschreitet, wird dementsprechend Text vom Anfang gelöscht.

Beispiele

Mitlaufende Text Anzeige



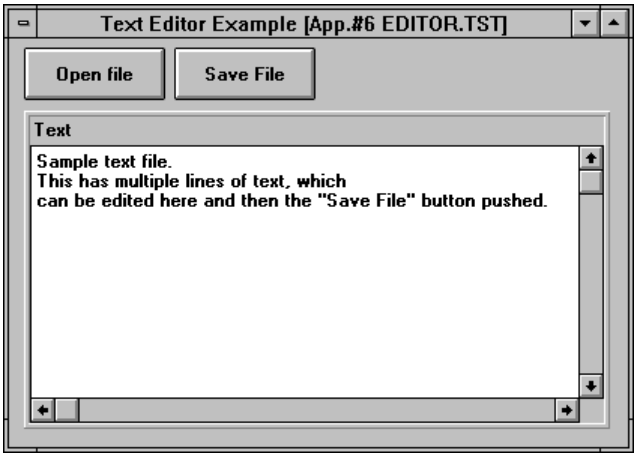
Die Action List des Start-Knopfes:

1) Start A/D A/D1#samples="continuous", rate=2Hz, channels=0, event after 1 sample

Die Action List des A/D1 Objektes:

1) Append to Data new line(s)=A/D1

Einfacher Text Editor



Die Action List des „Open File“-Knopfes:

1)	Show dialog	File1	

Die Action List des File1 Objektes (das Objekt ist nicht sichtbar):

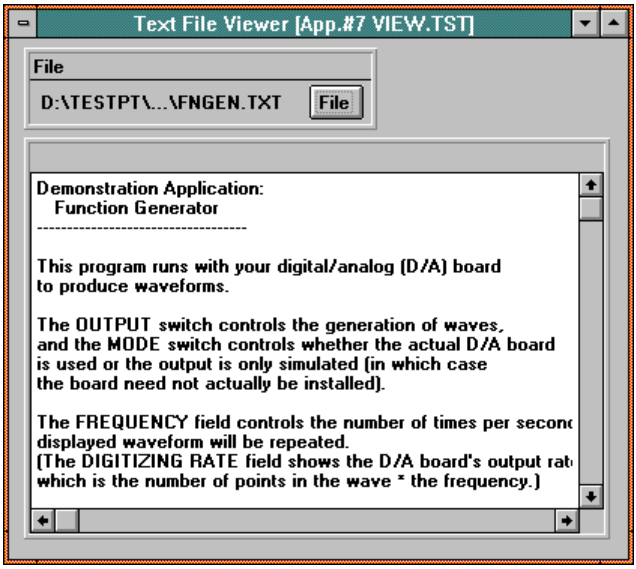
1)	Input from	File1	up to 32768 bytes
2)	Set	Text	to File1

Die Action List des „Save File“-Knopfes:

1)	Erase	File1	
2)	Output to	File1	from Text, term.=none
3)	Close	File1	

Achtung, dieses sehr einfache Beispiel erlaubt keine Dateien > 32K, noch warnt es Sie, die aktuelle Datei zu speichern, bevor Sie eine neue öffnen. Sie können solche Funktionen durch zusätzliche Objekte und Action List Zeilen hinzufügen.

Datei anzeigen



Dieses Beispiel zeigt einfach eine von Ihnen ausgewählte Textdatei an. Die Eigenschaft „Enabled“ des Data-Entry Objektes wurde deaktiviert, so daß es nicht möglich ist, den Text zu bearbeiten.

Sie können dieses Beispiel zur Anzeige von Hilfetexten in Applikationen benutzen, ohne daß Sie extra eine Hilfedatei (*.hlp) erstellen müssen.

Action List des File Objektes:

1)	Input from	File	up to 32768 bytes
2)	Set	Data-Entry1	to File

Kapitel 4. OLE2

Was in diesem Kapitel behandelt wird:

- **Was ist OLE und wie wird es eingesetzt.**

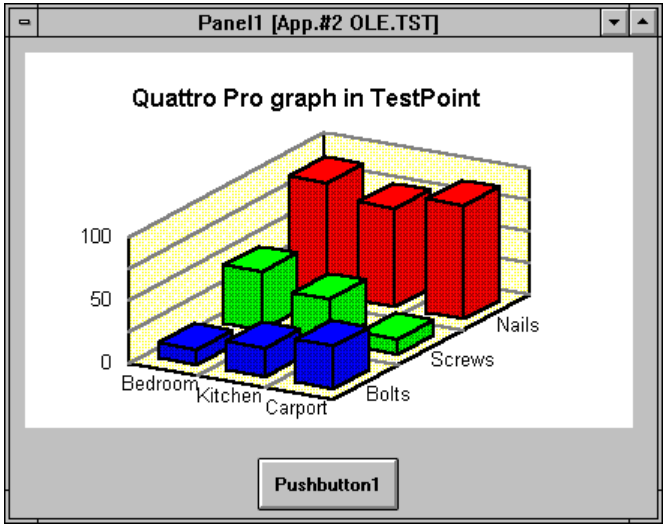
Achtung, dieses Kapitel ersetzt das bisherige Kapitel OLE im TestPoint Handbuch, welches das alte OLE1 Objekt beschreibt.

Existierende TestPoint Applikationen, welche das bisherige OLE Objekt benutzen, funktionieren weiterhin. Diese Objekte werden nicht automatisch durch das OLE2 Objekt ersetzt.

Was ist OLE?

OLE (Object Linking and Embedding) ist eine Funktion von Windows und auch von TestPoint, die es ermöglicht, daß Informationen von einem Programm bearbeitet und dargestellt werden, die Anzeige aber in einer anderen Anwendung geschieht.

Das TestPoint OLE2 Objekt erlaubt Ihnen Daten von anderen Programmen direkt auf einem TestPoint Panel anzuzeigen:



Es gibt zwei Möglichkeiten zum Einsatz des OLE2 Objektes: mit eingefügten (embedded) Daten oder mit verknüpften (linked) Daten.

Wichtig: um OLE2 in Windows 3.x einzusetzen, müssen Sie den SHARE Treiber unter DOS installiert haben. Sie müssen den Befehl SHARE.EXE in Ihre AUTOEXEC.EXE einfügen. Dieser Befehl ist in Ihrem DOS/Windows Handbuch beschrieben.

Eingefügte (embedded) OLE Daten

Eingefügte (embedded) Daten werden in Ihrer TestPoint Datei (*.TST) gespeichert. Auch wenn die Daten vom Format her zu einer anderen Anwendung gehören, sowie z. B. ein Tabellenblatt, die Speicherung findet trotzdem in der TestPoint Datei statt.

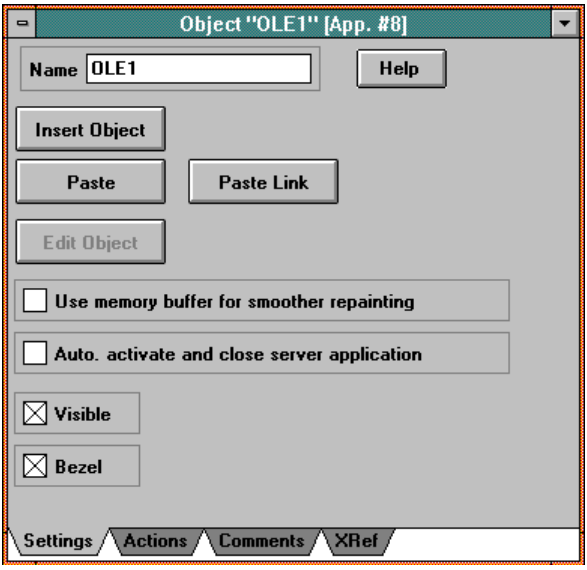
Als Beispiel, wenn Sie mit beim OLE2 Objekt den Knopf „Insert Objekt“ drücken und „Paintbrush Picture“ als Objekttyp gewählt haben, wird das Bild, das Sie zeichnen, in Ihrer TestPoint Datei gespeichert. Es ist keine separate Bitmap-Datei erforderlich, es wird auch keine Datei/Speichern Option in Paintbrush angeboten.
Das ist der Vorteil von eingefügten OLE Daten: Sie müssen keine extra Datei zu Ihrer TestPoint Applikation hinzufügen.

Verknüpfte (linked) OLE Daten

Verknüpfte (linked) Daten kommen aus einer separaten Datei. Sie können zum Beispiel ein Excel Tabellenblatt erzeugen und mit TestPoint verknüpfen.
Um ein verknüpftes OLE Objekt zu erzeugen, benutzen Sie die Bearbeiten/Kopieren Funktion in Ihrem Programm, das als Quelle für die Daten dienen soll (z. B. Excel) und fügen Sie sie dann in den Settings des OLE Objekt mit dem „Paste link“ Knopf ein.

OLE: Präsentieren von Daten aus anderen Anwendungen

Wenn Sie OLE Objekte in Ihre Applikation einfügen, sehen Sie folgendes Settings Fenster für das Objekt:



Als erstes müssen Sie entscheiden, ob Sie eingefügte oder verknüpfte Daten benötigen, wie in dem vorherigen Abschnitt beschrieben wurde. Eingefügte Daten kommen bei statischen Daten zum Einsatz, die nur von Hand verändert werden sollen (diese Daten vergrößern Ihre *.TST Datei). Sie werden in Ihren Anwendungen aber gewöhnlicherweise verknüpfte Daten (Linked Data) benötigen.

Erzeugen von eingefügten Daten oder Einfügen von einer bereits existierenden Datei:

Klicken Sie auf den „Insert Object“ Knopf. Damit öffnen Sie ein Windows Dialog Fenster, daß es Ihnen ermöglicht, den Typ des zu erzeugenden Objektes auszuwählen oder eine bereits bestehende Datei zu öffnen. Daraufhin startet die entsprechende Anwendung und Sie können die OLE Daten bearbeiten. Als Beispiel, Paintbrush dient zum Bearbeiten von Bitmap-Dateien, der Klangrecorder für Audiodateien, usw.

Anstatt einer Datei/Speichern Option finden Sie Datei/Aktualisieren, welche die Daten nach TestPoint übernimmt:

Einige Arten von Daten, wie z. B. Paintbrush Bilder, werden direkt auf dem TestPoint Panel angezeigt, andere, z. B. Audiodateien, nur als Icon:

Kopieren von OLE Daten aus anderen Anwendungen

Sie können OLE Daten auch kopieren und einfügen. Gehen Sie einfach in die entsprechende Anwendung, die die OLE Daten enthält, wie z. B. Excel Chart, wählen Sie die Daten aus und benutzen Sie Bearbeiten/Kopieren aus der Menüleiste.

Mit dem „**Paste**“ Knopf in den Settings des OLE Objektes können Sie die Daten in TestPoint einfügen.

Bearbeiten von eingefügten Daten:

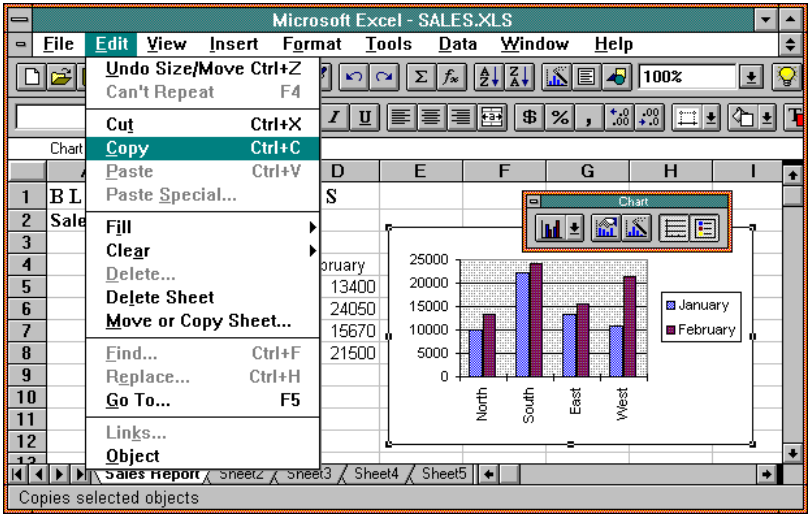
Öffnen Sie wieder das Settings Fenster des OLE Objektes und drücken Sie den „Edit Object“ Knopf.

Erzeugen von verknüpften Daten:

Als erstes benutzen Sie das jeweilige Programm, um die Daten zu erstellen (z. B. Excel, 123, Word, Paintbrush, usw.).

Speichern Sie die Daten des Quell-Programms in eine Datei. Das ist wichtig, denn eine OLE Verknüpfung muß auf ein Datei verweisen, deshalb müssen Sie die Daten speichern, bevor Sie die Verknüpfung erzeugen.

Markieren Sie die Daten im Quell-Programm und wählen Sie aus Bearbeiten/Kopieren aus dem Menü aus:



In TestPoint betätigen Sie dann den „Paste link“ Knopf des OLE Objektes.

Bidirektionale Verknüpfungen mit OLE: Aktualisieren von Darstellungen

Eine der mächtigsten Eigenschaften des OLE Objektes ist die Möglichkeit von bidirektionalen Verknüpfungen. Das bedeutet, TestPoint zeigt die Grafik einer anderen Anwendung an, z. B. Excel, und die Grafik beruht auf Daten, die TestPoint liefert. Immer wenn sich die Daten in TestPoint ändern, wird die OLE Grafik automatisch aktualisiert!

Das erfordert, daß das OLE Quell-Programm (z. B. Excel) gestartet ist, um die Daten automatisch zu bearbeiten. Es erfordert außerdem, daß dem Programm die Daten von TestPoint über eine „Verknüpfung einfügen“ Operation bereitgestellt werden.

Hier ist eine Schritt-für-Schritt Beschreibung, wie Sie eine solche Verknüpfung erstellen können (z. B. mit Excel):

1. Starten Sie TestPoint und Excel.

2. Stellen Sie sicher, daß Sie *.TST und die Excel Datei gespeichert haben (für eine Verknüpfung ist ein Dateiname unbedingt erforderlich).

3. Wählen Sie das Objekt aus, daß in TestPoint die Daten liefern soll, das kann ein Grid, ein Math Objekt, usw. sein. Als Beispiel fügen Sie ein Container Objekt ein und nennen es „OLE source“. Zum Aktualisieren der OLE Daten fügen Sie jeweils eine Zeile in der Action List ein, in der dem Objekt die neuen Daten zugewiesen werden.

4. Markieren Sie das Objekt, das die Daten („OLE Source“) enthält, im Object Fenster durch Anklicken.

5. Benutzen Sie aus der Menüleiste Edit/Copy, um das Objekt zu

kopieren.

6. Wechseln Sie nach Excel und wählen Sie eine Zelle aus. In Excel benutzen Sie Bearbeiten/Inhalte einfügen... und Verknüpfen. Das erzeugt eine Verknüpfung von TestPoint nach Excel.

7. Nun erstellen Sie die gewünschte OLE Information in Excel. In diesem Beispiel hier sollten Sie von Excel eine Graphik der Daten zeichnen lassen, die Sie gerade eingefügt (verknüpft) haben.

8. Markieren Sie die OLE Information in Excel (anklicken der Graphik). Wählen Sie Bearbeiten/Kopieren zum Kopieren der Graphik.

9. Wechseln Sie zurück nach TestPoint. Öffnen Sie das Settings Fenster für das OLE Objekt und betätigen Sie den „Paste link“ Knopf.

10. Stellen Sie sicher, daß Sie beide Dateien (TestPoint u. Excel) noch einmal gespeichert haben.

Das ist alles! Starten Sie nun Ihre TestPoint Anwendung. Immer wenn sich die Daten in TestPoint verändern, wird die OLE Graphik aktualisiert.

OLE Aktionen

Die meisten OLE Objekte besitzen Aktionen („Verbs“ im Microsoft Sprachgebrauch). Wenn Sie z. B. ein eingefügtes Sound Object erzeugen, stehen Ihnen zwei Actions zur Verfügung:

Die Anzahl der auswählbaren Actions hängt stark von dem Typ des Objektes ab, aber fast alle bieten die „Edit“ Action, die das Programm zum Bearbeiten der Daten startet, das Ihnen eine Manipulation der Daten erlaubt.

Automatisches Starten des OLE Quell-Programms

Wie schon angemerkt, muß das OLE Quell-Programm bereits gestartet sein, damit die automatische Aktualisierung funktioniert. Das OLE Objekt besitzt eine Eigenschaft (Setting), die ein selbständiges Starten der OLE Anwendung nach der Initialisierung von TestPoint ermöglicht. TestPoint ruft die Anwendung auf und schließt sie auch beim Beenden der TestPoint Applikation wieder.

Weitere Einstellungen für das OLE2 Objekt

Use memory buffer for smoother repainting

Diese Funktion ist normalerweise aktiviert, Sie erlaubt dem OLE Objekt ein Neuzeichnen der Graphik im Speicher, bevor der Bildschirm aktualisiert wird. Das erlaubt einen Wiederaufbau der Graphik in einem Stück, ohne ein sichtbares Löschen der bestehenden Graphik.

Sie können die Funktion deaktivieren, wenn die OLE Quelle zum Aufbau der Graphik relativ lange benötigt und Sie den Vorgang beobachten möchten.

Visible

Die standard TestPoint **visible** (sichtbar) Settings. Wenn Sie diese deaktivieren, wird das Objekt auf dem Panel nicht angezeigt.

Bezel

Die standard TestPoint **bezel** (Umrandung) Settings. Wenn Sie diese deaktivieren, wird kein Rahmen um das Objekt angezeigt.

OLE und Reports

Sie können OLE Objekte in TestPoint Reports ausdrucken:

1)	Print	Report1	OLE2

Die Ausgabe wird von der OLE Quell-Application (z. B. Excel) erstellt, genauso, als ob die OLE Quelle die Daten auf dem Bildschirm darstellen würde.

Wenn Sie die exakte Größe und/oder Position des OLE Objektes im Report festlegen wollen, können Sie die **Print (at)** Action des Report Objektes einsetzen:

1)	Print (at)	Report1	x=1, y=1, w=5, h=2, value=OLE2

Kurzübersicht von OLE mit häufig eingesetzten Anwendungen

Microsoft Excel

Komplette OLE Unterstützung.
Wenn Sie versuchen eine bidirektionale Verknüpfung mit eingefügten Daten zur erstellen, wird sie wegen eines Fehlers in Excel aktualisiert. Deshalb sollten Sie immer eine verknüpfte Datei wählen.

Microsoft Word 2.0

Unterstützt OLE - keine Anzeige als Text, nur als Icon.

Microsoft Draw (gehört zu Word und anderen Programmen)

Unterstützt OLE - bidirektionale Verknüpfungen funktionieren nicht.

Quattro Pro

Erneuert eine bidirektionale Verknüpfung nicht automatisch, wenn Quattro Pro gestartet wird. Benötigt ein manuelles Wiederherstellen der Verknüpfung mit Tools/Links/Refresh, nachdem Quattro gestartet wurde.

Microsoft PowerPoint

Unterstützt OLE, beinhaltet auch eine „Slide show“ Action, um eine Präsentation abzuspielen. „Slide show“ funktioniert nur mit einem verknüpften Objekt, nicht mit einem eingefügten.

Lotus Freelance

Unterstützt OLE

Audiorecorder

Unterstützt OLE - zeigt nur ein Icon. Beinhaltet eine „Play“ Action. Bidirektionale Verknüpfungen werden nicht unterstützt.

Medienwiedergabe

Unterstützt OLE - zeigt Videos als Bild an und besitzt eine Action zur Wiedergabe. Bidirektionale Verknüpfungen werden nicht unterstützt.

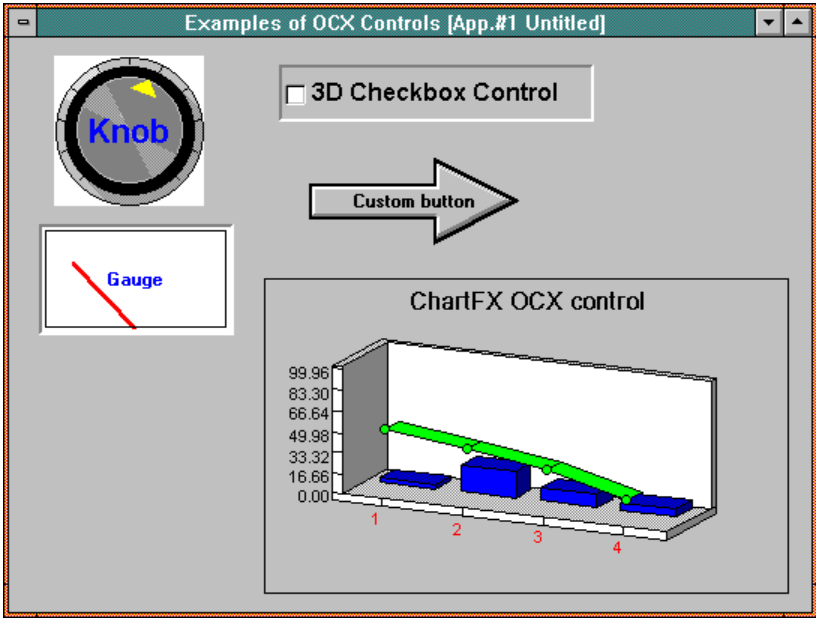
Kapitel 5.
OCX (OLE Custom Controls)

Inhalt dieses Kapitels:

- **Was sind OCX Controls und Software Komponenten?**
- **Wie benutzt man OCX Custom Controls in TestPoint**

Was sind OCX Controls?

OCX Controls sind eine standardisierte Form von Softwarekomponenten, die gekauft und in die eigene Softwareapplikation eingebunden werden können.



Hunderte von OCX Controls sind von Dutzenden von Herstellern erhältlich, sie bieten eine Vielzahl an Softwaremöglichkeiten, wie zum Beispiel Auswahlregler, Anzeigeobjekte und Hilfsmittel für Notizblockanwendungen, Tabellenkalkulationen, Graphikanwendungen und Datenbank Tools. Das obige Beispiel demonstriert einige am Markt befindliche Kontrollobjekte.

Da die OCX Technologie ein offener Standard ist und eine komplette und gut dokumentierte Softwareunterstützung bietet, wuchs der Markt der OCX Controls stetig. Es existieren mindestens zwei Magazine, die sich ausschließlich auf diese Softwareerweiterungen spezialisiert haben: *VB Tech Journal* und *VBX/OCX Developer*.

Eine Sammlung populärer OCX Controls ist **OLE Tools**, von der Firma MicroHelp, Inc., Tel.: 001-404-516-0899. Dieses Paket enthält über 50 Controls für einen ungefähren Preis von ca. DM 150.-.

Eine gute Informationsquelle für OCX Software ist **VBxtras**, diese Firma verkauft OCX Objekte per Katalog, Tel.: 001-404-952-6356.

Wie benutzt man OCX Controls in TestPoint

OCX Controls haben die Form von .OCX Programmdateien, wobei die Dokumentation ihrer Eigenschaften, "Properties", und Events beigefügt ist. OCX Eigenschaften sind ähnlich den TestPoint Settings.

Bitte beachten Sie: Die meisten OCX Controls sind für eine kostenlose Weitergabe in Runtime Programmen lizensiert, ähnlich wie TestPoint. Falls Sie diese Objekte von Ihrem Händler bekommen, wird ihnen eine entsprechende Lizenzdatei beigefügt, die es Ihnen erlaubt, damit neue Applikationen zu erstellen.

Installation von OCX Controls

Den OCX Control Paketen ist eine Installations-Software beigefügt, in der Regel ein SETUP Programm, das vom Windows Program Manager ausgeführt werden kann. Dieses kopiert alle notwendigen Dateien auf Ihren Computer und **registriert** ebenso die Objekte, so daß sie in Anwendungsprogrammen, wie zum Beispiel TestPoint, gleich erscheinen.

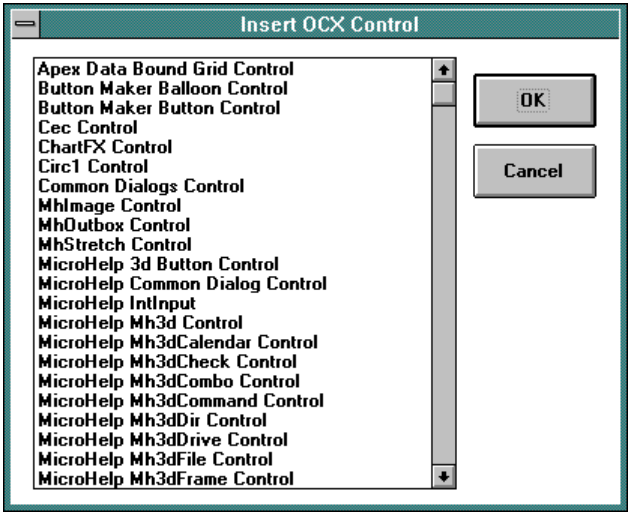
+ **Falls Sie die Wahl haben zwischen der Installation von 16-bit und 32-Bit Versionen der OCX-Objekte, wählen Sie die 16-bit Version. TestPoint hat beide 16- und 32-Bit Features in der Version 3 eingebaut, aber die Implementierung von OCX Elementen ist auf 16-bit Versionen limitiert, da es für alle Windows Versionen kompatibel ist. Generell sind die Eigenschaften beider Versionen identisch in der Funktionalität.**

Laden eines OCX Controls

Um ein OCX Control in Ihre Applikation einzufügen, zieht man das ursprüngliche OCX Objekt 

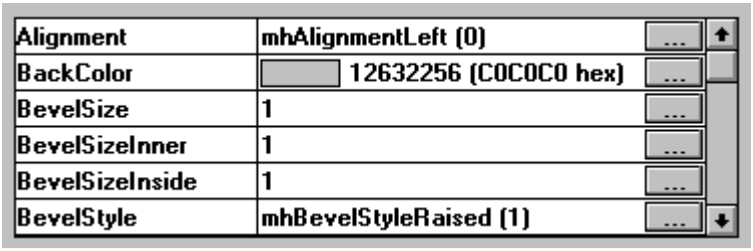
in das Panel.

Durch Drücken auf den **Load OCX** Druckknopf im Settings Fenster wählt man das gewünschte Objekt aus. OCX Installations-Programme **registrieren** diese Controls in dem Windows System, so daß jedes Programm sie benutzen kann. Falls die gewünschten OCX Objekte nicht gefunden werden können, kontrollieren Sie bitte die mitgelieferten Installationsanweisungen - vielleicht sind weitere Registrierungs-Schritte notwendig.



Die OCX Datei ist nun geladen, und das Control ist nun ein TestPoint Objekt, das in Action Lists angesprochen werden kann, und das Settings besitzt. Das optische Aussehen des Objektes jedoch und die Änderungen, die durch eine Settingseingabe durchgeführt werden, wird ausschließlich durch den OCX Control code gesteuert. Auf diese Weise können viele weitere Möglichkeiten zu TestPoint hinzugefügt werden.

Die OCX Settings erscheinen mit Einstellungen , die spezifisch für das benutzte Control Objekt sind, das gerade in der obigen Scrolling List geladen wurde:



Das Objekticon für das OCX Objekt ändert sich ebenso von dem ursprünglichen Icon mit der Aufschrift "OCX", zu einem Icon, das der Programmierer des Custom Controls in das Objekt implementiert hat, z.B.: 

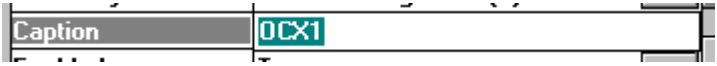
Properties sind noch mehr als blanke Einstellungen

Die Scroll Liste im Eigenschaften Fenster des OCX Objektes zeigt die sogenannten **Properties** für das benutzte Control in alphabetischer Ordnung. Unterschiedliche Controlobjekte besitzen unterschiedliche Properties, die in der mitgelieferten Beschreibung des Objektes dokumentiert sind.

Einige Properties sind gleich für eine Vielzahl von Controls, z. B. **BackColor**, **ForeColor**, und die **Font...** Property. Sie kontrollieren den Text, der in dem Objekt verwendet wird, z.B. die **Caption** oder **Text**-Property, die ein Zeichenstring beinhalten.

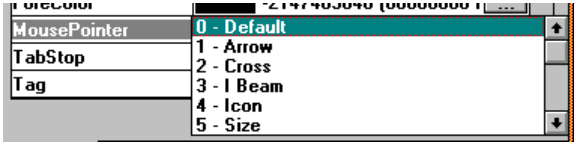
Properties können numerisch (Integer oder Fließkommawert), Zeichenketten, Farben, logische Werte (wahr/falsch), oder freie Wahlfelder sein. Sie können manuell die Property Werte ändern, indem Sie auf diesen Knopf [...] auf der rechten Seite der Property drücken.

Für numerische und String Werte bekommt man eine Editierbox, wie z. B.:



Für Farben bekommt man den Standard Windows Farbendialog.

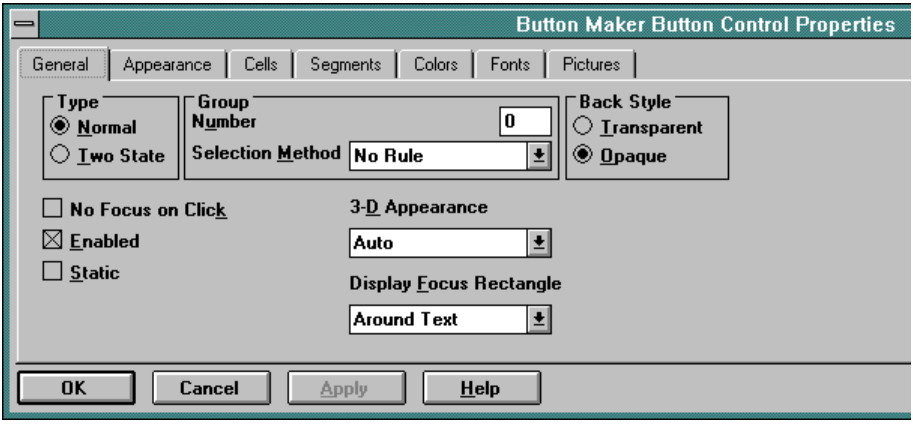
Für logische Werte und Auswahlmenüs bekommt man eine Dropdown Liste:



Custom Settings Dialogfenster

Die meisten OCX Controls besitzen ihre eigenen Custom Dialoge, um ihre Properties zu ändern, der durch den **Custom settings...** Knopf im Settings Fenster aufgerufen werden kann:

Hiermit erscheint ein Dialogfenster, ähnlich dem unten abgebildeten:



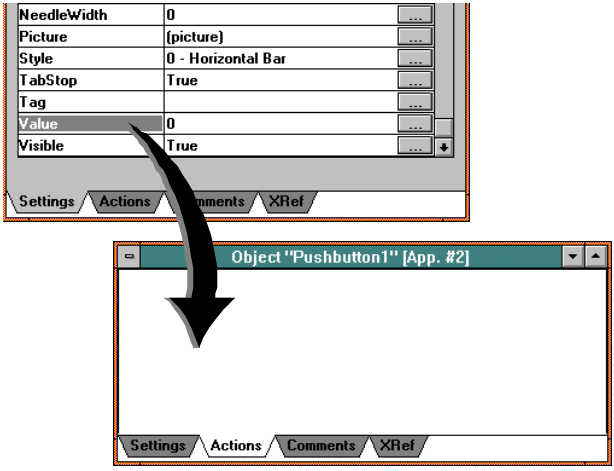
Da dieses Property-Fenster durch den OCX code aufgerufen wird und speziell für das verwendete Objekt geschaffen ist, ist es oft am günstigsten, die gewünschten Eigenschaften mit Hilfe dieses Dialoges einzustellen. Jedoch sind die Felder in diesem Fenster **keine** TestPoint Settings und können deshalb nicht in TestPoint Action Lists hineingezogen werden. Für den Fall, daß es notwendig ist, die OCX Settings während des Programmablaufs zu ändern, muß man das reguläre TestPoint Settings Fenster benutzen (siehe nächsten Abschnitt).

Ändern von OCX Properties in Aktionslisten

Viel vom Zusammenspiel zwischen TestPoint und einem OCX Control geschieht durch das Setzen und Auslesen der OCX Property Werte. Diese Werte können im TestPoint Editor mit Hilfe des Settings Fensters wie vorhin beschrieben voreingestellt werden.

Ziehen das Property aus dem Einstellung Fenster

Die einfachste Art, auf eine Property innerhalb einer TestPoint Action List zuzugreifen, ist analog dem üblichen TestPoint Verfahren: Man muß die Property aus dem Settings Fenster in die Action List ziehen.



Jede OCX Control Property kann geändert werden, indem sie gezogen wird und damit eine neue Action Line erstellt wird, z. B.:

1) Set Gauge(Value) to 83

oder Sie können den Wert einer Property auslesen, in dem Sie sie in einen Action List Parameter ziehen:

2) Calculate Math1 with x=Slider(Value)

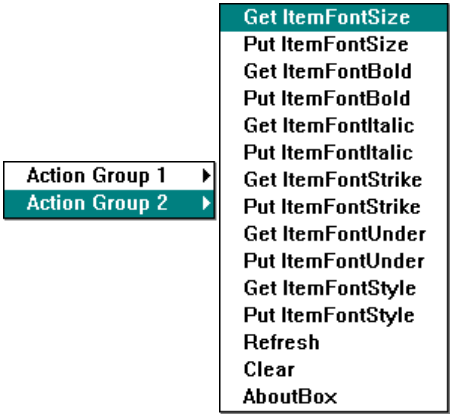
OCX Aktion

Jedes OCX Control bietet eine oder mehrere Actions an, die wiederum typisch für das jeweilige Objekt sind. Die meisten Objekte beinhalten jeweils eine **Properties...** Action (um die Objektproperties während des Programmablaufs darzustellen), und eine **About Box** Action, um die Copyright Information und dessen Versionsnummer anzuzeigen.

Die Dokumentation Ihres OCX Controls bezeichnet normalerweise diese Aktionen als sogenannte "Methoden".

Falls ein Control Objekt in eine Action List gezogen wird, erscheint eine Popup Liste von möglichen Aktionen, wie hier beispielsweise für ein Druckknopfobjekt dargestellt:

Einige OCX Objekte, die mehr als 25 Aktionen bieten, bekommen ein Mehrfachmenü:



OCX Daten Werte

TestPoint Objekte besitzen einen einzelnen Datenwert. OCX Controls haben eine Vielfalt von Properties, deshalb werden die meisten OCX Dateninformationen durch das Hineinziehen von Properties aus dem Settings-Fenster angesprochen, wie es im vorigen Abschnitt erläutert wurde.

Property Werte

Beispielsweise kann ein visuelles Knopfobjekt eine **Value** Property besitzen, die entsprechend der Position des Schalters einen Wert annimmt, der in Action Lists verwendet werden kann, wie folgender:

```
Calculate        Math1        with x=Slider(Value)
```

Daten Werte als Resultat von Aktionen

Einige Aktionen können Datenwerte zurückgeben. Ein Listenobjekt beispielsweise kann Aktionen anbieten, um auf den Status des entsprechenden Listenobjektes zuzugreifen, dies funktioniert auf die selbe Art und Weise, wie eine Bedienung per Hand:

```
1)  Get tagged      Listbox          Index=1
2)  If/Then            x is true     with x=Listbox
3)  Set                Display       to "item 1 selected"
4)  End If             x is true
```

Daten Werte als Resultat von Ereignissen

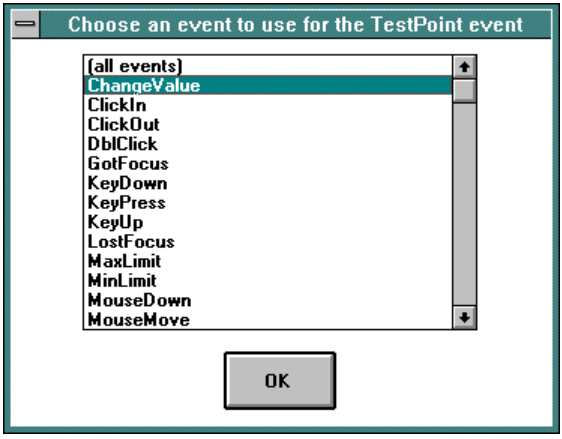
Ein Ereignis, das von einem OCX Control erzeugt wird, modifiziert ebenso den Datenwert des OCX Control (siehe nächster Abschnitt).

OCX Ereignisse

TestPoint Objekte besitzen eine einzige Action List. Einige Objekte führen ihre Action List aus, wenn verschiedenste Ereignisse auftreten, meistens aber reagiert das Objekt nur auf ein einziges Ereignis.

Auf der anderen Seite jedoch definieren OCX Controls typischerweise mehrere unterschiedliche Ereignisse, die sie detektieren können.

Ein Standardereignis wird dem Benutzer angeboten, sobald das Control geladen wird, dennoch ist es möglich, dieses Ereignis zu ändern, indem der **Choose event...** Knopf gedrückt wird und aus einer Liste von möglichen Ereignissen ausgewählt wird:



Viele Eingabe Controls (Knöpfe, Schieberegler, etc.) besitzen ein offensichtliches Vorzugsereignis, z. B. **Change** oder **ChangeValue**, welches ausgelöst wird, wenn der Bediener den Wert des Control Objekts ändert. Damit kann man leicht eine Action List eines OCX Objektes erstellen, die ausgeführt wird, sobald sich der Wert der Eingabe verändert.

Beispielsweise bietet das Schiebereglerobjekt aus dem OLE Tools Paket zahlreiche Eigenschaften, die nicht im TestPoint Schieberegler enthalten sind, aber auf dieselbe Art genutzt werden können. Sobald das Schiebereglerobjekt geladen wurde, kann man die **Change** Action wie oben beschrieben auswählen, die Action List des OCX Objektes auswählen und mit Hilfe des gewohnten Drag-and-drop Verfahrens aus dem Settings Fenster eine Action Zeile erstellen:

1) Set Display1 to Slider(Value)

die das Display erneuert, sobald der Schieberegler bewegt wird. Dies ist dieselbe Programmierweise, wie sie beim TestPoint Schiebregler behandelt wird.

Mehrere Ereignisse

Welche Möglichkeiten gibt es, wenn mehr als ein Ereignis von dem OCX Control benötigt werden? In diesem Fall muß nur in dem **Choose event...** Menü die Option **(all events)** gewählt werden:

Nun wird die Action List für dieses OCX Objekt ausgeführt, wenn irgendeine der ausgewählten Control Events auftreten: z. B. Maustastendruck, Tastendrücke, Wertänderungen, etc.

Um zu ermitteln, welches Ereignis aufgetreten ist, kann man den Datenwert des OCX Objekts verwenden, wie im folgenden beschrieben.

Daten Werte

Sobald ein Ereignis auftritt, wird der Datenwert des OCX Objekts zu einer **Liste**, die folgende Informationen beinhaltet: einen String mit dem Ereignis Namen, und weiteren Null oder mehr Elementen, die Argumente des Ereignisses darstellen, das aufgetreten ist.

Beispielsweise liefern die Ereignisse

- Click
- DblClick
- GotFocus
- LostFocus

Listen zurück, die ein Element enthalten: Den Ereignis Namen als String. Sie haben keine weiteren Argumente.

Diese allgemein definierten Ereignisse liefern folgende Daten zusätzlich zu dem Ereignis Namen zurück:

Event	**Daten Liste**
KeyDown	Tastencode, Flags*
KeyPress	Zeichencode
KeyUp	Zeichencode, Flags*
MouseDown	Tastenflags**, Flags*, x, y
MouseMove	Tastenflags**, Flags*, x, y
MouseUp	Tastenflags**, Flags*, x, y

* Flags geben den Status der Shift und der Ctrl Tasten an: 0 für keine, 1 für Shift, 2 für Ctrl, 3 für beide.

** Tastenflags geben den Status der Maustasten an: 0 für keine, 1 für links, 2 für rechts, 3 für beide.

Um auf die einzelnen Listenelemente zuzugreifen, kann die Mathematikfunktion **select()** verwendet werden. Beispielsweise kann ein Case Objekt in der OCX Action List verwendet werden, das den folgenden Ausdruck als Setting besitzt:

select(event,0)

```
1) Select          Case        with event=OCX
2) When            Case        is "KeyPress"
3)    Execute      Action1
4) When            Case        is "MouseDown"
5)    Execute      Action2
6) End             Case
```

Bemerkungen zu OCX Controls

Bitte beachten Sie: Die meisten OCX Controls sind für eine kostenfreie Erstellung von Runtime Versionen lizensiert, ebenso wie TestPoint. Mit diesen Objekten erhalten Sie eine entsprechende Lizenzdatei, die es Ihnen erlaubt, damit neue Applikationen zu erstellen.

OCX Controls sind externer Softwarecode, der zu TestPoint hinzugefügt wird. Unsere Tests ergaben, daß die meisten von sehr guter Qualität sind. **Einige Objekte jedoch sind fehlerhaft programmiert, diese Fehler müssen bei der Verwendung in TestPoint vermieden werden.** Oft findet man weitere Informationen und Updatedateien für diese Objekte auf einem Bulletin Board oder der Internet Seite des Herstellers.

Falls ein Problem mit einem OCX Objekt auftritt, und Sie neben TestPoint auch Visual Basic oder Borland / Microsoft C++ besitzen, ist es oftmals hilfreich, das Problem in diesen Sprachen zu duplizieren. Somit kann man herausfinden, ob das Problem im Code des OCX Objektes begründet ist.

Programmierung eigener OCX Controls

OCX Controls stellen einen offenen und offengelegten Standard dar, der von der Firma Microsoft unterstützt wird. Alle Hilfsmittel, die notwendig sind, um sie zu erstellen, sind dem Visual C++ Compiler beigefügt.

Es gibt zahlreiche Zeitschriften und Bücher, die sich mit der Programmierung von OCX Objekten beschäftigen.

Es ist oftmals nützlich, ein OCX Objekt zu erstellen, wenn man seine eigene Benutzerschnittstelle definieren will, oder um eine Schnittstelle zu entsprechender Hardware herzustellen. Indem man ein OCX Objekt erstellt, fügt man zu TestPoint ein weiteres Feature hinzu und kann das gleiche Control gleichzeitig in anderen Programmiersprachen benutzen.

Kapitel 6. OLE Automation

Inhalt dieses Kapitels:

- **Was ist OLE Atomation, und wie benutzt man es**
- **Beispiele von OLE Automation**

Was ist OLE Automation?

OLE Automation ist ein Set von Software Standards, die es einem Windows Programm, wie z. B. TestPoint, erlauben, Objekte zu kontrollieren, die von anderen Windowsanwendungen, z. B. Excel, erstellt wurden.

In TestPoint wird OLE Automation durch folgendes Objekt angesprochen: 

Applikationen, die Automation **Server** sind, z. B. Excel, können ein oder mehrere Typen von Objekten bieten, die von anderen Programmen kontrolliert werden dürfen. Excel beispielsweise bietet über 100 Objekttypen wie Grafikcharts, Arbeitsblätter, Zellen oder Zellbereiche. Ein OLE Automation Server kann entweder 16-bit oder 32-Bit code bieten. Automation Objekte können erstellt werden, indem man den Objekt Typ einen Namen vergibt (z. B. "Excel.Chart"), indem eine Datei von einem Datenträger geladen wird (z.B. C:\EXCEL\BOOK1.XLS), oder indem man sich auf ein bereits geladenes Objekt bezieht, wie beispielsweise ein OLE2 Objekt in TestPoint.

Sobald man ein OLE Automation Objekt benutzt, kann man seine **Properties** (Daten) ändern, oder man kann die sogenannten **Methods** aufrufen (diese sind ähnlich wie Unterprogramme). Beispielsweise ist es möglich, den Wert von Tabellenkalkulationszellen zu lesen oder zu schreiben, den Titel oder die Farbe eines Charts zu verändern, etc.

Bitte beachten Sie, daß viele Applikationen, die OLE Support bieten, nicht automatisch auch die Fähigkeit besitzen, OLE Automation zu unterstützen.

Warnung

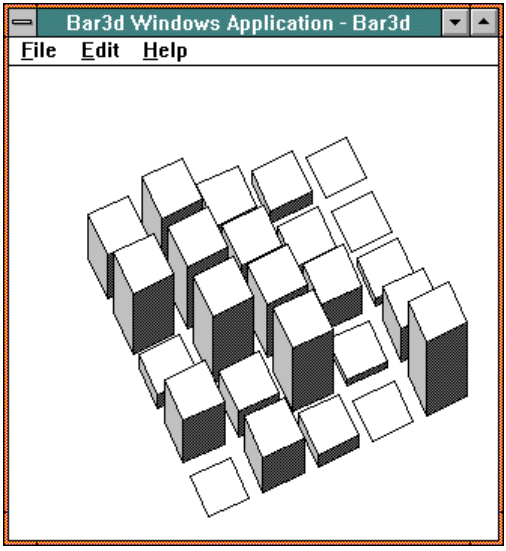
OLE Automation ist ein erweitertes Feature, hauptsächlich gedacht für erfahrene Windows Programmierer. Viele Programme, die dieses Feature unterstützen, eingeschlossen Microsoft Excel, bieten nur sehr eingeschränkte Dokumentation über die eingebauten Objekte, die Eigenschaften und Methoden. Es ist zu erwarten, daß man für die Anwendung dieser Objekte etwas experimentieren muß, und daß man oftmals an die Grenzen mancher OLE Server stoßen wird.

Beispiel: BAR3D

Die beste Methode, mit einer OLE Automation Anwendung zu starten, und dabei zu sehen, was OLE Automation ist, ist das BAR3D Beispiel, das mit TestPoint mitgeliefert wird.

Ausführen von BAR3D.EXE

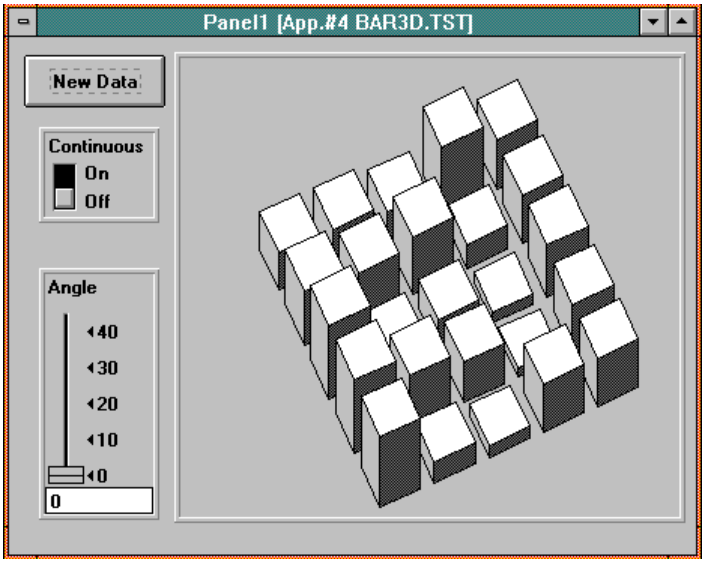
Erstens muß BAR3D.EXE vom EXAMPLES Verzeichnis aufgerufen werden. Dieser erste Durchlauf des Programms zeigt Ihnen, wie die BAR3D Server Applikation aussieht (Darstellung einer begrenzten Form von 3D Balkengrafik), ebenso **registriert** sich dieses Programm als ein Automation Server:



Nächster Schritt, schließen Sie BAR3D.

Ausführung von BAR3D.TST in TestPoint

Dann öffnen Sie TestPoint, laden und führen Sie das BAR3D.TST Beispiel aus:



Versuchen Sie den "New data" Knopf einige Male zu drücken. Dann stellen Sie den Schalter auf "Continuous" und bewegen den "Angle" Schieberegler, worauf sich die Grafik ändert..

Wie funktioniert dies?

Die Grafik, die Sie auf dem TestPoint Panel sehen, ist ein einfaches OLE2 Objekt, das eine eingebettete 3DBalkengrafik enthält.

Der **New Data** Knopf berechnet ein Math Objekt, das eine Reihe von Zufallszahlen generiert. Dieses Math Objekt, eigentlich der Datenwert des Math Objekts, wurde kopiert und in das BAR3D Objekt mit Hilfe eines **link** eingebettet. Dieser Schritt geschieht analog eines Two-way OLE link, wie es im vorigen OLE2 Kapitel schon beschrieben wurde.

Der **Angle** Schieberegler ist nun das Objekt, bei dem OLE Automation ins Spiel kommt. Seine Action List

```
1) Set property      Automation      name="Angle"
                                     to value Angle
```

greift auf die "Angle" Property des BAR3D Objektes zu. Das ist ein Automation Datenwert, der ferngesteuert gesetzt werden kann und welcher in diesem Fall den Sichtwinkel der Grafik representiert.

Der **Init** Knopf (der automatisch bei der Initialisierung ausgeführt wird), sorgt dafür, daß das OLE Automation Objekt sich auf diese Grafik bezieht, dies geschieht durch folgende Action List:

```
1) Set object        Automation      to    3dBar
```

Das OLE Automation Objekt

Das Objekt 

bezieht sich auf ein Objekt im Speicher, das durch ein anderes Programm, genannt **Server,** erzeugt wurde.

Aktionen

Create object - Erzeugen ein neues Automation Objekt, mit den Parametern Objekt-Typ, oder Klasse, -Name. Beispielsweise besitzen Excel Grafiken den Typ "Excel.Chart". Typnamen bestehen normalerweise aus einem Applikationsnamen, einer Periode, und einem Objektnamen innerhalb der Applikation.

Load object - Laden eines Automation Objekts von einer Datei. Damit dies funktioniert, muß Windows in der Lage sein, den Objekttyp aus dem Dateinamen zu bestimmen (normal aus der 3-Zeichen Extension einer Datei). Zum Beispiel bezieht sich eine .XLS Datei auf ein Excel Arbeitsblatt Objekt.

Set object - Setzen des OLE Automation Objekts in Beziehung zu einem Objekt, das bereits im Speicher geladen ist. Der Action Parameter kann entweder ein TestPoint OLE2 Objekt oder ein Objektdatenwert eines anderen OLE Automation Objekts sein (der Rückgabewert von einer Get property Action).

Release object - Löschen eines Objektes aus dem Speicher.

Set property - Setzen einer gegebenen Property des Objektes auf einen entsprechenden Wert. Die für das Objekt möglichen Properties Eigenschaften hängen vom Objekttyp ab.

Get property - Lesen des Werts einer gegebenen Property. Nach der Ausführung dieser Action ist der Datenwert des OLE Automation Objektes der zurückgegebene Property Wert. Die Properties, die für ein Objekt möglich sind, hängen von dem Objekttyp ab. Property Werte können Zahlen, Strings, etc sein. Sie können ebenso Datenreferenzen zu anderen Automation Objekten sein (siehe weiter unten).

Call - Aufruf einer Funktion, oder **Method**, die vom Objekt angeboten wird. Die möglichen Methoden hängen wiederum vom Objekttyp ab. Eine Methode benötigt Null oder mehr Parameter, die konstant oder Datenwerte von TestPoint Objekten sein dürfen. Eine Methode darf ebenso einen Rückgabewert beinhalten, der in den Datenwert des OLE Automation Objektes plaziert wird.

Properties, die Objekte sind

Einige Properties, oder „Method" Rückgabewerte, dürfen ihrerseits wieder OLE Automation Objekte sein. Beispielsweise erhält diese Action eines Excel Chart Objektes:

1) Get property ChartObject name="ChartTitle"

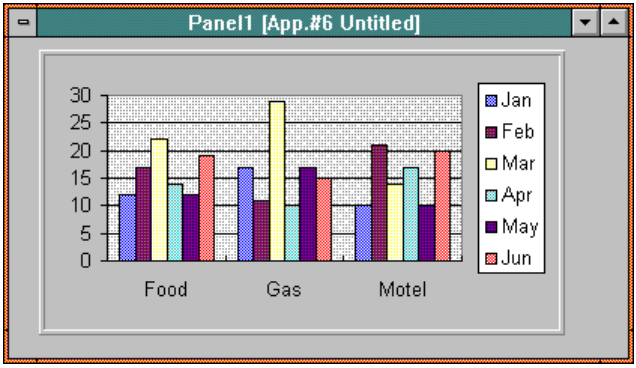
ein ChartTitle Automation Objekt von der Grafik. Dieser Property Wert ist **keine** Zeichenkette - es ist ein anderes Objekt mit eigenen Properties wie Text, Schriftname, etc. Man kann auf dieses Objekt zugreifen, indem man den Rückgabewert mit dem Action **Set object** auf ein anderes OLE Automation Objekt bezieht:

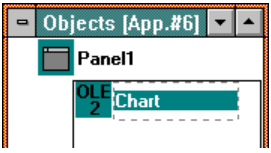
2) Set object TitleObject to ChartObject

Beispiel: Excel Chart

Dieses Beispiel zeigt, wie man Objekten innerhalb eines OLE2 Objektes erlaubt, Eigenschaften wie z. B. den Grafiktitel zu ändern.

Zuerst muß eine eingebettete Excel Grafik erzeugt werden, indem ein OLE2 Objekt und das **Insert Object** Setting verwendet wird. Danach sollte das Objekt **Chart** benannt werden.





Im Moment besitzt diese Grafik noch keinen Titel.

Fügen Sie einen Pushbutton (Druckschalter) hinzu und benennen Sie ihn **Set title**. Schalten Sie auf seine Action List um.

Fügen Sie als nächstes ein OLE Automation Objekt hinzu und benennen Sie es **ChartObject**. Ziehen Sie es in die Pushbutton Action List und erzeugen Sie somit eine Action Line:

1) Set object ChartObject to Chart

Dies teilt dem OLE Automation Objekt mit, daß es sich auf die eingebettete Grafik bezieht.

Ziehen Sie das Automation Objekt noch einmal in die Action List, um noch eine Action Line zu erzeugen

2)	Set property	ChartObject	name="HasTitle" to value 1

die nach Ausführung den Titel anzeigt.
Nun fügen Sie ein weiteres OLE Automation Objekt hinzu und bezeichnen Sie es als **TitleObject**.
Erzeugen Sie diese Action Lines:

3)	Get property	ChartObject	name="ChartTitle"
4)	Set object	TitleObject	to ChartObject

Dies holt das Titelobjekt von der Grafik und bezieht es auf das andere OLE Automation Objekt, von dem man die Properties des Titels selbst ändern kann.
Fügen Sie daraufhin folgende Zeile hinzu:

5)	Set property	TitleObject	name="Text" to value "Sample title"

Dies ändert den Titeltext entsprechend. In ähnlicher Weise kann man auch andere Properties des Titels oder andere Objekte innerhalb der Grafik ändern. Sie benötigen dafür die Excel Dokumentation, die Online Hilfe, oder ein Buch über Excelprogrammierung, um für diese Objekte alle Objektwerte und Properties zu finden, die möglich sind.

Zuletzt fügen Sie folgende Zeilen hinzu, um die Extra Objekt Referenzen wieder zu löschen:

6)	Release object	TitleObject
7)	Release object	ChartObject

Erzeugung eigener OLE Automation Server

Falls Sie erfahrener C oder C++ Programmierer sind, können Sie ziemlich einfach eigene Programme erzeugen, die OLE Automation Features bieten. Dies können Sie entweder mit Microsoft Visual C++ oder Borland C++.

Diesen Compilern ist der Support Code für OLE Automation beigefügt, der es dem Benutzer erlaubt, leicht Properties und Methods seiner Applikation hinzuzufügen, und dann seinen eigenen Code dazuzuschreiben, um die einzelnen Properties zu handhaben.

Dies ist schon ein sehr fortgeschrittenes Thema von TestPoint, aber es bietet eine guten, generellen Weg, um eigene Erweiterungen zu implementieren, die sowohl in TestPoint als auch in anderen Entwicklungsplattformen benutzt werden können.

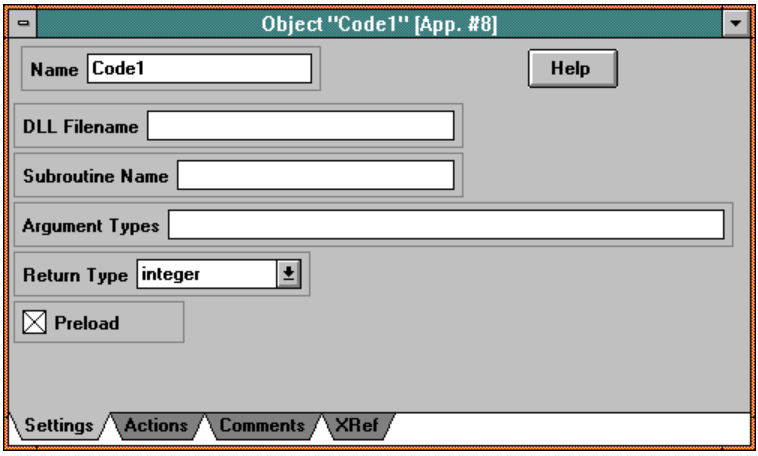
Diese eigenen OLE Automation Server Programme können ebenfalls in der eigenen Applikation eingesetzt werden, oder als Add-on mit TestPoint-Runtime Versionen verkauft werden.

Kapitel 7. Code „plus“

Das Code Objekt von TestPoint ist nun in der Lage, 16-Bit und 32-Bit externen Code aufzurufen **(32-bit Code ist nur unter Windows 95 oder NT möglich)**.

Alle existierenden Applikationen früherer Versionen von TestPoint können ohne jegliche Veränderung weiter benutzt werden.

Das Code Objekt besitzt immer noch die gleichen Settings, wie es sie immer in TestPoint gehabt hat, das bedeutet, der Bediener kann die externe Codedatei (DLL), die gewünschte Unterroutine und die zu übergebenden Argumente auswählen.



Das Code Objekt **erkennt automatisch**, ob der externe Code in 16-Bit oder 32-Bit Form vorliegt - der Programmierer muß dabei nichts beachten.

Bei der Benennung der 32-Bit Subroutinen sollte man sorgfältig auf Groß- und Kleinschreibung achten. Im Gegensatz zu 16-Bit Code wird bei 32-Bit Code zwischen Groß- und Kleinschreibung unterschieden.

Ein Beispiel

Im folgenden sind die Settings aufgelistet, die benutzt werden müssen, um durch eine Windows eigene DLL das Windows Verzeichnis zu erhalten:

DLL filename:	KERNEL
Subroutine name:	GetWindowsDirectory
Argument types:	var string, integer
Return type:	integer

Ein Code Objekt mit diesen Settings ruft dann die Windows Funktion **GetWindowsDirectory** auf und zeigt das Resultat davon an:

```
1) Call          Code1          with Display1, 80
```

Um das selbe mit 32-Bit System Code in Windows 95 oder Windows NT durchzuführen, müssen folgende Settings verwendet werden:

DLL filename:	KERNEL32.DLL
Subroutine name:	GetWindowsDirectoryA
Argument types:	var string, integer
Return type:	integer

Bitte beachten Sie, daß der einzige Unterschied der Name der Funktion und die zugehörige DLL Datei ist. TestPoint erkennt automatisch, daß KERNEL32 32-Bit Code ist und führt die notwendigen internen Änderungen durch.

Unterschiede zwischen 16-Bit und 32-Bit Code

Es gibt einige Unterschiede in der Art und Weise, in der man 16- und 32-Bit Code von TestPoint aus aufruft.

Länge von Argumenten

Bei 32-Bit Code werden alle Argumente als 32-Bit Einheiten übergeben. Deshalb wird eine Integervariable (die in TestPoint als 16-Bit definiert ist) auf einen 32-Bit Wert erweitert, sobald der externe Code aufgerufen wird. Integerwerte, die zurückkommen, werden auf 16-Bit Werte verkleinert.

Arrays von Bytes, Integer, etc. werden **nicht** auf 32-Bit verändert, wenn sie an 32-Bit Code übergeben werden.

Falls der 32-Bit Code einen 32-Bit Wert zurückgibt (die dann als Integer definiert ist, aber eigentlich 32-Bit Integer bezeichnet), sollte das Argument in TestPoint, oder der Rückgabewert, als **long** oder **dword** in TestPoint deklariert werden.

Die 32-Bit Code, Window "Handles" (die **hwnd** oder **window** Argument Typen) sind 32 Bit breit.

Aufrufmethoden

Die meisten 32-Bit Routinen benutzen die **stdcall** Aufrufmethode (siehe nächsten Abschnitt), hingegen verwendet 16-Bit Code hauptsächlich die **pascal** Aufrufmethode.

Funktionsnamen

32-Bit Code unterscheidet zwischen Groß- und Kleinschreibung, deshalb muß die korrekte Schreibweise im Setting des Funktionsnamens eingehalten werden. 16-Bit Code dagegen behandelt Klein- und Großschreibung gleich.

Zeichensätze

32-Bit Windows Code unterstützt zwei verschiedene Arten von Zeichensätzen, den ANSI- und den Unicodezeichensatz. ANSI Zeichen sind im 16-Bit Windows ebenso unterstützt. Unicode kann optional in 32-Bit Code genutzt werden, es ist der Standardzeichensatz für Windows NT Systeme.

Die Zeichensätze werden später genauer beschrieben.

Aufruf Methoden

Der Standard für 16-Bit Code Module (DLLs) war bisher immer die sogenannte **pascal** Aufrufmethode. Um alle Unterschiede zwischen 16 und 32 Bit Codemodellen und den verschiedenen Compilern von Borland, Microsoft, etc., handhaben zu können, unterstützt TestPoint nun drei unterschiedliche Aufrufmethoden: **pascal**, **cdecl**, und **stdcall**.

Die einzelnen Methoden unterscheiden sich in der Buchstabenzusammensetzung der Funktionsaufrufe: einige fügen dem Namen zusätzliche Buchstaben hinzu. Sie unterscheiden sich auch darin, ob die Argumente in der Reihenfolge rechts-nach-links oder links-nach-rechts übergeben werden. Demnach ist es **sehr** wichtig, die richtige Aufrufmethode zu benutzen, damit Ihr Programm korrekt funktioniert. Da glücklicherweise alle drei Methoden unterschiedliche Schreibweisen benutzen, ist TestPoint normalerweise dazu in der Lage, die Aufrufmethode automatisch zu erkennen!

Dennoch ist es möglich, die Aufrufmethode manuell zu bestimmen, indem man sie zu dem Argumenttyp im Codeobjekt hinzufügt, beispielsweise:

```
Argument types:                    cdecl   integer,integer
```

Im folgenden sind die Details der drei Aufrufmethoden aufgelistet:

pascal

An den Funktionsnamen wird nichts hinzugefügt (normal werden alle Buchstaben zu Großbuchstaben).
Argumente werden von links-nach-rechts übergeben.

cdecl

Funktionsname bekommt einen Unterstrich an erster Position, z. B.:

```
_mkdir
```

Argumente werden von rechts-nach-links übergeben.

stdcall

Funktionsname bekommt einen Unterstrich an erster Position, plus einem at-Zeichen (@) und der Anzahl von Bytes der Argumente am Ende, z.B.:

```
_AddTwoNumbers@8
```

Argumente werden von links-nach-rechts übergeben.

Zeichensätze und Windows NT

32-Bit Versionen von Windows unterstützen zwei Arten von Zeichensätzen: ANSI und Unicode. ANSI Zeichenketten sind die gewöhnlichen Ein-Byte-Pro-Zeichen Strings. Unicode Zeichenketten besitzten zwei Bytes pro Zeichen, und werden benötigt, um alle Sprachen mit einem einzigen Zeichensatz zu benutzen.

ANSI Zeichenketten werden sowohl unter 16-Bit (Windows 3.x) als auch 32-Bit (Windows 95 und NT) Umgebungen unterstützt, somit ist dies wohl der am besten geeignete Typ. Alle Funktionen, die im Windows implementiert sind, benutzen diesen Zeichentyp.

TestPoint Datentypen und Unicode

TestPoint **string** Datentypen sind ANSI (ein Byte pro Zeichen) Zeichenketten.

Um Unicode Strings zu benutzen, muß der Argument Typ als ein Array of Words deklariert werden, zum Beispiel:

```
var word[100], integer
```

anstatt:

```
var string, integer
```

Unicode = Windows NT

Unicode Zeichenketten können sowohl unter Windows 95 als auch unter Windows NT manipuliert werden, sind jedoch nur unter Windows NT voll unterstützt. Beispielsweise gibt der Funktionsaufruf GetWindowsDirectoryW nur einen Null String unter Windows 95 zurück, arbeitet aber korrekt unter Windows NT.

Aufrufen von Windows internen Funktionen mit Hilfe von Strings

In Windows 95 und NT gibt es zwei Versionen von String Argumenten für interne Funktionen mit String Argumenten. Sie unterscheiden sich darin, ob ein A oder ein W am Ende des Funktionsaufrufs angefügt wurde.

Zum Beispiel:

GetWindowsDirectoryA gibt einen ANSI String zurück.

GetWindowsDirectoryW gibt einen Unicode String zurück.

Um Funktionen mit Unicode Strings aufzurufen, müssen Arrays of **word** Argumente übergeben werden, wie es im vorigen Abschnitt erläutert wurde.

Umwandlungen zwischen den Zeichensätzen

Um die Zeichensätze ineinander umzuwandeln, gibt es die internen Windowsfunktionen **MultiByteToWideChar** (für ANSI nach Unicode), und **WideCharToMultiByte** (für Unicode nach ANSI). Diese Code Objekte müssen wie folgt deklariert werden:

Code Objekt "ToUnicode":

DLL filename:	KERNEL32.DLL
Subroutine name:	MultiByteToWideChar
Argument types:	int, dword, string, int, var word[100], int
Return value:	integer

Auf diese Weise muß das Codeobjekt aufgerufen werden:

```
1) Call          ToUnicode     with 0, 0, "Input string", -1,
                               Container1, 100
```

Das erste Argument ist die sogenannte "code page" - hierfür sollte man immer 0 wählen. Der zweite Parameter besteht aus Flags für die Umwandlungsoptionen - hierfür sollte ebenfalls 0 gesetzt werden. Der -1 Wert ist die Stringlänge: -1 bedeutet in diesem Fall, daß die Funktion die Länge des Strings bestimmen soll. Das letzte Argument ist die maximale Länge des zurückgegebenen Strings.

Code Objekt "FromUnicode":

```
DLL filename:       KERNEL32.DLL
Subroutine name:    WideCharToMultiByte
Argument types:     int,dword,word[],int,var string,int,dword,dword
Return value:       integer
```

Auf diese Weise wird das Code Objekt aufgerufen:

```
1) Call          FromUnicode   with 0, 0, Container1, -1,
                               Display1, 100, 0, 0
```

Das erste Argument ist wiederum die "code page" - - hierfür sollte man immer 0 wählen. Der zweite Parameter besteht aus Flags für die Umwandlungsoptionen - hierfür sollte ebenfalls 0 gesetzt werden Der -1 Wert ist die Stringlänge: -1 bedeutet in diesem Fall, daß die Funktion die Länge des Strings bestimmen soll. In diesem Beispiel wird der Rückgabewert in ein Display Objekt gesetzt, mit einer maximalen Länge von 100. Die letzten beiden Argumente sind Flags, um die Zeichen handzuhaben, die nicht direkt übersetzt werden können - bei einer 0 wird eine Standardbehandlung dieser Zeichen durchgeführt.

Weitere Details bei der Beschreibung dieser oder anderer Win32 Funktionen kann den technischen Handbüchern und Fachbüchern über Windowsprogrammierung von Microsoft entnommen werden.

Erweitertes Beispiel: GetWindowsDirectoryW

Dieses Beispiel ruft die Unicode Zeichen Version der Windows NT Funktion auf und konvertiert das Ergebnis in einen Standard ANSI String.

Code Objekt Settings:

DLL filename:	KERNEL32.DLL
Subroutine name:	GetWindowsDirectoryW
Argument types:	var word[270], integer
Return type:	integer

DLL filename:	KERNEL32.DLL
Subroutine name:	WideCharToMultiByte
Argument types:	int,dword,word[],int,var string,int,dword,dword
Return value:	integer

Action List:

1) Call	GetWindowsDirectoryW		with Container1, 270
2) Call	FromUnicode	with 0, 0, Container1, -1, Display1, 270, 0, 0	

Bitte beachten Sie, daß die Anwendung beider Funktionen eigentlich nicht unbedingt nötig ist - es ist in der Regel einfacher, die WindowsDirectoryA Funktion stattdessen zu benutzen. Falls jedoch eine 32-Bit Funktion verwendet wird, die ausschließlich die Anwendung von Unicode bietet, ist diese Technik sinnvoll.

Bitte beachten Sie, daß dieses Beispiel nur unter Windows NT, aber nicht unter Windows 95 lauffähig ist, weil die GetWindowsDirectoryW Funktion nur einen 0-Länge String in Win95 zurückgibt.

Kapitel 8. Code Objekt: Benutzen von System-Funktionen

Mit Hilfe des Code Objekts sind alle internen Windows Funktionen als Erweiterung von TestPoint benutzbar. Viele dieser Funktionen sind sehr nützlich und wurden auch schon oft von TestPoint Programmmierern gewünscht. Dieser Abschnitt befaßt sich mit der Dokumentation einiger populärer Windowsfunktionen.

Message Dialog Box

Es ist möglich, ein Info Fenster mit einem eigenen Titel und Test aufzurufen, auf dem verschiedene Knöpfe vorhanden sind. All dies benötigt nur ein Code Objekt:

Die Code Objekt Settings sind:

DLL filename:	USER
Subroutine name:	MessageBox
Argument types:	hwnd, string, string, integer
Return type:	integer

Damit muß man folgende Action List entwerfen:

1)	Call	MessageBox	with Panel1, "Message text", "Title text", 65

Folgende Argumente sind möglich:

hwnd	Fenster, dem die Message Box "gehört"
string	Jeglicher beliebiger Text, der in der Box er scheinen darf
string	Titeltext
integer	Ein Code, der anzeigt, welche Knöpfe und welches Icon gezeigt werden. Folgende Knöpfe kann man auswählen:

0	OK
1	OK/Cancel
2	Abort/Retry/Ignore
3	Yes/No/Cancel
4	Yes/No
5	Retry/Cancel

Dies wird mit einem der folgenden Codes kombiniert, indem die Zahlen addiert werden:

16	Stop Schild
32	Fragezeichen
48	Ausrufezeichen
64	Informationssymbol

Akustikgeber

PC-Lautsprecher

Um einen Ton aus dem PC-Lautsprecher zu erzeugen, muß folgendes Code Objekt benutzt werden:

DLL filename:	USER
Subroutine name:	MessageBeep
Argument types:	integer
Return type:	integer

Dieses Objekt kann in folgender Acion List verwendet werden:

1) Call MessageBeep with 0

Das Argument hat keine weitere Bedeutung und sollte deshalb Null sein.

Sounds (.WAV Dateien)

Um eine .WAV Datei über eine Soundkarte abzuspielen, muß folgendes Code Objekt verwendet werden:

DLL filename:	MMSYSTEM.DLL
Subroutine name:	sndPlaySound
Argument types:	string, integer
Return type:	integer

Als Action List ist hierbei nötig:

1) Call Play with "c:\windows\chimes.wav", 0

Die Argumente sind:

string	Der Name der .WAV Datei, die den Sound beinhaltet
integer	0, um den Sound zu spielen und dann zu warten, oder 1, um den Sound asynchron zu spielen (nicht warten).

Maximieren und Minimieren von Panels

Der **ShowWindow** Aufruf bietet die Möglichkeit, den Fensterstatus von jedem Fenster (maximiert, minimiert, etc.), also auch von TestPoint panels, zu manipulieren.

Die Settings des Code Objekts sind hierbei:

DLL filename:	USER
Subroutine name:	ShowWindow
Argument types:	hwnd, integer
Return type:	integer

Folgende Action List sollte programmiert werden:

1) Call ShowWindow with Panel1, 3

Die Argumente bedeuten:

hwnd	Fenster, das manipuliert werden soll
integer	Ein Code, der Aktion, die erfolgen soll: 0 Verstecken des Fensters 1 Zeigen in normaler Größe 2 Minimieren 3 Maximieren

Ausführen anderer Programme

Der **WinExec** Aufruf kann jedes Windows oder DOS Programm ausführen.

Die korrespondierenden Code Objekt Settings sind:

DLL filename:	KERNEL
Subroutine name:	WinExec
Argument types:	string, integer
Return type:	integer

Sie können dieses Objekt in folgender Action List verwenden:

1) Call WinExec with "notepad", 1

Die Argumente bedeuten:

string	Kommando Zeile, die ausgeführt werden soll
integer	ein Code, der den Status des Programmfen sters anzeigt (falls ein Windows- und kein DOS-Programm ausgeführt wird): 1 Zeigen des Fensters in Normalgröße 2 Minimierung 3 Maximierung

Die Kommandozeile kann jedes Windows- oder DOS-Programm sein, auch Programm Argumente können beigefügt werden. Im folgenden sind einige Beispiele aufgelistet:

```
1) Call      WinExec    with "notepad
                        c:\windows\setup.txt"
2) Call      WinExec    with "C:\TESTPT\
                        TPRUNOTHERAPP.TST"
3) Call      WinExec    with "command.com
                        /c mkdir C:\NEWDIR"
```

Bitte beachten Sie, daß beim letzten Beispiel der DOS Command Handler COMMAND.COM benutzt wird, um das DOS-interne Kommando **mkdir** auszuführen.

Starten einer Applikation und Reaktivierung von TestPoint

Die Windows Funktion **GetActiveWindow** und **SetActiveWindow** kann sicherstellen, daß TestPoint das aktive, das bedeutet im Vordergrund stehende, Programm bleibt, sogar, wenn ein anderes Programm, z. B. Excel, gestartet wurde.

Die Code Objekt Settings sind:

DLL filename:	USER
Subroutine name:	GetActiveWindow
Argument types:	
Return type:	word
DLL filename:	USER
Subroutine name:	SetActiveWindow
Argument types:	word
Return type:	word

Die zugehörige Action List lautet:

1)	Call	GetActiveWindow	
2)	Start application	Excel	command line= "mysheet.xls"
3)	Call	SetActiveWindow with	GetActiveWindow

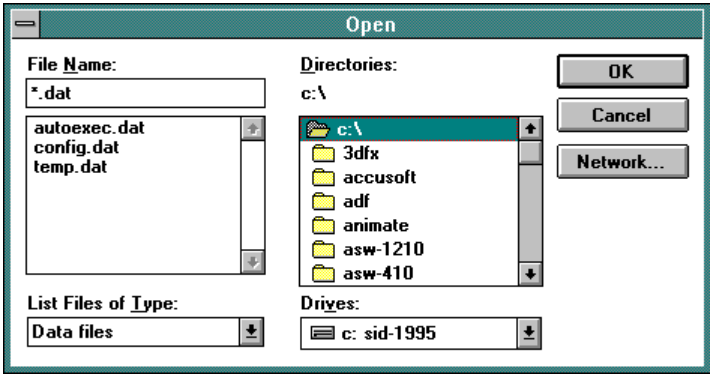
Kapitel 9. Weitere neue Features

Datei (File) Objekt

Das Datei Objekt besitzt nun eine **Show dialog** Action:

1) Show dialog File1

Diese bringt das gewohnte Windows Dateiauswahlmenü auf den Bildschirm, das ebenso erscheint, wenn der Benutzer den Datei Knopf im Datei Objekt gedrückt hat:



Diese Action kann dazu benutzt werden, wenn das Datei Objekt unsichtbar ist, der Bediener aber programmgesteuert nach einem Datei Namen gefragt wird.

D/A Objekt

Die neue Action **Other D/A commands** kann dazu benutzt werden, spezielle Eigenschaften von A/D-Boards auszunutzen.

Beispielsweise ist es möglich, die Clockrate der Keithley DAS1800 Serie zu takten (steuern), indem man folgende Action verwendet:

```
1) Other D/A commands  D/A1          keyword="DAGate",
                                     value="+"
```

Math Objekt

Die neue mathematische Funktion **randomize()** setzt keine Argumente voraus, und gibt auch nur eine 0 zurück, aber setzt TestPoints Zufallsgenerator auf einen Wert, der von der Systemzeit des Computers bestimmt wird, somit bekommt man eine neue Serie von Zufallszahlen, sobald das Programm losläuft (anstatt jeweils derselben Serie, die sich immer vom Start TestPoints wiederholt). Dies kann sehr hilfreich sein bei Aufgaben der Prozeßkontrolle oder Sample Applikationen.

Index

Notizen

Notizen

Notizen