ActiveDiagram

Version 2.02.01

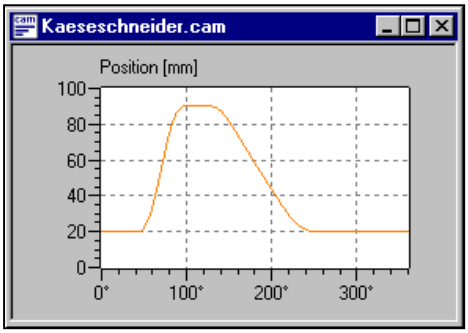
Inhaltsverzeichnis

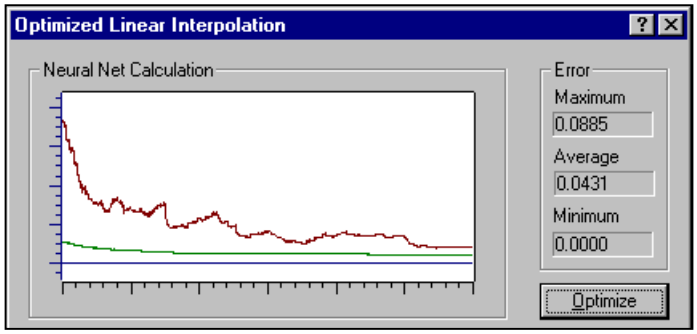
Einleitung

Immer wieder werden Ingenieure, Entwickler und Wissenschaftler mit dem Problem der grafischen Darstellung von Daten konfrontiert. Moderne Office-Programme wie Excel oder mathematische Tools wie MapleV bieten schon relativ viel Komfort für den Anwender. In der Regel bleibt aber die Darstellung der Daten auf die internen Funktionen der jeweiligen Programme beschränkt. Das Problem, grafische Darstellungen in eigene Applikationen einzubinden, wird damit nicht gelöst.

Mit *ActiveDiagram* bieten wir unseren Anwendern eine Komponente in modernster Softwaretechnologie. Als ein ActiveX kann man eine grafische Darstellung in beliebige Entwicklungen einbinden. Dabei können die Grafiken in Dialoge oder in eigenständige Fenster integriert werden. *ActiveDiagram* ist ein Control, das eine Zeichenfläche für die Darstellung von X-Y-Abhängigkeiten definiert. Es gibt eine X-Achse, der bis zu 8 Y-Achsen zugeordnet werden können. Über eine große Anzahl von Parametern kann die Darstellung den Anforderungen angepasst werden. Nach der Vorgabe der Grenzen für die Achsen wird die Darstellung automatisch skaliert. Die X-Achse kann wahlweise als „linear fest“, „linear verschieblich“ oder „logarithmisch fest“ parametriert werden. Bei einer verschieblichen Parametrierung bekommt die Darstellung die Charakteristik eines Kurvenschreibers, bei dem die Aufzeichnung von rechts nach links durchläuft, wenn das Maximum der X-Werte überschritten wird. Zusätzlich kann die X-Achse auch als eine Modulo-Achse definiert werden, die nach dem Erreichen einer festgelegten Periodenlänge wieder mit 0 beginnt (z.B. Aufzeichnung rotatorische Bewegungen von 0...360 Grad).

Das eigentliche Zeichnen der Daten in das Diagramm erfolgt mit Methodenaufrufen für das Setzen der X- und der Y-Werte (SetX, SetY, SetYX). Der Zeichenvorgang wird mit dem Setzen des Y-Wertes ausgeführt, der für die Abszisse den X-Wert des vorangegangenen SetX verwendet. Der X-Wert wird bis zum nächsten SetX gemerkt.





ActiveDiagram beinhaltet eine zweifache Datenspeicherung. Erstens werden alle Kurven in eine Bitmap geschrieben, die bei einem Neuzeichnen des Fensters auf den Bildschirm kopiert wird. Diese Methode verkürzt den Bildaufbau besonders bei großen Datenmengen beträchtlich. Zweitens kann für jede Kurve ein internes Datenarchiv aktiviert werden, das in Form einer Liste alle Daten aufzeichnet. Mit Hilfe dieser Liste können Datenaufzeichnungen einfach gezoomt und gescrollt werden. Auch ein Rücklesen aus dem Archiv oder Auswertungen mit Leselinien (maximal je 2 Leselinien in X- und in Y-Richtung) sind möglich.

Um die Programmierung von Applikationen noch weiter zu vereinfachen, geben Events Mitteilungen über interne Aktivitäten, die in ihrer Wirkung vom Anwender beeinflusst werden können. Integriert sind auch komplette Funktionsblöcke, wie beispielsweise das Selektieren eines Bereiches mit einem Gummiband-Rechteck.

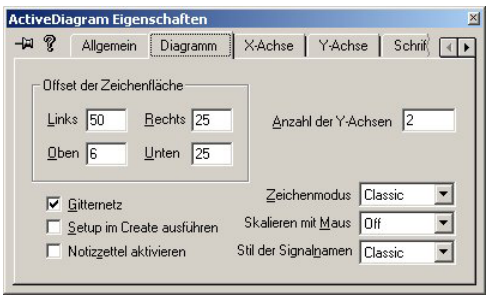
Jedes *ActiveDiagram* wird mit einer eindeutigen Seriennummer in einer Lizenzdatei ausgeliefert. Diese Lizenzdatei ist nur für den Entwicklungsprozess erforderlich und darf **nicht** mit der entwickelten Applikation weitergegeben werden. Diese Seriennummer wird in der Applikation gespeichert und kann im Bedarfsfall abgerufen werden.

Der angezeigte Inhalt des *ActiveDiagram* kann als Bitmap in die Zwischenablage kopiert werden . Ein Auslesen als Enhanced Metafile ist ebenfalls möglich. Diese Funktionen geben dem Anwender verschiede Möglichkeiten, den Inhalt von *ActiveDiagram* in Dokumente einzufügen und bei Bedarf auszudrucken.

Seit der Version 2.0 enthält ActiveDiagram ein Dual-Interface, das neben der standardmäßigen Dispatch-Schnittstelle ein VTable-Binding ermöglicht, wodurch die Performance des Datenaustauschs zwischen Client und Server stark gesteigert wird.

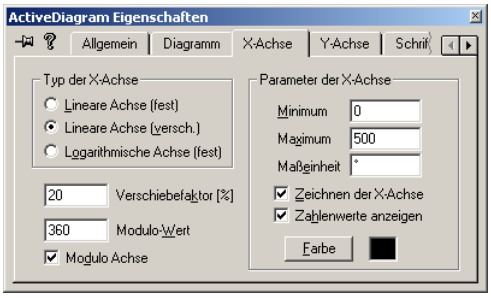
Eigenschaften, Methoden und Events

Allgemeine Eigenschaften des Diagramms



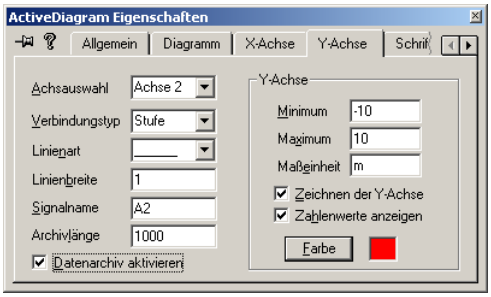
- Farbe von Hintergrund, Zeichenfläche, Leselinien
- Gitternetz (ein/aus) und Farbe des Gitternetzes
- Größe der Marker (Kreise, Kreuze, ...)
- Textfont für alle Ausschriften
- Abstände der Zeichenfläche von den Rändern
- Anzahl der Y-Achsen
- Setup-Bedingung, Notizzettel, Zeichenmodus

Eigenschaften der X-Achse



- Art der Achse (fest, verschieblich, logarithmisch)
- Minimum, Maximum
- Maßeinheit der Achse (an Zahlenbeschriftung angehängt)
- Farbe der Achse und der Beschriftung
- Flag (ein/aus) Achse zeichnen und Zahlenbeschriftung
- Wert des Verschiebungsfaktors (%)
- Modulo-Achse (ein/aus) und Periodenlänge

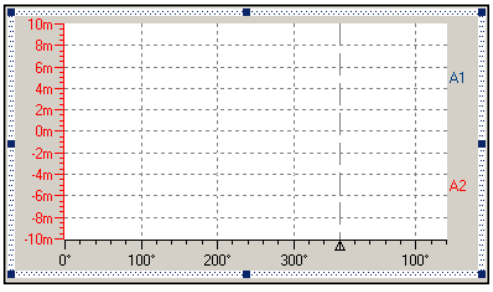
Eigenschaften der Y-Achsen



- Minimum, Maximum
- Maßeinheit der Achse (an Zahlenbeschriftung angehängt) und Signalname der Achse/Kurve
- Farbe der Achse und der Beschriftung
- Flag (ein/aus) Achse zeichnen und Zahlenbeschriftung
- Verbindungstyp der Kurve (Linie, Stufe)
- Datenarchiv (ein/aus), Anzahl der Datensätze
- Linienart und Linienbreite
- Bitdarstellung über projektierbare Bit-Achsen

Methoden des Diagramms

Beispiel im TestContainer



- Auswahl der aktiven Y-Achse
- Setzen der X- und der Y-Werte einer Kurve
- Setzen von Markern (Kreise, Kreuze,...)
- Löschen des gesamten Diagramms und Rücksetzen einzelner Achsen
- Umrechnen von physik.Werten in Pixel und umgekehrt
- Setzen des Modus einer Y-Achse (kontinuierliches Zeichnen, Blockmodus)
- Setzen des Mausmodus (Zoomrechteck, Leselinie)
- Ändern der Position und/oder der Größe des Controls
- Neuzeichnen aus Archiv, Lesen des Zeigers auf Archiv
- Kopieren der Grafik als Bitmap in Zwischenablage und Auslesen als Metafile
- Aktivieren von bis zu 4 Leselinien
- Verschieben/Dehnen/Stauchen von Kurven mit der Maus

Events des Diagramms

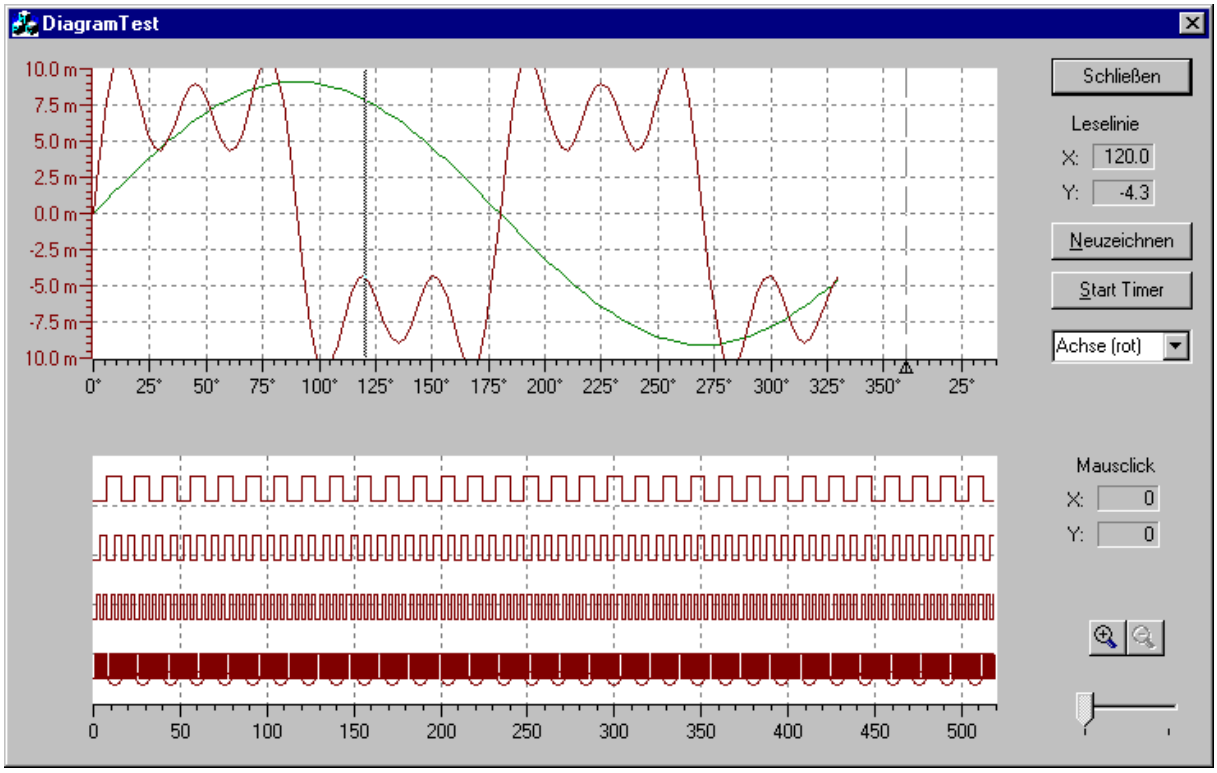
- Maus-Events werden an Container durchgereicht
- Event nach Abschluss des Setups und nach Löschen der Zeichenfläche
- Event als Reaktion auf Zoomrechteck und Leselinie

Dialogbasierte Applikation

Für das Integrieren von *ActiveDiagram* in einen Dialog wählt man im Ressourceneditor den Menüpunkt „ActiveX-Steuerelement einfügen ...“ und wählt den Eintrag „ActiveDiagram“. Nach dem Anpassen der Größe und dem Ausrichten der Lage im Dialog werden über die Eigenschaftsseiten von *ActiveDiagram* alle erforderlichen Parameter eingestellt (Siehe vorherige Seite).

Danach wird im „Klassenassistent“ der gewählten ID des *ActiveDiagram* eine Member-Variable zugefügt. Beim ersten Einfügen erstellt der Klassenassistent eine Wrapper-Klasse (Mantelklasse), über die alle Zugriffe auf das *ActiveDiagram* erfolgen.

Die Source für das Beispiel „DiagramTest“ ist im Anhang zu finden. Das Beispiel demonstriert einen Großteil der Funktionen von *ActiveDiagram*.



In der oberen Hälfte wird eine Kurvenschreiberfunktion demonstriert, die eine verschiebliche Modulo-X-Achse mit einer Periodenlänge von 360° enthält. Es gibt zwei Y-Achsen in den Farben ‚rot‘ und ‚grün‘. Die rote Achse ist an ein Archiv gekoppelt und die grüne nicht. Mit der Taste „Start Timer“ kann man das Einschreiben der Daten im 100 ms Takt aktivieren. Eine Besonderheit gibt es dabei bei der roten Kurve. Diese Kurve wird abwechselnd im **kontinuierlichen** und im **Blockmodus** gezeichnet. Damit soll nur demonstriert werden, dass ein Wechsel der Modi und sogar auch ein zwischenzeitliches Neuzeichnen aus dem Archiv (Taste „Neuzeichnen“) keine Verluste von Datenpunkten oder Zeichenunregelmäßigkeiten zur Folge haben. Beim Neuzeichnen wird die grüne Kurve nicht gezeichnet, weil für sie kein Archiv angelegt wurde.

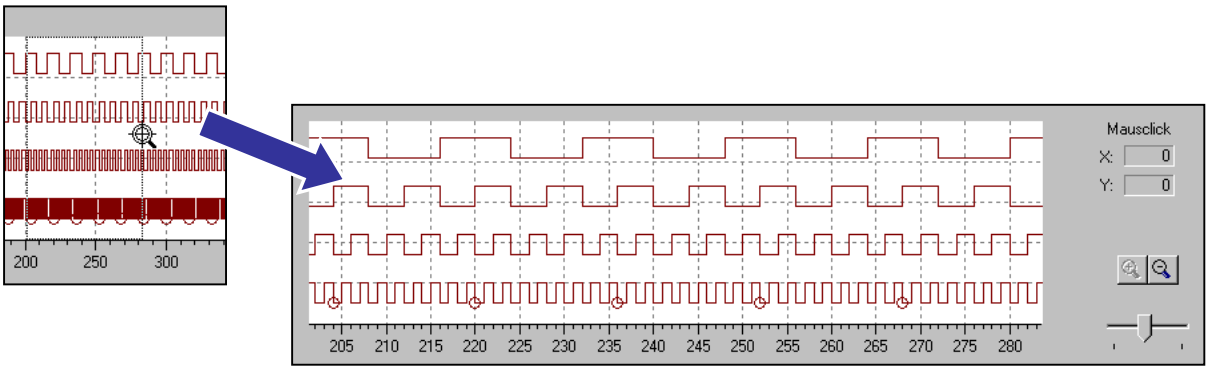
Bemerkt werden muss allerdings, dass der Blockmodus in diesem Beispiel genau so eingesetzt wird, wie es nicht getan werden sollte. Der Blockmodus dient dem Beschleunigen des Zeichenvorganges durch das Einschreiben der Daten in eine Polylinie, die dann mit dem nachfolgenden DRAW-Befehl komplett gezeichnet wird (SetYMode). Das heißt, Einschalten des Blockmodus, Einschreiben der Daten und der abschließende Zeichenbefehl sollten in einer Routine ohne Verzögerung abgearbeitet werden. Das hier vorliegende Beispiel soll die Robustheit der Funktionen demonstrieren, was durch diese ungewöhnliche Abfolge der Befehle gezeigt werden kann. Außerdem muss beachtet werden, dass ein aktiver Blockmodus das Neuzeichnen aus dem Archiv blockiert.

Wenn der Timer gestoppt ist, wird eine Leselinie angezeigt, die im Beispiel immer an der Position des letzten Datensatzes dargestellt wird. Man kann über die Leselinie die Daten aus dem Archiv auslesen

und anzeigen. Jede Bewegung der Leselinie generiert einen Event „ZoomRect“, der in „xstart“ die Pixelposition der Leselinie übergibt. Dieser Wert wird in den physikalischen X-Wert transformiert, von dem aus der nächstliegende Wert im Archiv gesucht wird. Die Archivdaten werden schließlich in den zugehörigen Textfenstern angezeigt.

Im unteren Fenster sind vier Kurven zu sehen, die eine Bitfolge darstellen sollen. Die Y-Achsen werden nicht gezeichnet. Alle Kurven sind an ein Archiv gekoppelt. Als Verbindungstyp wurde die „Stufe“ gewählt, wodurch die Bitcharakteristik erreicht wird. Die X-Achse überstreicht einen Definitionsbereich von 0...520. Mit der Zoomtaste „Lupe +“ kann über die Maus ein Bereich ausgewählt der gedehnt dargestellt werden soll. Der Typ des Zoomrechtecks ist auf XZOOM = 2 eingestellt. Im gezoomten Zustand kann über den Slider ein Scrollen der Kurven erfolgen. Mit der „Lupe -“ kann die Darstellung auf die Gesamtansicht zurückgeschaltet werden.

Die unterste Kurve von den Bitfolgen enthält zusätzlich kreisförmige Marker, die ebenfalls im Archiv gespeichert werden. Ein Click mit der linken Maustaste innerhalb des *ActiveDiagram* liefert einen Maus-Event, dessen Pixelkoordinaten in physikalische Werte umgerechnet und in Textfenstern angezeigt werden.



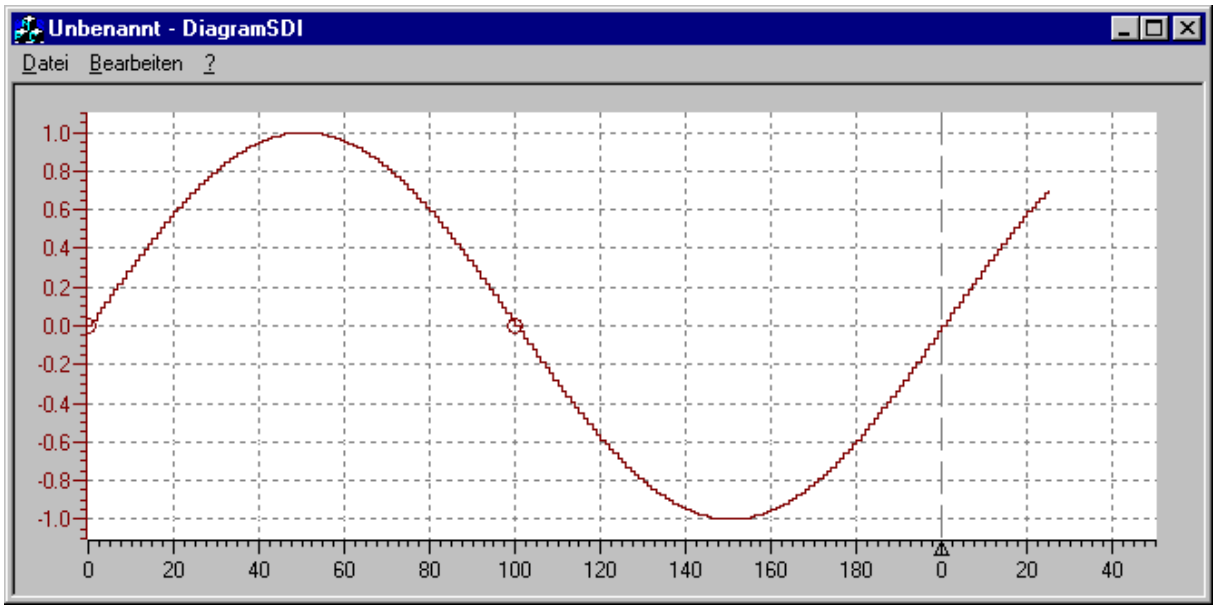
Windowbasierte Applikation

Im Gegensatz zur dialogbasierten Applikation wird in einer windowbasierten Applikation das *ActiveDiagram* dynamisch durch Aufrufen der Funktion „Create“ angelegt. Außerdem müssen alle relevanten Eigenschaftswerte innerhalb des Programmcodes gesetzt werden. Ein direktes Setzen der Parameter über die Eigenschaftsseiten ist nicht möglich. In vielen Fällen wird es außerdem erforderlich sein, die Größe des Controls an die Größe des Parentwindows anzupassen. Dazu muss in der Message „OnSize“ die Methode „MoveControl“ des *ActiveDiagram* aufgerufen werden.

Einen weiteren Unterschied gibt es auch beim Erzeugen der Wrapper-Klasse. In einer windowbasierten Applikation muss die Wrapper-Klasse über die Menüpunkte „Projekt / Dem Projekt hinzufügen / Komponenten und Steuerelemente...“ erzeugt werden.

Das Beispiel in der Anlage demonstriert eine SDI-Applikation, in die ein *ActiveDiagram* eingebunden ist. Im Diagramm wird eine einzelne Kurve mit dem Verbindungstyp „Stufe“ gezeichnet, deren Daten durch einen Timer-Event fortgeschrieben werden. Der Kurve ist ein Datenarchiv zugeordnet.

Wenn bei einer Größenänderung über „MoveControl“ auch die Größe des Controls geändert wird, erfolgt eine Neuskalierung der Achsen und ein Löschen der Zeichenfläche. Das *ActiveDiagram* signalisiert diesen Zustand über den Event „BitmapCleared“. Dabei wird ein Pointer auf eine Variable übergeben, über die das nachfolgende Zeichnen aus dem Datenarchiv gesteuert werden kann. Diese Variable ist mit dem Wert 1 vorbesetzt. Bleibt dieser Wert unverändert, wird das Zeichnen aus dem Archiv heraus durchgeführt.



Lizenzierung beim dynamischen Erzeugen

Beim Anlegen eines ActiveX in einer dialogbasierten Applikation wird im Ressourceneditor eine Instanz des ActiveX angelegt. Der Container ruft über das ActiveX den Lizenzstring aus der Lizenzdatei ab und speichert den Lizenzstring in den Initialisierungsdaten für den Dialog. Zur Laufzeit überprüft das ActiveX den Lizenzstring des Containers auf Gültigkeit.

Dieser Mechanismus funktioniert beim dynamischen Erzeugen nicht, da zur Entwurfszeit weder eine Instanz angelegt wird, über die ein Lizenzstring abgerufen werden kann, noch eine Möglichkeit vorgesehen ist, diesen Lizenzstring zu speichern. Im standardmäßigen „Create“-Aufruf wird als Lizenzstring der Wert NULL übergeben, was dazu führt, dass versucht wird, die Lizenzdatei des ActiveX zu lesen. Dieser Versuch muss aber in einer Lieferversion scheitern, da eine Weitergabe der Lizenzdatei verboten ist.

Als eine Lösung bleibt nur die direkte Übergabe des Lizenzstrings im „Create“-Aufruf. Doch auch da gibt es eine Reihe von Schwierigkeiten. Normalerweise kennt nur das ActiveX selbst die Lage und die Länge des Strings in der Lizenzdatei. Außerdem arbeiten verbesserte Lizenzmechanismen nicht mit den recht primitiven Standardalgorithmen von Microsoft sondern integrieren Verschlüsselungsprozeduren und eindeutige Seriennummern.

Ausgehend von diesen Schwierigkeiten wurde die Methode „ReadLicense“ entwickelt, die den Lizenzstring automatisch extrahiert und als zusätzlichen Service auch noch eine komfortable Projektierung des *ActiveDiagram*, so wie es in Dialogen üblich ist, ermöglicht. Dazu muss allerdings ein Dummy-Dialog in das Projekt eingefügt werden, der das *ActiveDiagram* als ein Control beinhaltet. In diesem Dialog kann auch das *ActiveDiagram* über die Eigenschaftsseiten parametriert werden. Der Methode „ReadLicense“ werden die ID’s des Dialogs und des *ActiveDiagram* übergeben. Die Methode ermittelt den Lizenzstring und eine Speicherdatei (CMemFile), in der die Parametrierung enthalten ist. Beide Daten werden dem „Create“-Aufruf übergeben.

Der Quellcode von „ReadLicense“ und die Anwendung dieser Methode ist in den Beispielen enthalten.

Microsoft hat in der „Knowledge Base“ zwei Artikel zu diesem Thema veröffentlicht:

- PRB: Dynamic Creation of Redistributable Control Fails
- LICREQST.EXE: Sample Requesting a License Key from an Object

Installation

Für eine Integration von *ActiveDiagram* müssen die Dateien

- **ActiveDiagram.ocx** DLL mit ActiveX
- **ActiveDiagram.tlb** Typbibliothek
- **ActiveDiagram.lic** Lizenzierung

in ein Zielverzeichnis kopiert und die Registrierung von *ActiveDiagram* vorgenommen werden. Die Registrierung erfolgt durch Aufruf von ‚Regsvr32.exe ActiveDiagram.ocx'. Das Programm ‚Regsvr32' ist Bestandteil von Windows und befindet sich im Windows-Verzeichnis bzw. im Systemverzeichnis von Windows. Beim Aufruf des Programms müssen die entsprechenden Pfad und Laufwerksangaben mit übergeben werden.

Bei einer Weitergabe einer Applikation, die *ActiveDiagram* enthält, darf nur die DLL (Active-Diagram.ocx) und **nicht die Lizenzdatei** mitgeliefert werden.

Zum Test werden die Beispielprojekte *DiagramTest* und *DiagramSDI* für Visual C++ , Vers. 6.0 mitgeliefert. Einen Großteil der Funktionen kann in den Beispielen ausprobiert und im Quellcode nachvollzogen werden.

Update auf neue Version

Für den Fall, dass *ActiveDiagram* bereits in einer bestehenden Applikation eingebunden war und eine neue Version mit ergänzenden Funktionen genutzt werden soll, muss eine neue Wrapper-Klasse (Mantelklasse) erstellt werden.

Leider unterstützt Visual C++, Vers. 6.0 diese Problemstellung nicht, so dass eine Reihe von Schritten durchgeführt werden müssen, um eine neue Klassengenerierung zu erreichen.

1. Öffnen des *Klassenassistenten* (Ctrl+W) in der Registerkarte *Member-Variablen*. Löschen der Variablen vom Datentyp *ActiveDiagram* (typischerweise vom Typ *CDiagram*).
2. Öffnen des *Klassenassistenten* (Ctrl+W) in der Registerkarte *Nachrichtenzuordnungstabelle*. Löschen der Nachrichten, die sich auf Events von *ActiveDiagram* beziehen.
3. Öffnen der Registerkarte *File-View* im Fenster *Arbeitsbereich*. Löschen der Header- und Sourcedateien, die für *ActiveDiagram* generiert wurden (typischerweise *Diagram.h* und *Diagram.cpp*).
4. Löschen der beiden Dateien aus dem Verzeichnis des aktuellen Projektes mit dem Explorer von Windows95 / NT / 2000.
5. Öffnen der Projektdatei *.dsp mit einem Texteditor. Suchen nach dem Klassennamen der Wrapper-Klasse (z.B. CDiagram) und löschen der entsprechenden Sektionen.

```
# Section DiagramTest : {90B0EF54-AB30-11D2-B344-0010A4006793}
#      2:21:DefaultSinkHeaderFile:diagram.h
#      2:16:DefaultSinkClass:CDiagram
# End Section
# Section DiagramTest : {90B0EF52-AB30-11D2-B344-0010A4006793}
#      2:5:Class:CDiagram
#      2:10:HeaderFile:diagram.h
#      2:8:ImplFile:diagram.cpp
# End Section
```


Einbinden über das Dual-Interface

In *ActiveDiagram* ist ein zusätzliches Interface *_DualDiagram* integriert, dass über die VTable angesprochen werden kann, eine höhere Performance im Datenaustausch besitzt und ein Zufügen einer Wrapper-Klasse unnötig macht. Um dieses Interface vom Client heraus ansprechen zu können, muss die Typbibliothek *ActiveDiagram.tlb* importiert werden. Da im Interface von *ActiveDiagram* Schnittstellen zu Fonts (IFont) und Bildern (IPicture) enthalten sind, muss zusätzlich ein Import dieser Definitionen aus der Typbibliothek *stdole2.tlb* erfolgen. Um doppelte Definitionen zu vermeiden, müssen gegebenenfalls einige Ausdrücke ausgeschlossen werden (exclude-Anweisung). Die Pointer auf die Interfaces werden als sogenannte Smart-Pointer definiert.

Definitionen

```
//-----------------------------------------------------------------------------------------
//  Einfügen der TypeLibs für das DualInterface

#import "stdole2.tlb" exclude("OLE_XSIZE_HIMETRIC", "OLE_XPOS_PIXELS", "OLE_HANDLE", \
                              "OLE_YSIZE_HIMETRIC", "OLE_YPOS_PIXELS", "OLE_COLOR"  )
using namespace stdole;

#import "ActiveDiagram.tlb"
using namespace DIAGRAMLib;
//-----------------------------------------------------------------------------------------
_DDiagramPtr          m_pDDiagram;          // Pointer auf primäres Dispatch-Interface
_DualDiagramPtr       m_pDualDiagram;       // Pointer auf Dual-Interface
```

Implementierung

```
//-----------------------------------------------------------------------------------------
//  Ermitteln des Pointers auf das Dual-Interface

m_pDDiagram = GetDlgItem(IDC_ACTIVEDIAGRAM)->GetControlUnknown();
m_pDualDiagram = m_pDDiagram;               // Ermitteln des Pointers

m_pDualDiagram->XMinimum = 0.1;             // Beispielhaftes Setzen der X-Skalierung
m_pDualDiagram->XMaximum = 333.0;
//-----------------------------------------------------------------------------------------
```

Probleme mit der Fehlerbehandlung in ActiveX

Nach den Designrichtlinien von Microsoft (*Behandeln von Fehlern in Ihrem ActiveX-Steuerelement*) werden falsche Parametrierungen von Eigenschaften über **COleControl::ThrowError** behandelt. Dabei müssen folgende Probleme beachtet werden:

1. Der Programmcode im Container, der nach dem Aufruf der Set-/Get-Routine liegt, die den Fehler auslöst, wird **nicht** bearbeitet.
2. Werden die Set-/Get-Routinen innerhalb eines Events des Controls aufgerufen, erfolgt **keine** Anzeige eines Fehlerdialogs. Im Ausgabefenster wird aber in jedem Fall die Exception angezeigt.

```
Nicht abgefangene Ausnahme in DiagramTest.exe (KERNEL32.DLL): 0xE06D7363: Microsoft C++
```

Während einer laufenden Debugsitzung kann man im Menü **Debug / Ausnahmen** die Exception 0xE06D7363 aktivieren (**Immer anhalten**). Dadurch kann man einen generellen Programmabbruch bei Fehlern, die im ActiveX mit ThrowError behandelt werden, erreichen.

Eigenschaften des Diagramms

Hintergrundfarbe

```
OLE_COLOR GetBackColor();
void SetBackColor(OLE_COLOR);
```

Font für alle Ausschriften an den Achsen

```
LPFONTDISP GetFont();
void SetFont(LPFONTDISP);
```

Farbe für den Hintergrund der Zeichenfläche

```
unsigned long GetPaperColor();
void SetPaperColor(unsigned long);
```

Abstände der Zeichenfläche von den Rändern

```
short GetOffsetLeft();
void SetOffsetLeft(short);
short GetOffsetTop();
void SetOffsetTop(short);
short GetOffsetRight();
void SetOffsetRight(short);
short GetOffsetBottom();
void SetOffsetBottom(short);
```

Gitternetz aktivieren / deaktivieren

```
BOOL GetGridEnable();
void SetGridEnable(BOOL);
```

Farbe des Gitternetzes

```
unsigned long GetGridColor();
void SetGridColor(unsigned long);
```

Anzahl der zugeordneten Y-Achsen

```
short GetNumberYAxis();
void SetNumberYAxis(short);
```

Auswahl der angezeigten Y-Achse

```
short GetYAxisSelection();
void SetYAxisSelection(short);
```

Position der Leselinie in Pixelwerten

```
short GetCursorLine();
void SetCursorLine(short);
```

Größe der Markierungen (Kreis, Kreuz,...)

```
short GetMarkerSize();
void SetMarkerSize(short);
```


Text für ungültige Diagrammgröße (Invalid)

Die Darstellung von *ActiveDiagram* benötigt je nach Projektierung eine Mindestgröße, um die Achsen und die Kurven sinnvoll darstellen zu können. Wird diese Mindestgröße unterschritten, wird anstelle des Diagramms standardmäßig der Text ‚Invalid' angezeigt. Dieser voreingestellte Textstring kann mit ‚SetTextInvalid' abgeändert werden.

Als Sonderfunktion kann anstelle des Textes eine vordefinierte Bitmap angezeigt werden. Um diese Bitmap anzuzeigen, muss als TextString ‚**$BITMAP**' übergeben werden.

Eine Abfrage des aktuellen Zustandes ist mit der Methode ‚IsSizeValid' möglich. Außerdem wird beim Übergang einer gültigen auf eine ungültige Größe der Event ‚SizeValid' ausgelöst. Man muss beachten, dass sowohl das Auslösen des Events als auch die Aktualisierung des Zustandsflags erst beim jeweils nächsten Zeichenvorgang erfolgt.

```
CString GetTextInvalid();
void SetTextInvalid(LPCTSTR);
```

Modus für das Zeichnen des Gitters

In den bisherigen Versionen wurden Gitter und Kurven in verschiedene Speicher-Bitmaps gezeichnet, die bei einer Zeichenaufforderung auf den Bildschirm kopiert werden. Diese Methode hat den Vorteil, dass beim Ein- und Ausschalten des Gitters oder beim Wechsel der selektierten Y-Achse bei eingeschaltetem Gitter kein Neuzeichnen der Kurven erforderlich ist. Es ist allerdings von Nachteil, dass durch das mehrfache Kopieren und Maskieren von Bitmaps ein hoher Rechenaufwand notwendig ist. Besonders beim Verschieben/Dehnen/Stauchen von Kurven und bei den verschieblichen Achsen ist bei eingeschaltetem Gitter ein deutlich größere Verzögerung wahrnehmbar als ohne Gitter. Aus diesem Grund wurde die Eigenschaft ‚SpeedDraw' eingeführt, die zwei zusätzliche Möglichkeiten für die Behandlung von Zeichenoperationen mit dem Gitter ermöglicht.

- DIAG_SDRAW_CLASSIC = 0: Defaulteinstellung. Behandlung wie oben beschrieben. Das Gitter liegt hinter den Kurven. Beim Ein- und Ausschalten des Gitters ist kein Neuzeichnen notwendig.
- DIAG_SDRAW_GRIDINFRONT = 1: Das Gitter wird direkt auf den Bildschirm gezeichnet. Das Gitter liegt vor den Kurven. Beim Ein- und Ausschalten des Gitters ist kein Neuzeichnen notwendig.
- DIAG_SDRAW_FAST = 2: Das Gitter wird beim Löschen der Speicher-Bitmap für die Kurven zuerst in diese Bitmap gezeichnet. Das Gitter liegt hinter den Kurven. Beim Ein- und Ausschalten des Gitters ist ein Neuzeichnen notwendig.

Die Modi 1 und 2 bewirken ein deutlich schnelleres Zeichnen. Bevorzugt sollte der Modus 2 eingesetzt werden, der in Zusammenhang mit den verbesserten Archivfunktionen keine Nachteile bringt. Lediglich bei verschieblichen Kurven kann es zu einem Pixelversatz der horizontalen Gitterlinien kommen.

```
short GetSpeedDraw();
void SetSpeedDraw(short);
```

Setup im Create ausführen

Die Setup-Funktion von *ActiveDiagram* wurde bis Version 2.01 beim ersten Zeichen ausgeführt. In Anwendungen, bei denen eine Visualisierung des Controls nicht direkt erfolgt kann dieses Verhalten nachteilig sein. Deshalb kann jetzt das ‚Setup' wahlweise im ‚Create' aufgerufen werden. Damit wird der Event ‚SetupFinished' auch dann ausgelöst, wenn das Control nicht gezeichnet wird.

Die Eigenschaft *SetupInCreate* ist defaultmäßig auf FALSE gesetzt, was einem Aufruf beim ersten Zeichnen entspricht.

```
BOOL SetupInCreate;
```


Eigenschaften der X-Achse

Grenzen der X-Achse

Wenn das Maximum kleiner oder gleich dem Minimum ist, wird beim Skalieren von linearen Achsen das Maximum auf Minimum + 100 gesetzt. Logarithmische Achsskalierungen sind nur für positive Werte definiert. Wenn das Minimum kleiner als 0 sein sollte, wird das Minimum auf 1 gesetzt. Wenn das Maximum kleiner oder gleich dem Minimum ist, wird beim Skalieren das Maximum auf Minimum * 100 gesetzt.

```
double GetXMinimum();
void SetXMinimum(double);
double GetXMaximum();
void SetXMaximum(double);
```

Text für Maßeinheiten nach Zahlenausschrift

```
CString GetXText();
void SetXText(LPCTSTR);
```

Farbe der X-Achse inklusive der Zahlen

```
unsigned long GetXColor();
void SetXColor(unsigned long);
```

Auschriften der X-Achse (Zahlen, Maßeinheit) aktivieren / deaktivieren

```
BOOL GetXTextEnable();
void SetXTextEnable(BOOL);
```

Zeichnen der X-Achse aktivieren / deaktivieren

```
BOOL GetXDrawEnable();
void SetXDrawEnable(BOOL);
```

Periodische Achsanzeige (modulo) aktivieren / deaktivieren

```
BOOL GetXModuloEnable();
void SetXModuloEnable(BOOL);
```

Wert für Periode einer Modulo-Achse

```
double GetXModuloValue();
void SetXModuloValue(double);
```

Verschiebbare Achse aktivieren / deaktivieren (Oszilloskopfunktion)

```
BOOL GetXMoveEnable();
void SetXMoveEnable(BOOL);
```

Wert für die Verschieblichkeit [0..100%]

```
short GetXMoveValue();
void SetXMoveValue(short);
```


Logarithmische Achsanzeige aktivieren / deaktivieren

Logarithmische Achsskalierungen sind nur für positive Werte definiert. Wenn das Minimum kleiner als 0 sein sollte, wird das Minimum auf 1 gesetzt. Wenn das Maximum kleiner oder gleich dem Minimum ist, wird beim Skalieren das Maximum auf Minimum * 100 gesetzt.

Die Granularität der Zahlenanzeige an der Achse hängt von Größe des verfügbaren Zeichenbereiches ab. Es muss zumindest soviel Achslänge vorgesehen werden, dass die Zehnerpotenzen ausgeschrieben werden können.

```
BOOL GetXLogEnable();
void SetXLogEnable(BOOL);
```

Eigenschaften der Y-Achse

Der Index der Y-Achse [0..9] ist die Variable **number**.

Grenzen der Y-Achse

Wenn das Maximum kleiner oder gleich dem Minimum ist, wird beim Skalieren das Maximum auf Minimum + 100 gesetzt.

```
double GetYMinimum(short number);
void SetYMinimum(short number, double newValue);
double GetYMaximum(short number);
void SetYMaximum(short number, double newValue);
```

Text für Maßeinheiten nach der Zahlenausschrift

```
CString GetYText(short number);
void SetYText(short number, LPCTSTR lpszNewValue);
```

Farbe der Y-Achse inklusive der Zahlen

```
unsigned long GetYColor(short number);
void SetYColor(short number, unsigned long newValue);
```

Auschriften der Y-Achse (Zahlen, Maßeinheit) aktivieren / deaktivieren

```
BOOL GetYTextEnable(short number);
void SetYTextEnable(short number, BOOL bNewValue);
```

Zeichnen der Y-Achse aktivieren / deaktivieren

```
BOOL GetYDrawEnable(short number);
void SetYDrawEnable(short number, BOOL bNewValue);
```

Verbindungstyp für die Punkte

Werte für die Variable *nNewValue*

- DIAG_CONNECT_LINE = 0: Linienverbindung
- DIAG_CONNECT_STEP = 1: Stufenverbindung

```
short GetYConnectType(short number);
void SetYConnectType(short number, short nNewValue);
```


Datenarchiv für die Y-Achse aktivieren / deaktivieren

```
BOOL GetYArchiveEnable(short number);
void SetYArchiveEnable(sho rt number, BOOL bNewValue);
```

Zeiger auf Datenarchiv für die Y-Achse

Der Zeiger auf das Archiv (CDataList*) wird als (long) übergeben.
Die Definitionen der Datentypen sind in der Datei „DiagramDef.h“.

```
long GetYArchive(short number);
```

Länge des Datenarchivs für die Y-Achse

Das Datenarchiv zur internen Datenverwaltung wird als reines RAM-Archiv angelegt. Um ein unkontrolliertes Überlaufen des Archivs zu verhindern, wird die maximale Länge als Parameter definiert. Der voreingestellte Wert ist **1000** Datensätze. Beim Überschreiben der maximalen Länge, werden die ältesten Datensätze gelöscht. Die mögliche Archivlänge kann auf Werte von 10 bis maximal 10.000 gesetzt werden.

```
short GetYArchiveLength(short number);
void SetYArchiveLength(short number, short nLength);
```

Text für Bezeichnung einer Y-Achse / Kurve

Für die Y-Achse / Kurve kann ein Name vergeben werden, der rechts neben der Zeichenfläche ausgeschrieben wird. Der Anwender muss den rechten Rand (`SetOffsetRight`) ausreichend breit definieren, damit der Text vollständig sichtbar ist. Die Höhe rechts neben der Zeichenfläche wird gleichmäßig auf die auszuschreibenden Texte aufgeteilt.

```
CString GetYAxisName(short number);
void SetYAxisName(short number, LPCTSTR lpszNewValue);
```

Linienart für eine Y-Kurve

Die dargestellten Kurven werden standardmäßig als durchgezogene Linien dargestellt. Zum Hervorheben spezieller Kurven oder für eine bessere Unterscheidung auf Schwarz-Weiss-Druckern kann man den Stil der Kurven ändern. Es muss berücksichtigt werden, dass bei Linienbreiten > 0 und gestricheltem Linienstil (d.h. nicht PS_SOLID) alle Kurven mit einem sogenannten ‚geometrischen’ Stift gezeichnet werden. Diese geometrischen Stifte haben die Eigenschaft, dass sie ihre Dicke und ihr Strich-Lücken-Verhältnis in logischen Koordinaten berechnen und somit ihr Aussehen an die jeweilige Skalierung anpassen. Bei Linienbreiten = 0 (Defaultwert) wird ein sogenannter ‚kosmetischer’ Stift verwendet, der die Kurven unabhängig von der Skalierung immer in der Pixelbreite 1 zeichnet. Diese Stiftart wird in auch einigen Zeichentools als ‚Haarlinie’ bezeichnet. Weiterhin wird ein gewählter Linienstil nur dann exakt wiedergegeben, wenn die Kurve in ihrer Gesamtheit gezeichnet wird (Zeichnen im BLOCK-Modus oder Zeichnen aus dem Archiv heraus). Im kontinuierlichen Modus kann der eingestellte Stil nicht garantiert werden (Siehe *SetYMode*). Als Parameter für die Linienart sind die folgenden Werte zulässig:

```
#define PS_SOLID            0
#define PS_DASH             1       /* -------  */
#define PS_DOT              2       /* .......  */
#define PS_DASHDOT          3       /* _._._._  */
#define PS_DASHDOTDOT       4       /* _.._.._  */
```

```
short GetYPenStyle(short number);
void SetYPenStyle(short number, short nNewValue);
```


Liniebreite für eine Y-Kurve

Die Breite der dargestellten Kurven kann ebenfalls geändert werden. Die standardmäßige Breite ist 0, was dem Zeichnen einer ‚Haarlinie' entspricht.

```
short GetYPenWidth(short number);
void SetYPenWidth(short number, short nNewValue);
```

Abstand für Anzeige von Markern

Die Eigenschaft ‚YMarkerShowDistance' legt einen X-Abstand in Pixel fest. Wird dieser Abstand zwischen 2 Markern unterschritten, werden alle Marker dieser Kurve nicht gezeichnet. Diese Abstandskontrolle funktioniert nur beim Zeichnen aus dem **Archiv** heraus.

Der Defaultwert ist 6. Wenn der Wert auf 0 gesetzt wird, werden immer alle Marker unabhängig von ihrem Abstand gezeichnet.

```
short GetYMarkerShowDistance(short number)
void SetYMarkerShowDistance(short number, short nDistance)
```

Bitdarstellung

Die Y-Achsen erhalten eine neue Eigenschaft, mit der sie zu einer Bit-Achse werden. Diese Achsen verwalten das Zeichnen von Bitspuren. In einem *ActiveDiagram* kann jede Achse als Bit-Achse projektiert werden. Diese Achsen können genau wie andere, beliebige Y-Achse ausgewählt und skaliert werden. Zusätzlich haben die Bit-Achsen noch weitere, „bitspezifische" Eigenschaften. Das Zeichnen der Bitspuren setzt voraus, dass die Kurven, die als Datenquellen projektiert sind, die Kurvenwerte in ihrem **Datenarchiv** gespeichert haben. Dabei kann sogar die Bit-Achse selbst als Datenquelle genutzt werden. Wenn kein Datenarchiv projektiert ist, werden die jeweiligen Bitspuren nicht angezeigt. Ist ein Datenarchiv projektiert und werden die Daten in ‚BitmapCleared' übergeben, wird zusätzlich zu den Bitspuren auch noch die Kurve dargestellt. Die Anzeige der Kurve kann man in diesem Fall verhindern, wenn man ‚SetYMode' = 3 setzt (kein Direktzeichnen – nur Achiveintrag).

Eine beliebige Y-Achse wird über das Setzen des Flags ‚YBitAxis' zu einer Bit-Achse. Gleichzeitig mit dem Setzen des Flags wird der Anzeigebereich so skaliert, dass alle projektierten Spuren ‚YBitTracks' im kompletten Bereich dargestellt werden, und das Flag ‚YBitActive' für die Darstellung der Bitkurven ebenfalls gesetzt wird.

```
BOOL GetYBitAxis(short number)
void SetYBitAxis(short number, BOOL bBitAxis)
```

Über die Grenzen der Y-Achse kann der Anzeigebereich variiert werden. Es wird jeder darzustellenden Spur eine „virtuelle" Höhe von 1.0 zugeordnet. Wenn beispielsweise 32 Bitspuren komplett im verfügbaren Zeichenbereich dargestellt werden sollen, muss die Bitachse die Skalierung 0..32 erhalten. Bei einer Skalierung von 16..32 werden nur die höherwertigen 16 Bit dargestellt. Skaliert man dagegen 0..64 werden alle 32 Bit in der unteren Hälfte des Anzeigebereiches dargestellt.

Die Anzahl der Bitspuren wird über ‚YBitTracks' projektiert. Es wird der Zeichenbereich für die vorgegebene Anzahl der Bitspuren reserviert. Diese Reserverierung ist unabhängig davon, ob der jeweiligen Spur bereits Bitdaten zugeordnet sind oder nicht. Bei einem Setzen des Wertes werden die Grenzen so gesetzt, dass alle Spuren im kompletten Bereich angezeigt werden. Das Flag ‚YBitActive' für die Darstellung der Bitkurven wird ebenfalls gesetzt.

```
short GetYBitTracks(short number);
void SetYBitTracks(short number, short nTracks);
```

Das Verhältnis von Spurhöhe zum Spurzwischenraum in Prozent wird über ‚YBitRatio' eingestellt (Defaultwert 80%). Wenn man sich die „virtuelle" Achsskalierung vorstellt, wird die Bitspur 0 bei einem

Wert von 80% im Zahlenbereich von 1,1 bis 1,9 dargestellt. Die 20% Zwischenraum werden gleichmäßig oberhalb und unterhalb der Spur aufgeteilt.

```
short GetYBitRatio(short number);
void SetYBitRatio(short number, short nPercent);
```

Jeder Bitspur kann ein Text für die Beschriftung zugeordnet werden. Der Text wird rechts ausserhalb des Zeichenbereiches dargestellt. Die Defaulteinstellung für den Text ist ‚Bit 0' bis ‚Bit 31'. Als Farbe für den Text und als Farbe für die Bitspur wird die Farbe der Quellkurve verwendet. Weiterhin werden für die Bitspur auch der Linienstil und die Linienbreite der Quellkurve eingesetzt.

```
CString GetYBitText(short number, short nBit);
void SetYBitText(short number, short nBit, LPCTSTR sText);
```

Eine Bitspur wird nur angezeigt, wenn ihr ein Bit aus den Daten einer Kurve zugewiesen wurde. Der Parameter ‚nTrack' ist die Nummer der Spur von *0..YBitTracks-1*. Die Datenquelle wird durch ‚nCurve' *[0..NumberYAxis-1]* und ‚nBit' *[0..31]* beschrieben. Eine Bitspur, der zugehörige Text am rechten Rand und die Beschriftung an der Achse werden nicht gezeichnet, wenn ‚nCurve' und/oder ‚nBit' ungültige Werte enthalten. Man kann somit durch Setzen auf *nBit = -1* das Zeichnen einzelner Spuren abschalten.

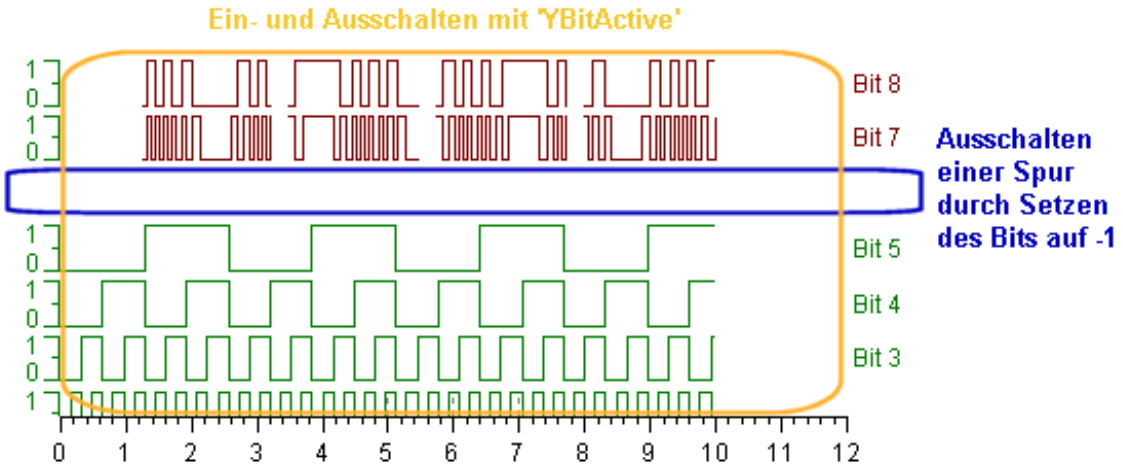
```
void SetYBitData(short number,short nTrack, short nCurve, short nBit)
```

Die Anzeige aller Bitdaten kann über ‚YBitActive' ein- oder ausgeschaltet werden. Das Setzen der Eigenschaft auf TRUE bewirkt ein Zeichnen von allen projektierten Bitspuren und das Anzeigen der Bitachse als selektierte Achse. Ein Setzen auf FALSE löscht die gezeichneten Bitspuren, wobei die Bitachse selektiert bleibt.

```
void SetYBitActive(short number, BOOL bActive);
```

Das untenstehende Bild zeigt ein Beispiel einer Bitanzeige. Für die Bitdarstellung wurden 10 Spuren projektiert, von denen nur die Spuren 3..8 vollständig angezeigt werden. Für die Spuren 0..5 wurden die Daten der Bit-Achse zugewiesen (Farbe der Bitkurve stimmt mit der Farbe der Achse überein). Bei den Spuren 7..9 wurde als Datenquelle die Kurve 0 zugewiesen. Bei der Spur 6 wurde als Bitwert –1 eingetragen, wodurch diese Spur komplett ausgeschaltet ist.

Wenn der Wert von ‚YBitActive' auf FALSE gesetzt wird, werden Bitdaten in der Zeichenfläche ausgeschaltet. Die Bit-Achse bleibt weiterhin ausgewählt und die Achsbeschriftung und die Namen der Bitspuren am rechten Rand werden weiterhin angezeigt



Methoden des Diagramms

Setzen des X-Wertes

```
void SetX(double value);
```

Setzen des Y-Wertes und Zeichnen der Verbindung

```
void SetY(short number, double value);
```

Setzen des X- und des Y-Wertes und Zeichnen der Verbindung

```
void SetYX(short number, double dYValue, double dXValue);
```

Erweitertes Setzen von X/Y-Werten für Verbindungen und Marker

Die bestehenden Methoden zum Zeichnen von Kurven und zum Setzen von Markern wurden um zwei neue Methoden mit erweiterter Funktionalität ergänzt. Es können Kurvenzüge unterbrochen und bei Bedarf mit einer Stützpunktanzeige kombiniert werden. Der Parameter ‚nType' gibt die Art und Weise des Zeichenvorganges an.

Bit 0..3	Nummer des Symbols für den Marker (Werte von 0..8).
DIAG_TYPE_MARKER	= 0x10: Zeichnen eines Markers.
DIAG_TYPE_CURVE	= 0x20: Zeichnen des Kurvenzuges (wenn ein Punkt definiert ist, sonst Anfangspunkt setzen).
DIAG_TYPE_NEWCURVE	= 0x40: Anfangspunkt setzen. Beginn eines neuen Kurvenzuges.

Es sind Kombinationen aus den einzelnen Bits möglich. Die Bits 0..3 werden nur ausgewertet, wenn auch das Bit 4 gesetzt ist. Wenn die Bits 5 oder 6 nicht gesetzt sind, bleibt der bis dahin gesetzte Vorgängerpunkt unverändert. Wenn die Bits 5 und 6 gemeinsam gesetzt sind, ist die Funktion von Bit 6 dominant (d.h es wird kein Kurvenzug gezeichnet sondern ein Anfangspunkt gesetzt). Bei der Kombination Bit 4 und Bit 5 oder Bit 6 wird ein Kurvenpunkt gesetzt und ein Symbol gezeichnet.

Wenn die Datenübergabe an eine Kurve im BLOCK-Modus erfolgt (SetYMode = 1), wird beim Setzen eines neuen Anfangspunktes (Bit 6 = 1) der bisherige Kurvenzug sofort gezeichnet. Danach wird das Array für das Polygon gelöscht, die Kurve ist wieder im BLOCK-Modus und kann weitere Daten für einen neuen Kurvenzug aufnehmen. Das Zeichnen des letzten Kurvenzuges muss wie bisher über SetYMode = 2 gesteuert werden.

Beispiele:

nType = 0x20 :	Funktionen mit ‚SetY' und ‚SetYX' identisch.
nType = 0x40 :	Es wird ein neuer Anfangspunkt gesetzt
nType = 0x10 + nMarker :	Funktionen identisch mit ‚SetYMarker'
nType = 0x30 + nMarker :	fortlaufende Kurve mit Stützpunkt auf der Position

```
void  SetYExt (short number, double dYValue, short nType)
void  SetYXExt(short number, double dYValue, double dXValue,
      short nType)
```

Zeichnen eines Markers

Zulässige Werte für die Variable *type*

• DIAG_MARK_CIRCLE	= 0	Kreis	○
• DIAG_MARK_CROSS	= 1	Kreuz	×
• DIAG_MARK_VSTROKE	= 2	senkrechter Strich	|
• DIAG_MARK_HSTROKE	= 3	waagerechter Strich	–
• DIAG_MARK_PLUS	= 4	Pluszeichen	+
• DIAG_MARK_SQUARE	= 5	Quadrat	□
• DIAG_MARK_TRIUP	= 6	Dreieck mit Spitze nach oben	△

- DIAG_MARK_TRIDOWN = 7 Dreieck mit Spitze nach unten ▽
- DIAG_MARK_RHOMB = 8 Rhombus ◇

```
void SetYMarker(short number, double value, short type);
```

Löschen der Zeichenfläche und Rücksetzen aller Achsen

Es werden die Zeichenspeicher aller Kurven rückgesetzt, so dass die danach gesetzten Punkte Anfangspunkte neuer Kurven sind. Wenn ein Archiv projektiert ist, wird dieses nicht gelöscht. Es erfolgt aber ein Löschen der Zeichenfläche (Speicherbitmap).

```
void ClearAll();
```

Rücksetzen einer einzelnen Y-Achse

Es wird der Zeichenspeicher der jeweiligen Kurve rückgesetzt, so dass ein danach gesetzter Punkt der Anfangspunkt einer neuen Kurve ist. Wenn ein Archiv projektiert ist, wird dieses komplett gelöscht. Es erfolgt aber kein Löschen der Zeichenfläche (Speicherbitmap).

```
void ResetY(short number);
```

Setzen des Zeichenmodes einer Y-Achse

Werte für die Variable *mode*

- DIAG_YMODE_CONTIN = 0: kontinuierliches Zeichnen bei SetY
- DIAG_YMODE_BLOCK = 1: blockweises Zeichnen einschalten, Daten [x,y] werden als Polygonzug in einem Array der Größe *size* gespeichert
- DIAG_YMODE_DRAW = 2: Zeichnen des Polygons ausführen (danach immer im Modus 0)
- DIAG_YMODE_ARCHIVE = 3: Direktes Zeichnen ausgeschaltet. Es wird generell aus dem Archiv heraus gezeichnet.

```
void SetYMode(short number, short mode, short size);
```

Dialog mit Info über ActiveDiagram

```
void AboutBox();
```

Neuzeichnen des ActiveX

```
void Refresh();
```

Transformationsroutinen von Koordinaten in Pixel und umgekehrt

Koordinaten der Pixel beziehen sich lediglich auf den Bereich der Zeichenfläche

```
short GetXPixel(double value);
short GetYPixel(short number, double value);
double GetXVal(short pixel);
double GetYVal(short number, short pixel);
```

Koordinaten der Pixel beziehen sich auf die Gesamtgröße des Controls

```
short GetXPixelAbs(double value);
short GetYPixelAbs(short number, double value);
double GetXValAbs(short pixel);
double GetYValAbs(short number, short pixel);
```


Nicht für neue Anwendungen

Aktivieren einer Mausfunktion

Mit dieser Funktion können komplexe Mausfunktionen aktiviert werden. Dazu gehören Gummiband-Rechtecke (für Zoomfunktion) und Leselinien.

Bei einer Zoomfunktion wird der Mauszeiger auf die Form einer „Lupe“ gesetzt und alle Mausaktivitäten werden vom Control abgefangen (SetCapture). Wenn innerhalb der Zeichenfläche die linke Maustaste gedrückt wird, kann man durch die Bewegung der Maus einen rechteckigen Ausschnitt definieren. Mit dem Loslassen der Maustaste wird ein Event „ZoomRect“ gesendet, der die Koordinaten des Bereiches in Pixelwerten enthält und der Mausmodus wird ausgeschaltet (mode = 0).

Bei der Aktivierung einer Leselinie wird eine senkrechte Linie innerhalb der Zeichenfläche dargestellt, die mit Hilfe der Maus erfasst (linke Maustaste halten) und verschoben werden kann. Jede Verschiebung erzeugt einen Event „ZoomRect“, der in den X-Werten die Pixelposition der Leselinie enthält. Die Farben der Leselinie ist einstellbar.

Bei fehlerhafter Funktion gibt der Event „ZoomRect“ in allen Koordinaten -1 zurück.

Werte für die Variable *mode*

- 0 = OFF: Mausfunktion ausgeschaltet
- 1 = ZOOM: Zoomrechteck mit mit wählbaren X- und Y-Koordinaten
- 2 = XZOOM: Zoomrechteck mit wählbren X-Koordinaten (Y liegt auf Rand)
- 3 = CURSORLINE: Einschalten der Leselinie (Initialisierung in der Mitte der Fläche)

```
void SetMouseMode(short mode);
```

Ab der Version 1.2 werden zusätzlich 4 Leselinien unterstützt. Es werden jeweils zwei Leselinien in X- und in Y-Richtung unterstützt. Die unteren 2 Bit im Parameter 'mode' werden für die bisherigen Funktionen verwendet. Die Bits 2..5 nutzen die neuen Funktionen für die Leselinien. Eine ausführliche Beschreibung der Funktion und die Rückmeldung über den Event 'ZoomRect' ist in der **Anlage 1** zu finden.

Ab der Version 2.0 können die 4 Leselinien (Bit 2..5) beim Zoom aktiv bleiben. In der Anwendung muss man dazu in der Maske für die Leselinien (Bit 2..5) zusätzlich das Bit 0 bzw. 1 je nach Art des Zoomrechtecks) setzen.

Beispiel:

```
je 1 Leselinie in X- und Y-Richtung      mode = 0x010100;
zusätzlich Zoomfunktion                  mode = mode | 1;
```

Weiterhin werden intern im *ActiveDiagram* die Leselinien in ihren physikalischen Werten gespeichert. Dadurch bleibt die Lage der Leselinien beim Umskalieren der Achsen und bei einer Größenänderung des Controls erhalten. Man muss beachten, dass die Leselinien auch außerhalb des definierten Zahlenbereiches zu liegen kommen können.

Ein automatisches Vorbesetzen der Leselinien auf die Mitte des Anzeigebereiches erfolgt beim ersten Setup, aber nur wenn sie nicht initialisiert wurden.

Größe und/oder Position des ActiveDiagram verändern

Dieser Aufruf verändert die Größe und/oder die Position des *ActiveDiagram*. Der typische Einsatz dieser Funktion ist eine Einbindung in die Message WM_SIZE der windowbasierten Applikation. Bei einer Größenänderung wird ein internes Neuskalieren vorgenommen, was ein Löschen der Zeichenfläche zur Folge hat.

```
void MoveControl(short left, short top, short right, short bottom);
```

Neuzeichnen der Daten aus dem Archiv heraus

Dieser Aufruf löscht die Zeichenfläche und ruft die Daten aus den Archiven der einzelnen Achsen ab, die dann neu gezeichnet werden. Die Zeichenfunktion ist optimiert, so dass nur die Datensätze gezeichnet werden, die auf der Zeichenfläche sichtbar sind.

Diese Methode darf niemals aus den Events ‚BitmapCleared' oder ‚SetupFinished' heraus aufgerufen werden.

```
void DrawArchive();
```

Kopieren der Bitmap des ActiveDiagram in die Zwischenablage

Dieser Aufruf kopiert den gesamten Inhalt des *ActiveDiagram* als Bitmap (DIB - device-independent bitmap) in die Zwischenablage.

Wenn die beiden Parameter *'width'* und *'height'* den Wert 0 besitzen oder eine Größe vorgegeben wird, die eine ungültige Zeichenfläche zur Folge hätte, wird das *ActiveDiagram* in der auf dem Bildschirm definierten Größe kopiert. Die Daten der Kurven werden aus der Speicherbitmap übernommen.

Anderenfalls wird das *ActiveDiagram* auf eine Bitmap entsprechend den Parametern umskaliert. Da die Kurven neu gezeichnet werden müssen, ist ein internes Archiv oder eine Behandlung des Events 'BitmapCleared' erforderlich. Die Hintergrundfarbe der Achsen wird auf 'weiß' gesetzt. Der Aufruf der Methode mit gesetzten Parametern ist eine einfache Möglichkeit zum Drucken des *ActiveDiagram*. Entsprechend der Auflösung des Druckers wird die Bitmap auf ein angepasstes Format vergrößert und kann aus der Zwischenablage heraus in den Device-Context des Druckers kopiert werden.

Es wird nicht empfohlen, die Pixelauflösung des Druckers als Parameter an das *ActiveDiagram* zu übergeben. Bei hochauflösenden Druckern würde die Bitmap sehr viel Speicher belegen und große Rechenzeiten bei der Darstellung benötigen. Vernünftige Größen liegen bei etwa 800 x 600. Die Größenanpassung an das Druckbild kann dann mit 'SetViewportExt' bzw. 'SetWindowExt' vorgenommen werden.

Über das Flag 'bCusorLine' kann man wählen, ob die Leselinien ebenfalls in die Bitmap gezeichnet werden sollen oder nicht.

```
void CopyToClipboard(short width, short height, BOOL bCursorLine);
```

Ab der Version 2.0 wird sichergestellt, dass beim Kopieren der Bitmap in die Zwischenablage immer der eingestellte Font genommen wird. Bisher wurde beim Kopieren aus einem ‚unsichtbarem' *ActiveDiagram*, das noch nicht wenigstens einmal ‚sichtbar' war, der Systemfont genommen.

Zeichnung als Enhanced Metafile auslesen

Es ist eine Methode ‚*GetPicture'* implementiert, die eine direkte Übergabe des Inhaltes des *ActiveDiagram* als Enhanced Metafile ermöglicht. Ein Aufruf dieser Methode ist auch bei unsichtbarem *ActiveDiagram* möglich (SW_HIDE).

```
LPPICTUREDISP GetPicture( OLE_HANDLE hDC, OLE_XSIZE_HIMETRIC width,
                          OLE_YSIZE_HIMETRIC height, short flag);
```

Die Methode benötigt zur Übergabe einen Handle *hDC* auf einen Referenz-Gerätekontext (wenn hDC = NULL ist, wird der Gerätekontext des Bildschirms als Referenz verwendet). Die Größenangaben des Bildes werden in HIMETRIC (0.01mm) übergeben. Bei einer Ausgabe in ein Rechteck von 18 cm Breite und 10 cm Höhe muss man width = 18000 und height = 10000 übergeben. Die Skalierung im *ActiveDiagram* erfolgt dabei so, dass der eingestellte Font für die Achsen in der richtigen Größe ausgegeben wird.

In der Methode *GetPicture* müssen die Daten für den Zeichenvorgang aktualisiert werden. Das Zeichnen kann dabei über den Event *BitmapCleared* erfolgen. Ein Aktualisierung aus dem Archiv heraus ist ebenfalls möglich. Es muss beachtet werden, dass auch bei verschieblich programmierter X-Achse ein Verschieben des Zeichenbereiches **nicht** erfolgt.

Der Zusatzschalter *flag* dient zur Steuerung des Ausgabeformates.

Bit 0: = 0: Hintergrund wird mit Farbe gefüllt
= 1: Hintergrund wird nicht mit Farbe gefüllt

Bit 1: = 0: Hintergrund wird mit weisser Farbe gefüllt (Bit 0 muss 0 sein)
= 1: Hintergrund wird mit der Farbzuordnung der Bidschirmausgabe gefüllt (Bit 0 muss 0 sein)

Bit 2: = 0: Achsen, Gitter und Kurven in Farbzuordnung des Bildschirms gezeichnet
= 1: Achsen, Gitter und Kurven werden schwarz gezeichnet

Bit 3: = 0: Leselinien werden nicht gezeichnet
= 1: Leselinien werden gezeichnet

Bit 4: = 0: Der linke Offset wird nicht verändert
= 1: Der linke Offset wird automatisch an den Gerätekontext und an die Skalierung angepasst.

Linken Offset einer Y-Achse berechnen

Die Methode berechnet den für eine Y-Achse den notwendigen linken Offset und übergibt den Wert an den Client. Der Offsetwert beinhaltet die Länge der Skalierungsstriche, den Zwischenraum zu den Zahlen und die Breite der Zahlenausgabe. Für eine korrekte Berechnung müssen die Größe des *ActiveDiagram* und die Skalierungswerte (Min, Max) für die jeweilige Y-Achse gesetzt sein. Durch einen nachfolgenden Aufruf von ‚*SetOffsetLeft'* kann die Breite für das Zeichnen der Y-Achse an die jeweilige Skalierung angepasst werden.

Die Y-Achse wird virtuell so gezeichnet, wie es die Schalter (*YTextEnable, YDrawEn*able) für diese Achse angeben. Aus dieser virtuellen Zeichnung wird danach der Offset ermittelt.

Wenn für den Parameter ‚*nYNumber'* der Wert –1 angegeben wird, wird die selektierte Achse als Bezug verwendet.

```
short GetYOffsetLeft(short nYNumber);
```

Abfragen auf gültige Größe

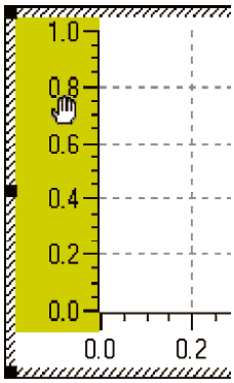
Die Darstellung von *ActiveDiagram* benötigt je nach Projektierung eine Mindestgröße, um die Achsen und die Kurven sinnvoll darstellen zu können. Wird diese Mindestgröße unterschritten, wird anstelle des Diagramms standardmäßig der Text ‚Invalid' angezeigt.

Eine Abfrage des aktuellen Zustandes ist mit der Methode ‚IsSizeValid' möglich. Außerdem wird beim Übergang einer gültigen auf eine ungültige Größe der Event ‚SizeValid' ausgelöst. Man muss beachten, dass sowohl das Auslösen des Events als auch die Aktualisierung des Zustandsflags erst beim jeweils nächsten Zeichenvorgang erfolgt.

```
BOOL IsSizeValid();
```


Grafisches Verschieben/Dehnen und Stauchen von Kurven

Im Einsatz von ActiveDiagram ist es sinnvoll, einzelne Kurven mit Hilfe der Maus in Y-Richtung zu verschieben, zu dehnen oder zu stauchen. Wenn die Maus bei eingeschalteter Funktion den linken Randbereich überstreicht, wandelt sich der Mauszeiger zu einer „Hand“. Durch ein Klicken mit der linken Maustaste kann eine Stelle der Achse erfasst und nach unten oder oben verschoben werden.



Wird zusätzlich zum Mausklick noch die Control-Taste gedrückt, kann die eintsprechende Kurve gedehnt oder gestaucht werden. Ein Ziehen mit der Maus von außen zur Mitte zu bewirkt ein Stauchen und ein Ziehen von der Mitte nach außen ein Dehnen.

Die Funktion wirkt nur auf die ausgewählte Kurve. Es wird während der Funktionsausführung sowohl die Achsskalierung angepasst als auch die Kurve in den neuen Grenzen gezeichnet. Auch bei 8 Kurven mit 2000 Punkten, die sich im Archiv befinden, ergibt sich eine kaum merkbare Verzögerung im Bildaufbau. Man sollte aber unbedingt die Eigenschaft ‚SpeedDraw’ auf den Wert 1 oder 2 setzen und als Linienstil PS_COSMETIC verwenden (keine dicken, gestrichelten Linien).

Das Ein- und Ausschalten der Funktion wird mit einer Methode vorgenommen, bei der man über zwei Bits steuern kann, ob man nur eine der beiden Funktionen oder beide verwenden möchte. Das Ausschalten erfolgt durch Rücksetzen der beiden Bits (Defaulteinstellung: Bit 0 = 0, Bit 1 = 0).

DIAG_SCALE_MOVE = 1 (Bit 0 = 1): Verschieben aktiv

DIAG_SCALE_SIZE = 2 (Bit 1 = 1): Dehnen/Stauchen aktiv

```
void  ScaleByMouse(short nMode);
```

Wenn die Funktion beendet ist (Loslassen der Maustaste), wird ein **Event** ausgelöst, der das Ergebnis der Funktion übermittelt. Es werden die Nummer der beeinflussten Y-Achse, ihre neuen Grenzen und die Art der Veränderung als Parameter übergeben (*nMode* entspricht dem Wert, der in ‚ScaleByMouse’ übergeben wurde).

```
void  ScaleChanged(short number, double dMinNew, double dMaxNew,
      short nMode);
```

Waagerechte Begrenzungs- und Hilfslinien

Im *ActiveDiagram* sind pro Kurve zwei waagerechte Begrenzungslinien zuschaltbar. Die Linien können vom Anwender in ihrer Y-Position parametiert werden. Als zusätzliche Eigenschaften können die Farbe, die Linienbreite und der Linienstil eingestellt werden. Ein Verschieben der Linien mit der Maus ist nicht möglich. Als Linienart wird generell PS_COSMETIC verwendet (Damit sind keine dicken gestrichelten Linien möglich). In der Default-Einstellung haben die Hilfslinien die Farbe der dazugehörigen Kurve und sind 1 Pixel breit und gestrichelt (PS_DASH).

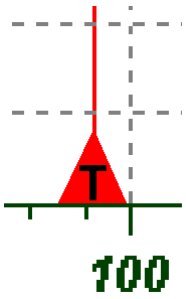
Die Default-Einstellung wird durch *SetYLineStyle(nYNumber, RGB(0,0,0), 0, 0)* wieder hergestellt.

Von der Zeichenreihenfolge her werden zuerst das Gitter, dann die Hilfslinien und zuletzt die Kurven gezeichnet. Die Linien liegen damit optisch unter den Kurven. Die Eigenschaft ‚SpeedDraw’ beinflusst die Bedingungen beim Zeichnen. Wird bei einer aktiven Begrenzungslinie die Anzahl der Y-Achsen verringert, so dass die zugehörige Achsnummer ungültig wird, erfolgt ein Deaktivieren der Begrenzungslinie (bVisible=FALSE).

```
void ShowYLine1( short nYNumber, double dYPos, bool bVisible)
void ShowYLine2( short nYNumber, double dYPos, bool bVisible)
void SetYLineStyle1( short nYNumber, unsigned long newColor,
                     short PenStyle, short PenWidth)
void SetYLineStyle2( short nYNumber, unsigned long newColor,
                     short PenStyle, short PenWidth)
```


Senkrechte Triggerlinie

Im *ActiveDiagram* kann eine senkrechte Triggerlinie gesetzt werden. Als Kennzeichnung für die Triggerlinie dient ein kleines Dreieck am unteren Rand, das zusätzlich einen Buchstaben enthalten kann. Die Linien können vom Anwender in ihrer X-Position parametiert werden. Als zusätzliche Eigenschaften können die Textfarbe, die Linienfarbe, die Linienbreite und der Linienstil eingestellt werden. Ein Verschieben der Linien mit der Maus ist nicht möglich. Als Linienart wird generell PS_COSMETIC verwendet (Damit sind keine dicken gestrichelten Linien möglich). In der Default-Einstellung haben die Hilfslinien die Farbe ‚rot', sind 1 Pixel breit und gestrichelt (PS_DASH). Die Textfarbe ist ‚schwarz'.



Die Größe des Dreiecks wird an die Größe des gewählten Fonts angepasst. Da das Dreieck möglichst klein gehalten wird, kann bei Unterlängen ‚g' oder extremen Breiten ‚W' ein komplettes Umschließen des Charakters nicht garantiert werden.

Von der Zeichenreihenfolge her werden zuerst das Gitter, dann die Triggerlinie und zuletzt die Kurven gezeichnet. Die Triggerlinie liegt damit optisch unter den Kurven. Die Eigenschaft ‚SpeedDraw' beinflusst die Bedingungen beim Zeichnen.

```
void ShowXLine1( double dXPos, char cSymbol, BOOL bVisible );
void SetXLineStyle1( unsigned long LineColor, short PenStyle,
                     short PenWidth, unsigned long TextColor );
```

Zoom-Funktion

Im *ActiveDiagram* kann die Methode *SetMouseMode* sowohl für Zoom-Funktionen als auch für Leselinien genutzt werden. Es handelt sich bei dieser Methode um eine Funktionalität, die aus Kompatibilität zu früheren Versionen beibehalten wird. Für neue Anwendungen wird empfohlen, die neuen, leistungsfähigeren Methoden für Zoom und Leselinien einzusetzen und auf *SetMouseMode* zu verzichten. Es ist keinesfalls empfehlenswert, die neuen Funktionen und *SetMouseMode* gemischt einzusetzen.

Um die Handhabung zu verbessern, wurde die Eigenschaft *ZoomMode* implementiert. Der Parameter *nMode* kann die Werte von 0..3 annehmen.

- DIAG_ZOOM_OFF = 0: Zoomfunktion ausgeschaltet
- DIAG_ZOOM_XY = 1: Zoomrechteck mit mit wählbaren X- und Y-Koordinaten
- DIAG_ZOOM_X = 2: Zoomrechteck mit wählbaren X-Koordinaten (Y liegt auf Rand)
- DIAG_ZOOM_PERM = 3: Permanenter Zoom

Bei nMode = 3 wird eine permanente Zoom-Funktion aktiviert. Immer wenn ein Click mit der Maus im innern der Zeichenfläche erfolgt, kann ein Rechteck aufgezogen werden, dessen Koordinaten beim Loslassen der Maustaste zum Ausführen einer Zoom-Funktion genutzt werden kann. Es wird nur dann ein Event generiert, wenn sich die Pixelpositionen von Drücken und Loslassen unterscheiden. Es wird nur der Event *ZoomRectNew* generiert.

```
short GetZoomMode();
void SetZoomMode( short nMode );
```

Messcursor / Leselinien

Die Funktionen für die Leselinien wurden in der Version 2.01.01 komplett überarbeitet. Es werden maximal 4 Leselinien unterstützt.

Farbe der aktiven Leselinie

Der Defaultwert für die Farbe der aktiven Linie ist ‚rot'.

```
unsigned long GetCursorColorA();
void SetCursorColorA(unsigned long);
```


Farbe der inaktiven Leselinien

Der Defaultwert für die Farbe der inaktiven Linie ist ‚blau'.

```
unsigned long GetCursorColorI();
void SetCursorColorI(unsigned long);
```

Breite der Leselinien

Der Defaultwert für die Breite ist 1. Da der Parkbereich eine Breite von 3 Pixel hat, muss man berücksichtigen, dass Linien, die breiter sind als 3 Pixel, in die Achs- bzw. Zeichenfläche hineinragen.

```
short GetCursorLineWidth();
void SetCursorLineWidth( short nWidth );
```

Linienstil der Leselinien

Die Defaultwert für den Linienstil PS_SOLID. Es wird der Stift PS_COSMETIC verwendet, wodurch keine dicken, gestrichelten Linien möglich sind.

```
short GetCursorLineStyle();
void SetCursorLineStyle( short nStyle );
```

Referenzachse der waagerechten Leselinien

Für die waagerechten Leselinien (2 und 3) muss die Referenzachse festgelegt werden. Diese Zuordnung kann auch über die Methode ‚SetCursorVal' erfolgen. Beim Umschalten des Wertes für ‚CursorYAxis' bleibt die Linie auf ihrer Pixelposition liegen und der physikalische Wert wird auf die neue Achse umgerechnet. Der Defaultwert ist Y-Achse = 0.

Bei den senkrechten Linien (0 und 1) hat der Befehl keine Wirkung. Ein Umschalten der Achsen führt nicht zum Neuzeichnen der Cursorlinien, da diese Aktion nicht sinnvoll ist.

```
void SetCursorYAxis( short nLineNumber, short nYNumber );
short GetCursorYAxis( short nLineNumber );
```

Position der Leselinien in Pixelwerten

Die Pixelwerte dieser Leselinien können gelesen und geschrieben werden. Wenn eine Leselinie beim ersten Zeichnen noch auf keinen Wert initialisiert war, wird sie auf die Mitte der Zeichenfläche gesetzt.

Wenn ein Parkbereich für die jeweilige Linie aktiviert ist, führt ein Unterschreiten des linken Randes (X-Richtung) bzw. ein Überschreiten des unteren Randes (Y-Richtung) zum Aktivieren des Parkzustandes (‚CursorParked'). Anderenfalls wird ein eventuell vorhandener Parkzustand deaktiviert. Ein Event (‚CursorState') wird dabei nicht generiert. Beim Lesen von Werten einer geparkten Linie erhält man die Pixelwerte, die der Parkposition entsprechen.

```
short GetCursorXY(short nLineNumber);
void SetCursorXY(short nLineNumber, short nNewValue);
```

Position der Leselinien in physikalischen Größen

Die vier möglichen Cursorlinien können nicht nur mit Pixelwerten sondern auch mittels physikalischer Größen gesetzt und gelesen werden. Der Methode muss die Nummer der Cursorlinie *nLineNumber* und bei den waagerechten Linien (2 und 3) auch die Nummer der Y-Achse *nYNumber* übergeben werden. Bei den senkrechten Linien (0, 1) wird der Wert von *nYNumber* nicht ausgewertet. Beim Auslesen beziehen sich die Werte auf die Y-Achse (Linien 2, 3), für die auch das Einschreiben erfolgte. Der Defaultwert ist die Y-Achse 0.

Ein Setzen eines Wertes mit *SetCursorVal* führt in jedem Fall zum Deaktivieren eines Parkzustandes. Beim Lesen von Werten einer geparkten Linie erhält man die physikalischen Werte, die der Parkposition entsprechen.


```
void SetCursorVal( short nLineNumber, short nYNumber, double dValue);
double GetCursorVal( short nLineNumber );
```

Bei der Berechnung der physikalischen Werte aus den Pixeln wird ein ‚intelligenter' Rundungsalgorithmus eingesetzt. Es wird überprüft, ob der Pixelwert der Cursorlinie mit dem Pixelwert eines Skalierungsstriches übereinstimmt. Sollte das der Fall sein, wird dem physikalischen Wert der Cursorlinie exakt der Wert des Skalierungsstriches zugewiesen. Dadurch erhält man beim Rücklesen einen physikalischen Zahlenwert, der auch dem grafischen Eindruck der Darstellung entspricht. Weiterhin ist sichergestellt, dass beim vergrößerten Auslesen des Bildes die Cursorlinie pixelgenau auf ihrer physikalischen Position bleibt.

Aktivieren der Leselinien in der Parkposition

Es können sowohl in X- als auch in Y-Richtung beide Linien unabhängig von einander aktiviert und angezeigt werden. Das heisst, es kann die Linie mit der Nummer 1 aktiviert und bearbeitet werden, obwohl die Linie 0 inaktiv ist.

Es wird am linken Rand (für X-Linien) und am unteren Rand (für Y-Linien) ein Parkbereich von 3 Pixel Breite realisiert. Um die Breite dieses Parkbereiches werden die Y-Achsen nach links und die X-Achse nach unten verschoben. Aus diesem Parkbereich können jeweils maximal 2 Messcursorlinien entnommen werden. Das Anlegen der Parkbereiche und das Anzeigen der Cursorlinien im Parkbereich erfolgt durch *SetCursorActive*. Wenn die Linie 0 oder 1 aktiv ist, wird der linke Parkbereich reserviert. Wenn die Linie 2 oder 3 aktiv ist, wird der untere Parkbereich reserviert. Gleichzeitig mit der Aktivierung wird die Cursorposition auf inaktiv (parken) und der Zustand des Cursors auf freigeschaltet (*CursorEnable* = *TRUE*) gesetzt. Der Defaultwert ist FALSE.

Wird *bActive* = FALSE übergeben, wird immer auch *CursorEnable* auf FALSE gesetzt. Wenn sich der Cursor zum Zeitpunkt der Deaktivierung auf der Parkposition befand, wird der Cursor von der Parkposition auf die Pixelpostion des jeweiligen Randes (links oder unten) gesetzt.

```
void SetCursorActive( short nLineNumber, BOOL bActive );
BOOL GetCursorActive( short nLineNumber );
```

Abfragen und Setzen des Parkzustandes

Der Parkzustand der Linien kann über die Eigenschaft *CursorParked* abgefragt werden. Wenn beim Setzen der Eigenschaft der Parkbereich *CursorActive* für die jeweilige Linie aktiviert ist, kann der Cursor eingeparkt oder ausgeparkt werden. Ist der Parkbereich nicht aktiviert, wird die Linie in jedem Fall den Zustand ‚Ausgeparkt' haben.

```
void SetCursorParked( short nLineNumber, BOOL bParked );
short GetCursorParked( short nLineNumber );
```

Ein- oder Ausschalten von Leselinien

Um die aktivierten Cursorlinien darzustellen oder zu verstecken, gibt es die Eigenschaft *CursorEnable*. Beim Ein- und Ausschalten dieses Flags bleiben der Zustand des Cursors (Parken oder nicht), die Position und die Aktivierung unverändert. Einfach gesagt, dieses Flag wirkt lediglich auf das Zeichnen und die Mausaktionen.

Die zuletzt eingeschaltete Linie ist auch die ausgewählte Linie (Farbe *CursorColorA*). Wird eine ausgewählte Linie ausgeschaltet, wird die eingeschaltete Linie mit dem höchsten Index zur ausgewählten Linie.

```
void SetCursorEnable( short nLineNumber, BOOL bEnable );
BOOL GetCursorEnable( short nLineNumber );
```


Event beim Verschieben der Leselinien

Der Event *CursorChanged* liefert die Veränderungen der Cursorlinien. Neben der Nummer Cursorlinie nLineNumber werden auch der physikalische Wert *dValue* und die Nummer der Y-Achse nYNumber (nur Linie 2, 3) übergeben.

```
void CursorChanged(short nLineNumber, short nYNumber, double dValue);
```

Event beim Ein-und Ausparken der Leselinien

Der Event *CursorState* zeigt das Ein- und Ausparken des Messcursors an. Events werden nur ausgelöst, wenn der Cursor mit der Maus in den Parkbereich hinein oder aus ihm heraus verschoben wird.

```
void CursorState( short nLineNumber, BOOL bParked );
```

Notizzettelfunktion

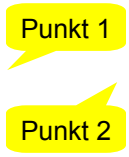
An jeder beliebigen Stelle im Zeichenbereich von *ActiveDiagram* kann der Anwender mit der Maus Hilfspunkte setzen und beschriften. Damit können Punkte oder Positionen für eine spätere Auswertung markiert werden.

Die Notizzettel werden über ein Objekt *INotice* realisiert. Die Objekte haben die Eigenschaften Nummer der Y-Achse, X- und Y-Position (physikal. Werte), Textstring, Farbe des Textes und des Hintergrundes. Beim Auslesen von Metafiles im S/W-Modus wird der Text *schwarz* und der Hintergrund *hellgrau* gezeichnet.

Die Notizzettel sind von ihren Y-Werten her immer einer Y-Achse zugewiesen. Beim Umskalieren dieser Achse werden auch die Notizzettel verschoben. Optisch bleiben sie damit immer exakt der gleichen Position dieser Kurven zugewiesen. Es können nur Objekte eingefügt werden, die mit ihren Koordinaten innerhalb der aktuellen Grenzen der X-/Y-Achsen liegen.

Bei aktivierter Funktion können Notizzettel über einen Doppelklick der Maus zugefügt werden. Ein Doppelklick auf eine bestehende Notiz löscht diese. Wenn mehrere Notizen übereinanderliegen, wird immer die Notiz mit dem niedrigsten Index gelöscht.

Die Notizzettel werden als abgerundetes Rechteck mit einem kurzen Pfeil, der auf die Position zeigt gezeichnet. Beim Zeichnen wird sichergestellt, dass die Notizzettel, deren Position innerhalb der aktuellen Grenzen liegt, immer komplett angezeigt werden. Vorzugsweise ist der Bezugspunkt links, unterhalb der Notiz angeordnet. Wenn die Notizen zu weit oben oder zu weit rechts liegen, wird der Pfeil nach oben bzw. nach rechts verschoben.



Die Größe des Notizzettels wird der Größe des Textes und des Fonts angepasst. Die voreingestellte Farbe des Hintergrundes ist ‚gelb' und die des Textes ‚schwarz'. Beim Einfügen mit der Maus wird als Text die Nummer des Index in der Collection genommen. Beim Einfügen – Löschen – Einfügen kann es vorkommen, dass gleiche Nummern mehrfach vorhanden sind.

Eigenschaften ‚INotice':

```
short YAxis;
double XPos;
double YPos;
BSTR Text;
OLE_COLOR TextColor;
OLE_COLOR BackColor;
```

Defaulteinstellung für die Farbe des Texten neuangelegterNotizen

```
OLE_COLOR NoticeTextColor;
```


Defaulteinstellung für die Farbe des Hintegrundes neuangelegter Notizen

```
OLE_COLOR NoticeBackColor;
```

Aktivieren und Deaktivieren aller Notizzettelfunktionen

Beim Deaktivieren werden alle Notizzettel gelöscht. Beim Löschen durch Deaktivieren wird kein Event ausgelöst.

```
void SetNoticeActive( BOOL bActive );
BOOL GetNoticeActive();
```

Abfragen der Anzahl der INotice-Objekte

```
short GetNoticeCount();
```

Interface eines INotice-Objektes mit dem gewählten Index

```
INotice* GetNoticeObject(short nIndex);
```

Neuanlegen von INotice-Objekten

Das Objekt wird auf die gewählte Position gesetzt (Pixel oder physikalische Werte). Dem Objekt werden die selektierte Y-Achse und die eingestellten Farben zugewiesen. Diese Funktion führt zum Auslösen eines *NoticeChanged*-Event.

```
void AddNotice(double dXpos, double dYpos, BSTR sText);
void AddNoticePix(short nXpos, short nYpos, BSTR sText);
```

Löschen des INotice-Objekte mit dem gewählten Index

Diese Funktion führt zum Auslösen eines *NoticeChanged*-Event.

```
void RemoveNoticeAt(short nIndex);
```

NoticeChanged-Event

Beim Erzeugen und Löschen von *INotice*-Objekten wird ein *NoticeChanged*-Event ausgelöst. Dieser Event wird sowohl beim Aufrufen der entsprechenden Methoden als auch bei den Mausfunktionen ausgelöst.

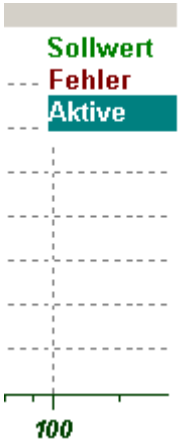
Beim Neuanlegen (*bAdded*=*TRUE*) ist das *INotice*-Objekt bereits instanziiert und kann in seinen Eigenschaften modifiziert werden. Das erste Zeichnen des Objektes erfolgt erst nach dem Event.

Beim Löschen ist das Objekt während des Event-Aufrufes noch vorhanden. Bei Bedarf können im Event noch die Eigenschaften des Objektes ausgelesen werden.

```
void NoticeChanged(short nIndex, boolean bAdded);
```


Beschriftung der Kurven

Es wird eine Eigenschaft definiert, mit deren Hilfe die Lage und das Design der Beschriftung von Kurven modifiziert werden kann. Defaultmäßig (*SignalNameStyle* = *0*) liegt die Beschriftung von Kurven rechts außerhalb der Zeichenfläche, die Signalnamen werden gleichmäßig in der Höhe verteilt und in der Farbe der Kurven gezeichnet. Wenn die Summe der Höhe aller Ausschriften die Höhe der Zeichenfläche übersteigt, erfolgt keine Ausschrift.



Bei SignalNameStyle = 1 erfolgt die Beschriftung der Kurven im Innern der Zeichenfläche am rechten, oberen Rand. Die Signalnamen der Kurven werden oben rechts beginnend zeilenweise in der Farbe der entsprechenden Kurve dargestellt. Wenn eine Achse keine Beschriftung enthält, d.h. der Textstring des Signalnamens ist leer, wird diese Achse übersprungen. Wenn der Text aller Achsen ‚leer' ist, erfolgt keinerlei Ausschrift.

Zur Kennzeichnung der ausgewählten Kurve wird der Hintergrund in der Farbe der Kurve und der Text in der Farbe der Zeichenfläche geschrieben. Damit entsprechen die Kontrastverhältnisse denen der Kurvendarstellung. Alle anderen Beschriftungen erfolgen transparent.

Die Größe des reservierten Bereiches wird über die maximale Pixellänge der auszuschreibenden Texte festgelegt. Der Text wird links ausgerichtet geschrieben.

Bei *SignalNameStyle* = 2 erfolgt die Beschriftung der Kurven im rechts außerhalb der Zeichenfläche, aber in der gleichen Darstellungsart wie bei *SignalNameStyle* = 1.

Bei den Bitachsen werden die Signalnamen nur bei *SignalNameStyle* = 1 gezeichnet.

DIAG_SIGNAL_CLASSIC = 0,

DIAG_SIGNAL_INSIDE = 1

DIAG_SIGNAL_OUTSIDE = 2

```
short SignalNameStyle;
```

Events des Diagramms

Betätigung der Maustaste

```
void MouseDown(short Button, short Shift, OLE_XPOS_PIXELS x,
               OLE_YPOS_PIXELS y);
```

Loslassen der Maustaste

```
void MouseUp(short Button, short Shift, OLE_XPOS_PIXELS x,
             OLE_YPOS_PIXELS y);
```

Verschieben der Maus

```
void MouseMove(short Button, short Shift, OLE_XPOS_PIXELS x,
               OLE_YPOS_PIXELS y);
```

Das interne Setup des ActiveX ist abgeschlossen

Meldung an den Container, dass alle notwendigen Initialisierungen abgeschlossen sind. Erst ab diesem Zeitpunkt nimmt *ActiveDiagram* Zeichnungsaufträge entgegen (SetY, SetYMarker). Wenn allerdings ein Archiv mit der Y-Achse verbunden ist, können Daten schon eher an *ActiveDiagram* übergeben werden. Das Zeichnen erfolgt allerdings erst nach diesem Event.

Für den Fall, dass die Variable *mode auf ihren voreingestellten Wert = 1 belassen wird, erfolgt ein Zeichnen aus dem Archiv. Eine Änderung dieses Wertes verhindert ein Zeichnen aus dem Archiv. Wenn kein Archiv vorhanden ist, hat der Wert dieser Variable keine Bedeutung.

```
void SetupFinished(short FAR* mode);
```


Zeichenfläche wurde gelöscht

Meldung an den Container, dass die Zeichenfläche gelöscht wurde und die Daten neu gezeichnet werden müssen. Wenn die Achsen mit einem Archiv verbunden sind, kann das Neuzeichnen von *ActiveDiagram* übernommen werden. Für diesen Fall muss die Variable *mode auf ihren voreingestellten Wert = 1 belassen werden. Eine Änderung dieses Wertes verhindert ein Zeichnen aus dem Archiv. Wenn kein Archiv vorhanden ist, hat der Wert dieser Variable keine Bedeutung.

```
void BitmapCleared(short FAR* mode);
```

Rückmeldung über Mausfunktion

Meldung an den Container, dass entweder ein Zoomrechteck ermittelt wurde oder eine Verschiebung der Leselinie stattfand. Die Position der Leselinie ist in xstart. Im Fehlerfall enthalten alle Koordinaten den Wert -1.

Ab der Version 1.2 werden zusätzlich 4 Leselinien unterstützt. Eine ausführliche Beschreibung der Rückmeldung über den Event 'ZoomRect' ist in der **Anlage 1** zu finden.

```
void ZoomRect(short xstart, short ystart, short xend, short yend);
```

ZoomRechteck / Leselinien

Um bei gleichzeitiger Nutzung von Zoomfunktion und Leselinien einen höheren Komfort für den Anwender zu bieten, wurden zwei neue Events eingeführt, die beide Funktionen trennen.

Der Event *ZoomRect* bleibt aus Gründen der Kompatibilität unverändert erhalten. Es wird ein neuer Event *ZoomRectNew* implementiert, der ausschließlich die Zoomfunktion bedient. Die Funktionalität dieses Events entspricht bezogen auf die Zoomfunktionen der von *ZoomRect*.

Ein weiterer Event *CursorChanged* liefert die Veränderungen der Cursorlinien. Neben der Nummer Cursorlinie *nLineNumber* werden auch der physikalische Wert *dValue* und die Nummer der Y-Achse *nYNumber* (nur Linie 2, 3) übergeben.

```
void ZoomRectNew(short xstart, short ystart, short xend, short yend);
void CursorChanged(short nLineNumber, short nYNumber, double dValue);
```

Gültigkeit der Größe geändert

Die Darstellung von *ActiveDiagram* benötigt je nach Projektierung eine Mindestgröße, um die Achsen und die Kurven sinnvoll darstellen zu können. Wird diese Mindestgröße unterschritten, wird anstelle des Diagramms standardmäßig der Text ‚Invalid' angezeigt.

Eine Abfrage des aktuellen Zustandes ist mit der Methode ‚IsSizeValid' möglich. Außerdem wird beim Übergang einer gültigen auf eine ungültige Größe der Event ‚SizeValid' ausgelöst. Man muss beachten, dass sowohl das Auslösen des Events als auch die Aktualisierung des Zustandsflags erst beim jeweils nächsten Zeichenvorgang erfolgt.

Das Flag ‚bValid' kennzeichnet den aktuellen Zustand (Größe gültig = TRUE).

```
void SizeValid(BOOL bValid);
```

Skalierung mit der Maus geändert

Die Methode ‚ScaleByMouse' ermöglicht ein Verschieben / Dehnen / Stauchen von Kurven mit Hilfe der Maus. Wenn die Umskalierung abgeschlossen ist und die Maustaste losgelassen wurde, wird der Anwender mit diesem Event über die Umskalierung informiert. Es werden die Nummer der beeinflussten Y-Achse, ihre neuen Grenzen und die Art der Veränderung als Parameter übergeben (*nMode* entspricht dem Wert, der in ‚ScaleByMouse' übergeben wurde).

```
void ScaleChanged(short nYNumber, double dMinNew, double dMaxNew,
                  short nMode);
```


Neues in Version 1.1

Änderungen gegenüber der Version 1.0

- Fehler im Blockmodus bei Version 1.0 beseitigt: Reservierter Speicher wurde nicht gelöscht
- In der Methode ‚**SetMode**' wird beim Verbindungstyp ‚Stufe' intern die doppelte Anzahl von Punkten angelegt.
- Der Event ‚**BitmapCleared'** hat einen Parameter bekommen, über den der Anwender die Zeichenfunktion über das interne Archiv steuern kann.

Neue Funktionen

- Neue Methode ‚**SetMouseMode**' (0..normal, 1..Zoom-Rechteck, 2..XZoom-Rechteck, 3..Leselinie) Auslösen des Events **ZoomRect(xstart, ystart, xend, yend)** in Pixelwerten. Mit dem Event wird intern der Modus auf 0 zurückgesetzt. Wenn das Rechteck ungültig ist (z.B. der MouseClick erfolgte außerhalb des Datenfeldes) liefert der Event in allen Positionen den Wert –1.
- Setzen von Daten (**SetX, SetY**)

  ActiveDiagram kann Daten erst nach Abschluss des Setup in der Zeichenfläche darstellen. Dieser Zustand wird durch den Event ‚**SetupFinished**' angezeigt (siehe unten). Für den Fall, dass eine Archivfunktion aktiviert ist, können die Daten auch eher (i.a. in der Initialisierungsphase) in das Control eingeschrieben werden. Die Daten werden im Archiv gespeichert und nach Abschluss der Initialisierungsphase auf dem Bildschirm dargestellt. Im Event ‚**SetupFinished**' darf der Wert der Variablen ***mode=1** nicht verändert werden, wenn ein Zeichnen aus dem Archiv erfolgen soll.
- Neue Eigenschaft ‚**SetYArchiveEnable**' / ‚**GetYArchiveEnable**' zum Aktivieren der Archivfunktion für eine einzelne Achse (Kurven und Marker)
- Neue Events zum Steuern der Archivfunktionen

  SetupFinished(*mode) – erstes Setup des Controls ist erfolgt (kann z.B. zum Einschreiben der Daten genutzt werden)

  BitmapCleared(*mode) – Zeichenfläche gelöscht

  nach beiden Routinen wird bei aktivem Archiv das Neueinschreiben der Daten realisiert.
- Beim Neuzeichnen der Daten über die Archivfunktion wird der Blockmodus (SetYMode=1) genutzt. Wenn sich eine oder mehrere Y-Achsen vom Anwender aus im Blockmodus befinden, wird das Neuzeichnen aus dem Archiv solange verzögert, bis alle Achsen den Blockmodus verlassen haben.
- Nach dem Aktivieren der internen Archivfunktion und nach dem Zeichnen mit ‚SetYMode' wird der Modus der jeweiligen Y-Achse wieder in den **kontinuierlichen Modus** (0) gesetzt.

Neues in Version 1.2

Änderungen gegenüber der Version 1.1

- Die Darstellung der Leselinie 'SetMouseMode(3)' ist an die neuen Leselinien der Version 1.2 angepasst worden. Die Schnittstellen zum Anwender sind unverändert. Damit ist diese Funktion intern kompatibel zu 'SetMouseMode(4)'.
- Bei den Zoomfunktionen 'SetMouseMode(1 bzw. 2)' wurde teilweise das Rechteck nicht pixelgenau gezeichnet. Das Problem wurde behoben und das Rechteck zeigt jetzt exakt die Koordinaten an, die auch beim Event berechnet und übergeben werden.
- Änderung der Form des Mauszeigers in einen Doppelpfeil, wenn sich die Maus im Fangbereich einer Leselinie befindet.
- Bei einer Änderung der Farbe der Zeichenfläche 'SetPaperColor' wurde der Farbwert erst bei einem neuen Setup übernommen. Um die Farbänderung sofort wirksam werden zu lassen, wird das Setup direkt bei Farbauswahl angestoßen.

- Die Eigenschaftsseiten wurden komplett überarbeitet. Es gibt jetzt neben einer Seite mit allgemeinen Eigenschaften des Diagramms jeweils eine Seite für die X- und für die Y-Achsen.
- Im Gegensatz zur Version 1.1 werden in der Version 1.2 die Minimum- und Maximumwerte der Achsen zusätzlich noch in den entsprechenden Eigenschaftsseiten überwacht und nicht nur in den entsprechenden Skalierfunktionen der Achsen. Wenn bei linearen Achsen das Minimum gößer oder gleich dem Maximum ist, wird das Maximum auf Minimum + 100 gesetzt.

  Bei logarithmischen Achsen muss das Minimum größer als 0 sein. Im Fehlerfall wird das Minimum auf 1 gesetzt. Wenn das Minimum größer oder gleich dem Maximum ist wird das Maximum auf Minimum * 100 gesetzt.
- Die Voreinstellung des Modulowertes bei Modulo-X-Achse ist 360. Der bisherige, voreingestellte Wert von 0 ist als Modulowert unzulässig. Eine eventuelle falsche Parametrierung wurde aber auch schon bisher im *ActiveDiagram* angefangen.

Neue Funktionen

- Es wurden jeweils 2 Leselinien in X- und Y-Richtung implementiert. Siehe Methode **'SetMouseMode'** bzw. **Anlage 1.**
- Zur Darstellung der Signalnamen hat jede Y-Achse / Kurve die Eigenschaft 'YAxisName' bekommen. Siehe **'SetYAxisName'**
- Der Inhalt des *ActiveDiagram* kann in Form einer DIB (device-independent bitmap) in die Zwischenablage kopiert werden. Je nach den Parametern erfolgt das Kopieren in der Originalgröße oder in einer angepassten Größe. Siehe **'CopyToClipboard'**
- Die X-Achsen können außer der linearen jetzt auch eine logarithmische Skalierung bekommen. Siehe **'SetXLogEnable'**
- Begrenzung der Archivlänge auf einen einstellbaren Wert (10...10000, Default 1000). Wenn das Archiv über die maximale Länge hinaus beschrieben wird, werden die ältesten Einträge gelöscht. Siehe **'SetYArchiveLength'**

Neues in Version 2.0

Änderungen gegenüber der Version 1.2

- Beim ersten Setup wurde bisher nur der Event **SetupFinished** erzeugt. Bei allen Veränderungen, die ein neues Setzen der Daten erfordern, erfolgte die Benachrichtigung über **BitmapCleared**. Diese Situation hatte zur Folge, dass bei den meisten Anwendungen beide Events mit identischer Funktionalität bedient werden mussten (Einschreiben der Daten in *ActiveDiagram*). Ab der Version 2.0 wird beim ersten Setup zusätzlich der Event **BitmapCleared** generiert. Dadurch kann man in vielen Anwendungen auf eine Behandlung der Nachricht **SetupFinished** verzichten. Wenn die Version 2.0 in älteren Applikationen Verwendung findet und man auf eine Anpassung des Verhaltens verzichten möchte, werden in der Regel keine Probleme auftreten. Es werden lediglich beim ersten Setup die Daten zweimal an das *ActiveDiagram* übergeben.
- Bei **extremen Skalierungen** kam es beim Setup zu einer stark verlängerten Rechenzeit, die man als Endlosschleife interpretieren konnte. Dieses Verhalten wurde beseitigt. Es können jetzt Skalierungen bis an die Grenzen der Genauigkeit von double ausgeführt werden, ohne dass die Rechenzeit ansteigt. Im Extremfall kommt es zu aufgrund der Skalierung zu Problemen in der grafischen Darstellung (es werden nicht mehr alle Zahlen an die Achsen geschrieben oder die Skalierungsstriche fallen aus), was aber zu keinem weiteren Fehlverhalten in der Reaktion zum Client führt.

  ```
  double delta = 10; // delta kleiner 100000 ging bisher nicht!
  double min  = 20000000000000.0;
  double max = min + delta;
  ```

  In den oben gezeigten Quellcodezeilen führte ein *delta* von 100000 zu dem genannten Problem. In der korrigierten Version 2.0 kann man bis *delta* von 10 gehen.

- Es gab in der bisherigen Version teilweise Probleme bei der Skalierung von **Logarithmischen Achsen**. Dieses Verhalten wurde untersucht, und es wurden Korrekturen im Code vorgenommen, die ein einwandfreies Skalieren der Logarithmischen Achsen erlauben.
- Die Methode ‚**CopyToClipboard'** wurde so abgeändert, dass sich das Auslesen der DIB identisch zum Metafile bei ‚GetPicture' verhält (beispielsweise exakte Fonteinstellungen bei unsichtbarem ActiveDiagram).
- Der Aufruf von ‚**GetReadyState'** liefert nach der Instanziierung den Wert READYSTATE_LOADING. Nach erfolgtem Setup, was durch den ersten Aufruf von ‚OnDraw' erfolgt, ändert sich der Status auf READYSTATE_COMPLETE (Meldung erfolgt durch Event ‚**ReadyStateChanged'**). Ab diesem Zeitpunkt können Daten zum Zeichnen an das ActiveDiagram übergeben werden. Beim Event ‚ReadyStateChange' wird entgegen der Standard-Definition in der ODL-Datei und entgegen der fehlerhaften Dokumentation von Microsoft der Zustand als Parameter übergeben. Eine Korrektur in den entsprechenden Dateien wurden vorgenommen.

Neue Funktionen

- In *ActiveDiagram* ist ein zusätzliches Interface *_DualDiagram* integriert (**Dual-Interface**), dass über die VTable angesprochen werden kann, eine höhere Performance im Datenaustausch besitzt und ein Zufügen einer Wrapper-Klasse unnötig macht. Ein weiterer Schritt zur Erhöhung der Performance im Datenaustausch ist die Methode **SetYX**, die ein Wertepaar übergibt.
- Es ist eine Methode **GetPicture** implementiert, die eine direkte Übergabe des Inhaltes des *ActiveDiagram* als Enhanced Metafile ermöglicht. Ein Aufruf dieser Methode ist auch bei unsichtbarem *ActiveDiagram* möglich.
- Die Methode **GetYOffset** berechnet den für eine Y-Achse den notwendigen linken Offset und übergibt den Wert an den Client. Der Offsetwert beinhaltet die Länge der Skalierungsstriche, den Zwischenraum zu den Zahlen und die Breite der Zahlenausgabe.
- Die dargestellten Kurven werden standardmäßig als durchgezogene Linien dargestellt. Zum Hervorheben spezieller Kurven oder für eine bessere Unterscheidung auf Schwarz-Weiss-Druckern kann man den Stil (**YPenStyle**) und die Breite (**YPenWidth**) der Kurven ändern.
- Es wurden einige Erweiterungen für die Leselinien vorgenommen. Bei Auswahl der Zoomfunktion können die Leselinien eingeschaltet bleiben. Auch nach der Veränderung der Skalierung und/oder der Größe des Controls bleiben die Leselinien auf ihren physikalischen Positionswerten liegen. Die Positionen der Leselinien können ab der Version 2.0 auch in physikalischen Werten geschrieben und gelesen werden (**GetCursorVal** / **SetCursorVal**). Ein neuer Event **CursorChanged** informiert den Anwender über eine Verschiebung der Leselinie.
- Um bei gleichzeitiger Nutzung von Zoomfunktion und Leselinien einen höheren Komfort für den Anwender zu bieten, wurden zwei neue Events eingeführt. Der Event **ZoomRectNew** bedient ausschließlich die Zoomfunktionen und der Event **CursorChanged** die Leselinien.

Neues in Version 2.00.02

Es wurden Eigenschaften, Methoden und Events hinzugefügt, die eine flexiblere Behandlung von ungültigen Zeichengrößen duch den Anwender ermöglichen. Die Darstellung von *ActiveDiagram* benötigt je nach Projektierung eine Mindestgröße, um die Achsen und die Kurven sinnvoll darstellen zu können. Wird diese Mindestgröße unterschritten, wird anstelle des Diagramms standardmäßig der Text ‚Invalid' angezeigt. Dieser voreingestellte Textstring kann mit ‚SetTextInvalid' abgeändert werden.

Als Sonderfunktion kann anstelle des Textes eine vordefinierte Bitmap angezeigt werden. Um diese Bitmap anzuzeigen, muss als TextString ‚$BITMAP' übergeben werden.

Eine Abfrage des aktuellen Zustandes ist mit der Methode ‚IsSizeValid' möglich. Außerdem wird beim Übergang einer gültigen auf eine ungültige Größe der Event ‚SizeValid' ausgelöst. Man muss beachten, dass sowohl das Auslösen des Events als auch die Aktualisierung des Zustandsflags erst beim jeweils nächsten Zeichenvorgang erfolgt.

Neues in Version 2.01.00

Änderungen gegenüber der Version 2.0

1. Aufgrund der großen Anzahl von Ergänzungen wurde die Lizenznummer auf 20100 erhöht. Für den Einsatz in einer Entwicklungsumgebung ist eine **neue Lizenzdatei** notwendig. Bestehende Applikationen laufen ohne Änderung mit der neuen Version von *ActiveDiagram*.
2. Die Anzahl der projektierbaren Kurven wurde auf **10** erhöht. Durch die Ergänzung einer Bit-Darstellung, die als Eigenschaft jeder beliebigen Achse zugewiesen werden kann, ist somit sichergestellt, dass nach wie vor mindestens 8 Achsen für Kurven zur Verfügung stehen.
3. Die Marker wurden um 5 neue Typen ergänzt. Eine neue Eigenschaft der Y-Achsen **‚YMarker-ShowDistance'** legt einen X-Abstand in Pixel fest. Wird dieser Abstand zwischen 2 Markern unterschritten, werden alle Marker dieser Kurve nicht gezeichnet. Diese Abstandskontrolle funktioniert nur beim Zeichnen aus dem Archiv heraus.
4. Bei den **verschieblichen Achsen** gab es in der Vergangenheit immer das Problem, dass die verschobenen Kurven-Bitmaps nicht pixelgenau zur X-Achse angeordnet waren. Der Grund liegt in der Rundung von Verschiebewerten, die als ‚double' vorliegen, auf eine Pixelverschiebung, die nur ganzzahlig sein kann. Besonders bei „laufenden" Datendiagrammen, bei denen eine große Anzahl von kleinen Schiebeschritten auftreten, machten sich diese **Pixelfehler** störend bemerkbar. Es wurde eine Lösung gefunden, die ausgehend von einem Verschieberwert in Pixeln, rückwirkend die Skalierung der Achsen so anpasst, dass keine Pixelfehler mehr auftreten.
5. Beim Einsatz von **Modulo-Achsen** konnte der Fall auftreten, dass die Strichlinie am Modulo-Wert in der Nähe des rechten Randes nicht gezeichnet wurde. Dieses Problem wurde behoben.
6. Unter Windows XP wurde bei einem weissen Hintergrund der Zeichenfläche das Gitter nicht gezeichnet. Dieser Fehler wurde beseitigt.
7. Beim Zeichnen aus dem Archiv heraus wurden bisher die Kurven in der Reihenfolge ihres Index gezeichnet. Die Reihenfolge wurde jetzt so abgeändert, dass die selektierte Kurve immer als letzte gezeichnet wird. Dadurch ist der komplette Kurvenzug dieser Kurve sichtbar, was sonst bei nahezu gleichen Kurvenzügen nicht gegeben war.
8. Um auch große Punktmengen (z.B. 10000 Datensätze) bei mehreren Kurven performant zeichnen zu können, wurden die Zeichenalgorithmen überarbeitet.
9. Wie zum Teil bekannt ist, gab es enorme Probleme bei der Performance des Zeichnens von Kurven, die einen geometrischen Stift (PS_GEOMETRIC) verwenden. Unter Windows XP kam es zum Hängen des gesamten Systems für die Dauer von mehreren Minuten. Microsoft bestätigte diesen Fehler als Systemeigenschaft und empfahl, auf die Verwendung von geometrischen Stiften zu verzichten. Da es aber zum jetzigen Stand der Kenntnisse keine andere Möglichkeit zum Zeichnen von dicken, gestrichelten Linien gibt, wurde dieser Linienstil beibehalten. Er sollte aber möglichst nur in Fällen verwendet werden, bei denen die Datenmengen beschränkt sind. Für Stifte der Breite 0 (alle Linienstile) und für durchgezogene Linien (PS_SOLID) beliebiger Breiten werden jetzt generell kosmetische Stifte (PS_CSOMETIC) eingesetzt, bei denen das Problem nicht auftritt.
10. Das Zeichnen von Kurven als eine einfache grafische Darstellung war der Ausgangspunkt für die Entwicklung der Vorgängerprodukte von *ActiveDiagram*. Jeder Kurvenzug wurde direkt auf dem Bildschirm und parallel in eine Speicher-Bitmap gezeichnet. Die Daten waren nicht mehr direkt verfügbar und bei einer WM_PAINT-Anforderung wurde die Speicher-Bitmap auf den Bildschirm kopiert. Erst später wurden das Zeichnen im Blockmodus und die Archivfunktionalität ergänzt. Leistungsfähige PC-Systeme unterstützten diese neuen Funktionen. Dieses Grundprinzip wurde bis jetzt beibehalten. Einige neue Funktionen allerdings lassen sich nur realisieren, wenn man konsequent Datenarchive benutzt und auf ein direktes Zeichnen verzichtet. Die Bitdarstellung beispielsweise ist nur mit einem Datenarchiv möglich. Auch das Ausblenden von Markern, wenn ihr minimaler Abstand unterhalb eines projektierten Wertes liegt, ist nur möglich, wenn auf ein direktes Zeichnen verzichtet wird. Für solche Funktionen wurde die Methode **‚SetYMode'** um den Parameter **ARCHIVE = 3** erweitert. In diesem Modus erfolgt bei der Datenübergabe an das *ActiveDiagram* lediglich ein Einschreiben der Daten in das Archiv, ohne direkt einen Zeichenvorgang auszuführen. Dieser kann im Anschluss über **‚DrawArchive'** erfolgen (niemals in den Events ‚BitmapCleared' oder ‚SetupFinished' aufrufen).

Verschieben / Dehnen / Stauchen von Kurven

Als neue Funktion wurde das Verschieben, Dehnen und Stauchen mit der Maus integriert. Wenn die Funktion aktiviert ist, kann man im Bereich der Y-Achse mit der Maus ein Umskalieren vornehmen.

Bitdarstellung

Jede Y-Achse kann als eine Bit-Achse projektiert werden. Diese Bit-Achsen können bis zu 32 Bitspuren darstellen, wobei als Datenquelle jeder Bitspur ein beliebiges Bit einer beliebigen Kurve zugewiesen werden kann. Besonders vorteilhaft ist es, dass auch die Bit-Achse selbst als Datenquelle projektiert werden kann. Das Zeichnen der Bitspuren kann nur aus dem Archiv heraus erfolgen.

Zeichnen von Kurven

Die Speicherung der Daten im Archiv wurde überarbeitet, so dass jetzt auch Kurven, die in einzelne Kurvenstücke untergliedert sind gespeichert und gezeichnet werden können. Optional können den einzelnen Kurvenpunkten zusätzlich Marker zugewiesen werden.

Neues in Version 2.01.01

Änderungen gegenüber der Version 2.01.00

1. Die Setup-Funktion von *ActiveDiagram* wurde bisher beim ersten Zeichen ausgeführt. In Anwendungen, bei denen eine Visualisierung des Controls nicht direkt erfolgt kann dieses Verhalten nachteilig sein. Deshalb wird jetzt das ‚Setup' im ‚Create' aufgerufen. Damit wird der Event ‚SetupFinished' auch dann ausgelöst, wenn das Control nicht gezeichnet wird.
2. Die Methode ‚SetMouseMode' lässt jetzt auch das Aktivieren der jeweils 2. Linien ohne Aktivierung der 1. Linien zu. Bisher waren als Parameter beispielsweise nur 0x04 und 0x0c zulässig. Jetzt ist auch 0x08 möglich.
3. Die Methode ‚**SetMouseMode'** sollte in neuen Applikationen **nicht mehr verwendet** werden. Die Eigenschaft ‚ZoomMode' und die speziellen Eigenschaften und Methoden für die Cursorlinien lassen eine differenzierteren Einsatz mit Funktionserweiterungen zu.

Trigger- und Begrenzungslinien

Im *ActiveDiagram* sind pro Kurve zwei waagerechte Begrenzungslinien zuschaltbar. Die Linien können vom Anwender in ihrer Y-Position parametiert werden. Weiterhin kann eine senkrechte Triggerlinie gesetzt werden. Als Kennzeichnung für die Triggerlinie dient ein kleines Dreieck am unteren Rand, das zusätzlich einen Buchstaben enthalten kann. Die Linien können vom Anwender in ihrer X-Position parametiert werden.

Messcursorlinien / Zoom-Funktion

Die bisherigen Messcursorlinien wurden um die Möglichkeit eines Parkbereiches erweitert, in dem ein Cursor abgelegt oder aus ihm entnommen werden kann. Um die Handhabung von Cursor und Zoom zu verbessern, wurde eine neue Eigenschaft *ZoomMode* implementiert.

Notizzettel

An jeder beliebigen Stelle im Zeichenbereich kann der Anwender mit der Maus Hilfspunkte setzen und beschriften. Damit können Punkte oder Positionen für eine spätere Auswertung markiert werden.Die Notizzettel werden über ein Objekt *INotice* realisiert. Die Objekte haben die Eigenschaften Nummer der Y-Achse, X- und Y-Position (physikal. Werte), Textstring, Farbe des Textes und des Hintergrundes.

Beschriftung der Kurven

Es wird eine Eigenschaft definiert, mit deren Hilfe die Lage und das Design der Beschriftung von Kurven modifiziert werden kann.

Neues in Version 2.02.00

Änderungen gegenüber der Version 2.01.01

1. In der Version 2.01.01 wurde die Setup-Funktion in das Create verlegt. Da es auch Anwendungen gibt, bei denen das Setup im ersten Zeichnen notwendig ist, wurde die Eigenschaft ‚SetupInCreate' eingeführt, über die man die Ausführung des Setup steuern kann.
2. Es wurden 5 zusätzliche Eigenschaften in die allgemeine Property-Page aufgenommen, was auch die einzige Möglichkeit ist, die Eigenschaft ‚SetupInCreate' auf TRUE zu setzen.
3. Signalnamen mit dem Stil = 1 (innerhalb der Zeichenfläche) werden auch bei Bitachsen dargestellt.
4. *ActiveDiagram* kann nicht mehr ‚windowless' aktiviert werden. In Visual Basic gab es bei einer fensterlosen Aktivierung Probleme bei der korrekten Berechnung der Zeichenkoordinaten.
5. In die TypeLib wurden eine Reihe von Konstanten aufgenommen, die zu einer Vereinfachung im Umgang mit den Funktionen führt.

Fragen und Antworten (FAQ)

Wie und wo sollte man Daten an ActiveDiagram übergeben?

Der beste Ort, die Daten in die Kurven einzuschreiben, ist ‚SetupFinished'. Wenn Sie den Kurven kein Archiv zugewiesen haben, müssen die Daten auch zusätzlich in ‚BitmapCleared' übergeben werden.

Ab der Version 2.0 wird beim ersten Setup zusätzlich zum Event ‚SetupFinished' auch der Event ‚BitmapCleared' generiert. Diese Veränderung ermöglicht es, bei Nichtverwendung eines Archivs die Daten nur in ‚BitmapCleared' übergeben zu müssen.

Wie kann man Kurven mit Lücken zeichnen?

Der geschlossene Kurvenzug ist der Normalfall in ActiveDiagram. Wenn Sie Kurven mit Lücken darstellen möchten, so müssen Sie auf die Archivfunktion verzichten und die Daten der Kurven in den Events ‚SetupFinished' und ‚BitmapCleared' direkt übergeben. Jedes einzelne Kurvenstück beginnen Sie dabei mit dem Aufruf ‚ResetY', der eine Kurve rücksetzt, das Archiv löscht, aber nicht die Hintergrund-Bitmap beeinflusst. Alternativ dazu können Sie auch jedem Kurvenstück eine eigene Y-Achse zuordnen, die alle gleich skaliert sein sollten und dann auch ein Archiv haben können. Allerdings sind Sie dabei auch maximal 8 Kurvenstücke beschränkt.

Ich habe Daten mit X-Werten von 0..150 direkt in das Archiv geschrieben und mit ‚DrawArchive' einen Zeichenvorgang ausgelöst. Die X-Achse ist auf 0..100 skaliert und verschieblich projektiert ist. Warum führt das Zeichnen nicht zu einer Verschiebung der Achse?

Ihre Vorgehensweise die Daten direkt in das Archiv zu schreiben, ist unüblich aber durchaus legitim. Allerdings führt ein Zeichnen aus dem Archiv, wie Sie richtig bemerkt haben, nicht zu einer Verschiebung der X-Skalierung. Das wäre auch nicht sinnvoll, da das Zeichnen aus dem Archiv u.a. auch für gezoomte Darstellungen und für das Scrollen genutzt wird. Würde das Zeichnen aus dem Archiv ein Verschieben der X-Skalierung bewirken, könnte man immer nur das Ende der Kurven sehen. Wenn Sie diese Art der Datenübergabe beibehalten möchten und trotzdem ein Verschieben erreichen wollen, so sollten Sie einfach den letzten X-Wert zusätzlich mit ‚SetX' übergeben.

Ich habe eine Impulsfolge gezeichnet und festgestellt, dass nach einer erfolgten Verschiebung und Neuskalierung der X-Achse manchmal Bitflanken um 1 Pixel verschoben gezeichnet werden. Was ist der Grund für diese Ungenauigkeit?

Die Skalierung von Gleitkommawerten auf die Pixellänge der Achse führt zu einer Rundung der Werte, wodurch Ungenauigkeiten von 1 Pixel auftreten. Wenn Sie beispielsweise eine Länge von 143 Pixel auf 10 Abstände aufteilen wollen, werden einige Abstände 14 und einige Abstände 15 Pixel groß sein. In der Hintergrund-Bitmap sind die Kurven bezogen auf eine aktuelle Skalierung „eingefroren". Ein Verschieben der Bitmap, verbunden mit einer Neuskalierung, führt zu einer neuen Zuordnung der Abstände, wodurch es zu dieser 1-Pixel-

Ungenauigkeit kommen kann. Wenn diese Ungenauigkeit störend ist, kann man nach einer erfolgten Verschiebung den Inhalt der Kurven aus dem Archiv heraus neu zeichnen.

```
double dMin = m_ctrlDiag.GetXMinimum();      // altes Minimum merken
m_ctrlDiag.SetX( value );                    // X-Wert setzen
if( dMin != m_ctrlDiag.GetXMinimum() )       // Ist das Minimum verändert?
        m_ctrlDiag.DrawArchive();            // ja, Neuzeichnen aus Archiv
```

Welche Aktionen führen zu einem Löschen der Bitmap, wodurch ein Neuzeichnen der Daten notwendig wird?

Alle Aktionen, die einen Einfluss auf den Inhalt der Hintergrund-Bitmap haben, löschen diese und aktivieren den Event ‚BitmapCleared', wodurch ein Neuzeichnen ausgelöst werden muss. Das sind: Setzen aller Minima und Maxima der Achsen, SetPaperColor, SetOffset... , SetXModuloEnable, SetXModuloValue, SetXLogEnable, SetYColor, SetYConnectType, ClearAll, DrawArchive, MoveControl, CopyToClipBoard mit Übergabe von Größen.

Wie kann ich die Version meines ActiveDiagram ermitteln?

Ein Doppelclick mit der linken Maustaste bei gedrückter Strg-Taste (Ctrl) auf das ActiveDiagram öffnet den Info-Dialog, der die Versionsnummer enthält.

Kann die neue Version von ActiveDiagram in älteren Applikationen genutzt werden?

Die Versionen von ActiveDiagram sind generell aufwärts kompatibel. Es kann somit problemlos eine neuere Version auf dem PC installiert werden, mit der auch ältere Programme arbeiten können. Wenn Sie allerdings die neueste Version in Ihrer Entwicklungsumgebung nutzen wollen, benötigen Sie ein Update und eine neue Lizenzdatei.

Kann ich auf einfache Art und Weise Kurven ein- und ausblenden, ohne die Daten neu einschreiben zu müssen?

Wenn Sie der Kurve ein Archiv zugewiesen haben, können Sie durch mit ‚SetYArchiveEnable' das Archiv aktivieren bzw. deaktivieren, wobei der Inhalt des Archivs nicht verändert wird. Ein nachfolgender Aufruf von ‚DrawArchive' führt dann den Zeichenvorgang aus, so dass je nach Zustand des Flags die Kurve dargestellt wird oder nicht.

ActiveDiagram kann durch das Aufrufen der Methode ‚GetYOffsetLeft' den notwendigen Zwischenraum für die Y-Achse berechnen. Warum wird dann nicht auch gleich der linke Offset gesetzt?

Das automatische Setzen des linken Offset ist nicht in jeder Applikation gewünscht. Wenn Sie beispielsweise mehrere Diagramme übereinander haben, die sich auf eine gemeinsame X-Skalierung beziehen, wird ein für alle Diagramme ein gleicher linker und rechter Offset gefordert. Diese Forderung könnte man bei einer automatischen Skalierung nicht erfüllen.

Trotz einer hohen Auflösung bei der Übergabe an GetPicture erscheint die ausgedruckte Grafik relativ grob. Warum ist das so?

Bei der Ausgabe einer Grafik in einen Gerätekontext kann man sich verschiedene Philosophien vorstellen. Wir haben bei der Ausgabe über die Methode *GetPicture* definiert, dass der Anwender die Größe der Grafik in 0,01 mm-Einheiten (HIMETRIC) festlegt. Die Ausgabe in den übergebenen Gerätekontext erfolgt dann so, dass bei einem Drucken in der vorgegebenen Breite und Höhe die Fonts exakt in der richtigen Größe gedruckt werden. Diese Anforderung hat zur Folge, dass intern die Umrechungsfaktoren von logischen Koordinaten in Gerätekoordinaten entsprechend angepasst werden müssen. Dadurch entsteht eine Pixelauflösung der Grafik die niedriger ist, als es die übergebenen HIMETRIC-Werte versprechen.

Sie können eine höhere Auflösung erreichen, indem Sie die Breite und Höhe bei *GetPicture* größer wählen als es für die Ausgabe in *PlayMetaFile* erforderlich ist. Sinnvollerweise sollte dabei auch die Fontgröße entsprechend angepasst werden.

Ich habe die Eigenschaft ‚SetupInCreate' im Programm gesetzt, aber trotzdem wird das Setup beim ersten Zeichnen ausgeführt. Warum?

Die Eigenschaft ‚SetupInCreate' muss beim Create des *ActiveDiagram* gesetzt sein. Aus diesem Grund ist es nicht möglich, die Eigenschaft aus dem Programm heraus zu setzen, da entweder die Instanz noch nicht angelegt ist oder das Create bereits abgearbeitet ist. Setzen Sie die Eigenschaft in der Property-Page.

Das Setzen von Grenzen scheint im Event ‚BitmapCleared' nicht zu funktionieren. Warum?

Der Event ‚BitmapCleared' wird immer dann aufgerufen, wenn durch Änderungen in der Parametrierung ein Neuzeichnen der Kurven erforderlich wird. Das Setzen der Grenzen ist eine solche Änderung und würde damit zum rekursiven Aufruf des Zeichnens führen. Das wird im ActiveDiagram verhindert, indem zuerst die Parameter neu berechnet, dann die Zeichenfläche gelöscht, der Event bearbeitet und erst danach die Merkzelle für eine Umparametrierung gelöscht wird. Dadurch wird zwar die Rekursion vermieden, aber zugleich auch eine Umparametrierung wirkungslos. Führen Sie Umparametrierungen nie in den Events aus.

Welche Funktionen laufen beim ‚ClearAll' ab?

Es werden zuerst die Kurvenspeicher aller projektierten Achsen zurückgesetzt. Danach erfolgt das Löschen der Speicherbitmap und der Anwender wird über den Event ‚BitmapCleared' informiert. Durch ein ‚Invalidate' wird eine Zeichenanforderung abgesetzt.

Das ‚ClearAll' ist nur wirksam, wenn das Setup des *ActiveDiagrams* bereits erfolgt ist. Bei den Defaulteinstellungen ist ‚ClearAll' beispielsweise in einem ‚OnInitDialog' unwirksam.

Service

Bei Problemen oder Fragen können sich lizenzierte Kunden unter Angabe der Lizenznummer jederzeit an folgende Adresse wenden:

Ingenieurbüro Halbritter
Am Langen Hardt 18
91180 Heideck

Tel: 09177 4988342
Fax: 09177 4988344
E-Mail: mailto:service@i-halbritter.de
Internet: http://www.i-halbritter.de/

Anlage 1: Funktionen für Zoom und Leselinien

Nicht für neue Anwendungen

SetMouseMode

Event ZoomRect ²

Mode	Bitzuordnung	Funktion	Parameter 1	Parameter 2	Parameter 3	Parameter 4
0	00 00 00	Keine Funktion aktiv	-	-	-	-
1	00 00 01	Markieren eines Rechteckbereiches in X- und Y-Richtung mit der linken Maustaste (Festlegen eines Zoombereiches). Event wird beim Loslassen generiert und Deaktivierung der Funktion (Mode = 0).	Startpunkt X	Startpunkt Y	Endpunkt X	Endpunkt Y
2	00 00 10	Markieren eines Rechteckbereiches in X-Richtung mit der linken Maustaste (Festlegen eines Zoombereiches). Die Y-Koordinaten liegen auf dem Rand der Zeichenfläche. Event wird beim Loslassen generiert und Deaktivierung der Funktion (Mode = 0).	Startpunkt X	Startpunkt Y	Endpunkt X	Endpunkt Y
3	00 00 11	Aktivieren einer Leselinie (X-Richtung). Die Leselinie wird in der Mitte des Bereiches initialisiert. Jede Verschiebung der Leselinie generiert einen Event. Abschalten der Leselinie durch Mode = 0.	Linie X	-	-	-
4[1]	xx x1 xx	X Anzeigen der 1. Leselinie in X-Richtung	1. Linie X	0 (Linie.)	Delta	-
12[1]	xx 1x xx	X Anzeigen der 2. Leselinien in X-Richtung	2. Linie X	1 (Linie.)	Delta	-
16[1]	x1 xx xx	Y Anzeigen der 1. Leselinie in Y-Richtung	1. Linie Y	2 (Linie.)	Delta	-
48[1]	1x xx xx	Y Anzeigen der 2. Leselinien in Y-Richtung	2. Linie Y	3 (Linie.)	Delta	-

x Bitbelegung kann beliebig sein
\- Kennzeichnung von undefinierten Werten
[1] Nummer für den Mode kann mit Bits der jeweils anderen Achse und mit den Bits für die Zoomfunktion verknüpft werden.
² Neudefinierte Events zusätzlich zu ZoomRect ab der Version 2.0 (Siehe Seite 30)

- Es können bis zu jeweils 2 Leselinien in X- und Y-Richtung dargestellt werden (z.B. alle 4 Linien mit Mode = 60).
- Jede dargestellte Linie kann durch Clicken mit der Maus aktiviert und verschoben werden. Die Liniennummer der gerade aktiven Linie wird im Event (Parameter 2) übergeben und kann die Werte von 0...3 annehmen. Der dazugehörige Pixelwert ist im Parameter 1 enthalten.
- Das 'Delta' (Parameter 3) gibt die Differenz in Pixels zwischen der gezeichneten Lage der Leselinie und der gewünschten Lage durch die Mausbewegung an. 'Delta' ist nur dann $\neq 0$, wenn die Leselinie über den Rand hinausgezogen werden soll.
- Der Initialisierungswert für die Leselinien ist die Mitte der Zeichenfläche. Durch die Methoden *SetCursorXY* / *GetCursorXY* und *GetCursorVal* / *SetCursorVal* kann jede Linie auf einen gewünschten Wert gesetzt bzw. der aktuelle Wert gelesen werden.