# X1 Beispiele

# X1 Beispiele

## Beschreibung der Beispieldokumente

---

*Das Handbuch bietet einen Überblick über Aufbau und Anwendung der Beispieldokumente im Lieferumfang*

# Inhaltsverzeichnis

# 1 Beispiele

Dieses Handbuch beschreibt die mit X1 gelieferten Beispiele. Die Beispiele gliedern sich nach

- Grafikobjekte
- Kurvenobjekte
- CX1
- Datenbank
- ActiveX

## 1.1 Grafikobjekte

**Verzeichnis**
<X1>/samples/elemente

**Beschreibung**
Graphikelemente sind graphische Objekte, die in das Layout eines X1-Dokumentes aufgenommen werden können. Im obigen Verzeichnis finden sie eine Reihe von x1g- Dateien mit unterschiedlichen Grafikobjekten.

## 1.2 Kurvenobjekte

**Verzeichnis**
<X1>/samples/kurven

**Beschreibung**
Als *Kurvenobjekte* werden alle Objekte bezeichnet, die in Diagramme eingezeichnet werden können. Grundsätzlich wird unterschieden in zweidimensionale und dreidimensionale *Kurvenobjekte*. Im obigen Verzeichnis finden sie eine Reihe von x1g- Dateien mit unterschiedlichen Kurvenobjekten.

## 1.3 CX1

**Verzeichnis**
<X1>/samples/cx1

**Beschreibung**
Im obigen Verzeichnis finden sie eine Reihe von x1g- Dateien mit einfachen Programmierbeispielen.
Sie starten die Beispiele (d.h. sie führen das CX1- Skript des jeweiligen Dokumentes aus) mit der Run- Taste.

Objekte, die ein Skript besitzen, sind im Projektfenster mit dem Symbol  markiert. Um das Skript zu bearbeiten schalten sie X1 in den Bearbeitungsmodus und öffnen über das Kontextmenü des Objektes dessen Skript. Fehlerhafte Skripte werden nach dem Compilieren mit dem Symbol  markiert.

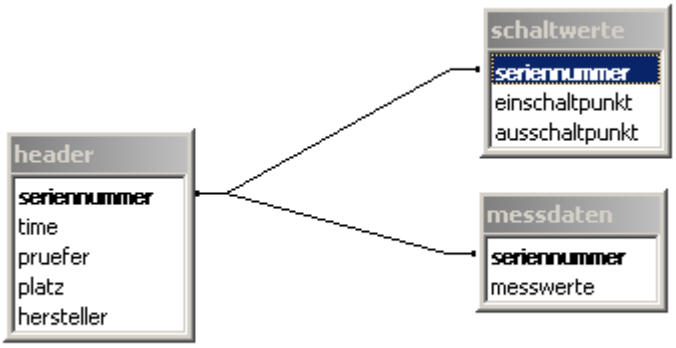
## 1.4 Datenbank

**Verzeichnis**
<X1>/samples/datenbank
**Beschreibung**
Die hier beschriebenen Beispiele sollen die Anbindung von X1 an eine ODBC Datenquelle verdeutlichen. Alle Beispiele greifen auf eine Datenbank zu und werten diese auf verschiedene Art und weise aus. Hierbei wurde besonders viel Wert auf das Bearbeiten von BLOB's (**B**inary **L**ogic **Ob**jekt) gelegt.
Viele der Beispiele würden sich bezüglich der Laufzeit noch verbessern lassen, hierauf wurde aber verzichtet um die Lesbarkeit der Beispiele zu erhöhen.

## 1.4.1 Datenbank Aufbau

### 1.4.1.1 Datenbank Aufbau

Bei der Datenbank handelt es sich um eine Access Datenbank mit folgenden Aufbau:



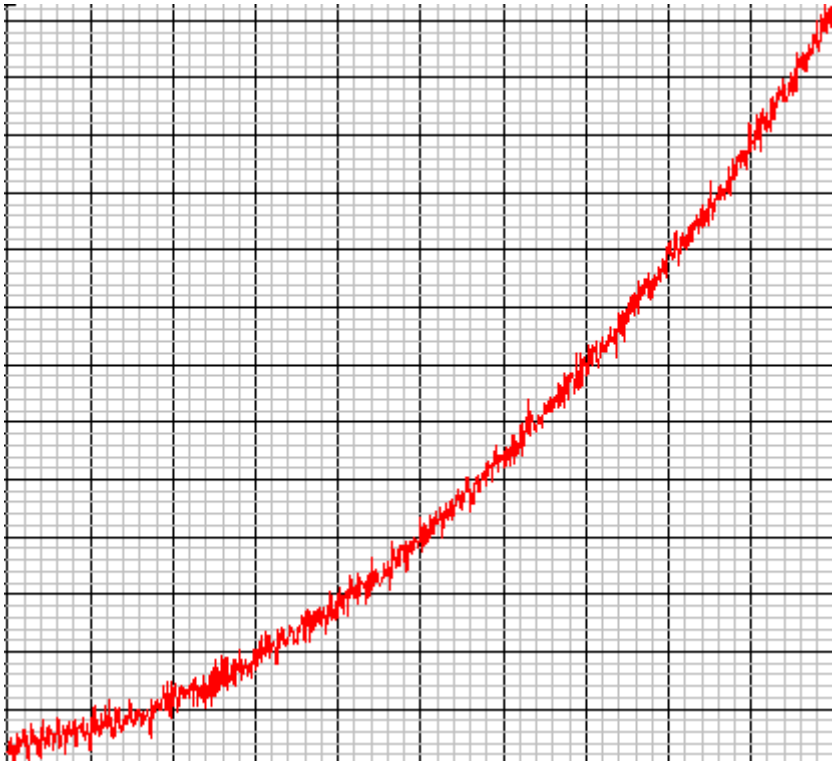

| | |
|---|---|
| seriennummer | Text |
| time | Datum/Uhrzeit |
| pruefer | Text |
| platz | Text |
| hersteller | Text |
| einschaltpunkt | Zahl |
| ausschaltpunkt | Zahl |
| messwerte | OLE-Objekt |

Eine Messung besteht immer aus einem Eintrag in der Tabelle header, schaltwerte und messdaten.

### 1.4.1.2 Aufbau eines OLE Objekts

**Aufbau des OLE-Objekts:**

Bei den *OLE-Objekten* (Feld `messwerte` in der Tabelle `messdaten`) handelt es sich um ein so genanntes BLOB (**B**inary **L**ogic **Ob**jekt). Hinter diesem Objekt verbirgt sich eine Messkurve bei der alle Messwerte hintereinander abgespeichert wurden.
Grafische Auswertung eines solchen Blobs:



**Aufbau einer Stützstelle(Messwert):**

Es handelt sich um 8 Byte große Zahlen mit folgenden Aufbau:

| LSB | | | | | | | MSB |
|---|---|---|---|---|---|---|---|
| Byte 7 | Byte 6 | Byte 5 | Byte 4 | Byte 3 | Byte 2 | Byte 1 | Byte 0 |

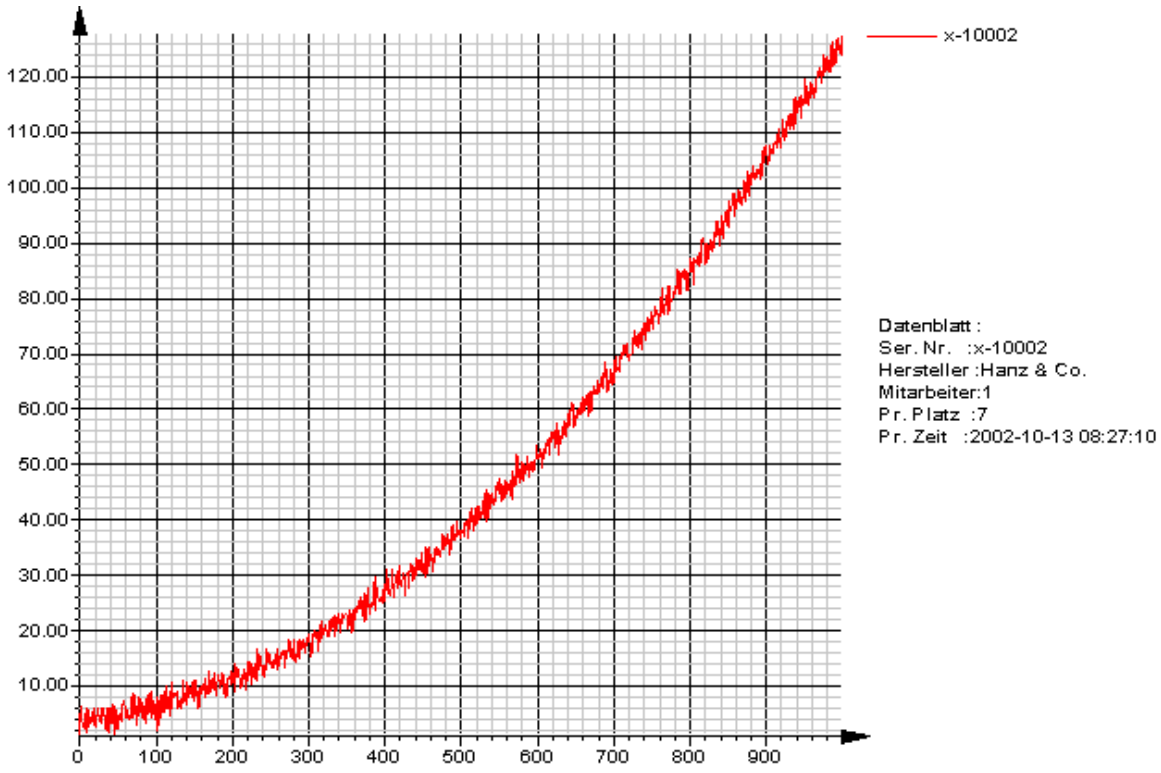
Leider benötigt X1 zur Interpretation als Doublwerte diese in umgekehrter Reihenfolge:

| Byte 0 | Byte 1 | Byte 2 | Byte 3 | Byte 4 | Byte 5 | Byte 6 | Byte 7 |
|---|---|---|---|---|---|---|---|

Um diesem Problem zu begegnen gibt es die Funktion LVDouble. Diese Funktion wandelt eine UNIX-Gleitkommazahl in das unter Windows gebräuchliche format um. Sie ändert die Byte-Reinfolge des *doubles*. Dies ist notwendig da die Bytes im Blob in umgekehrter Reihnfolge vorliegen (UNIX- Format).

```
double LVDouble(char* pC)
{
        char buf[8];
        buf[0]=pC[7];
        buf[1]=pC[6];
        buf[2]=pC[5];
        buf[3]=pC[4];
        buf[4]=pC[3];
        buf[5]=pC[2];
        buf[6]=pC[1];
        buf[7]=pC[0];
        void *pV=buf;

        return *(double*)pV;
}
```

## 1.4.2 oneLine




### 1.4.2.1 Kurzbeschreibung

Dieses Beispiel zeichnet eine einzelne Kurve die aus einem BLOB (**B**inary **L**arge **Ob**ject) gelesen wird. Der *BLOB* wird aus einer ODBC- Datenquelle eingelesen. Zusätzlich werden aus dieser ODBC Datenquelle die zu der Kurve gehörenden Produktionsdaten entnommen und angezeigt.

%.2E

### 1.4.2.2 Bedienungsanleitung

Sie starten dieses Beispiel mit der Run- Taste.

Nach dem Start des Beispiels werden Sie aufgefordert die Seriennummer der zu zeichnenen Kurve einzugeben.
Anschließend wird diese Kurve Gezeichnet.

Mögliche Seriennummern:

- x-10000 bis x-10099
- y-10000 bis y-10099

### 1.4.2.3 Skriptbeschreibung

Um das Skript zu bearbeiten müßen Sie sich in der Betriebsart Bearbeiten befinden. Drücken Sie nun mit der Rechten Maustaste auf **ROOTO** und wählen den Menue Punkt Skript an. Der Einsprungpunkt für das X1 ist die Methode OnRun.

1.4.2.3.1 paintL

Diese Funktion zeichnet einen Graph.

Parameter:

| | |
|---|---|
| `double* x` | *double* Array mit Messdaten |
| `int len` | Länge des Array x |
| `int r` | Farbe des Graphen Rot Anteil |
| `int g` | Farbe des Graphen Grün Anteil |
| `int b` | Farbe des Graphen Blau Anteil |
| `char* name` | Name des Graphen, der in der Legende erscheinen soll |

```
void paintL(double* x, int len, int r, int g, int b, char* name)
{       // diese Funktion zeichnet einen Graph
        // x Messwerte
        // r,g,b Farbe des Graphen
        // len Anzahl der Stützstellen
        // name Name des Graphen in der Legende

        CKurve2DTrace *mK=new CKurve2DTrace();
        mK->SetTraceAtt(1150,3);                    // Dieses Attribut gibt an, mit
welcher Funktion die Stützpunkte einer Kurve interpoliert werden
        mK->SetTraceAtt(1152,300);                  // Größe des Symbols zur
Punktmarkierung in [1/100 mm]
        mK->SetTraceAtt(4150,"AX");                 // legt die X Achse fest
        mK->SetTraceAtt(4151,"AY");                 // legt die Y Achse fest
        mK->SetTraceAtt(4154,name);                 // Name des Graphen in der Legende
        mK->SetTraceAtt(10021,x,len);         //Y-Werte explizit
        mK->SetTraceAtt(10020,0.0,1.0,len);   //X-Werte implizit
        mK->SetTraceAtt(9220,"X1-Sym",0);     // Font und Symbol für die Markierung
        mK->SetTraceAtt(7220,0,0,r,g,b);      // Füllmuster für die Markierungssymbole
        mK->SetTraceAtt(6220,1,0,0,0,255);    // Linienart für die Umrandung der
Markierungssymbole
        mK->SetTraceAtt(6221,1,0,r,g,b);      // Linienart und Farbe für die Kurve
        PUB.AddTrace(mK);                                  // fügt den Graphen in die
Legende ein
        return;
}

void paintL(double* x, int len, int r, int g, int b){
        paintL(x,len,r,g,b,"XXX");
        return;
}
```

1.4.2.3.2 LVDouble

Diese Funktion wandelt eine UNIX-Gleitkommazahl in das unter Windows gebräuchliche format um. Sie ändert die Byte- Reinfolge des *doubles*. Dies ist notwendig da die Bytes im Blob in umgekehrter Reihnfolge vorliegen (UNIX- Format).

**Parameter:**

`char* pC` double Wert im Unix Format

**Rückgabewert:**

`double` double Wert im Windows Format

```
double LVDouble(char* pC)
{
        char buf[8];
        buf[0]=pC[7];
        buf[1]=pC[6];
        buf[2]=pC[5];
        buf[3]=pC[4];
        buf[4]=pC[3];
        buf[5]=pC[2];
        buf[6]=pC[1];
        buf[7]=pC[0];
        void *pV=buf;

        return *(double*)pV;
}
```

1.4.2.3.3 oneKurven

Läd einen BLOB aus der ODBC Datenquelle. Ermittelt aus dem BLOB eine Kurve und zeichnet diese.

**Parameter**

| | |
|---|---|
| `CText *T` | Text Abfrage welche die Seriennummer der zu zeichnen Kurve enthält |
| `CDatabase *pDB` | Datenquelle |

Rückgabewert

| | |
|---|---|
| int | Anzahl der Messungen im BLOB |

```
int oneKurven(CText *T,CDatabase *pDB)
{       // Diese Methode zeichnet Das Band der Messungen und den Mittelwert
        // Rückgabewert Anzahl der Stützstellen
        CDBVariant DV;                  // Virtueller Datentyp für MFC ODBC Klasse
        CRecordset RS(pDB);             // Ergebnistyp  der ODBC Klasse (Abfrage Ergebnis)

        char qry[256];                          // SQL Abfrage String
        sprintf(qry,"SELECT * FROM messdaten where seriennummer='%s'",T->GetData());
        int len = 0; // Anzahl der Stützstellen
        int sz =0;   // Länge des Blobs

        if (RS.Open(2,qry,0) &&
           //mit pRS->Open wird die SQL anfrage an pRS und damit an die DB übergeben pRS
wird geöffnet
           !RS.IsBOF() &&
           //pRS->IsBOF() damit wird geprüft ob die Aktuelle Position des Ergebniszeigers
vor dem                                         //ersten Ergebnis steht (Beginn of File)
           !RS.IsEOF())
           //pRS.IsBOF() damit wird geprüft ob die Aktuelle Position des Ergebniszeigers
           //hinter dem ersten Ergebnis steht (End of File)
           //!pRS.IsBOF() && !pRS.IsEOF() hiermit wird überprüft ob der
           //Ergebniszeiger nicht vor und nach den Ergebnissen gleichzeitig steht,
           //dieser Fall das ist nur möglich wenn ein Ergebnis der Länge 0 zurückkam
        {

                char   *pV=0;           // Aktueller Blob
                double *pD=0;           // Aktuelle Stützstellen aus Blob

                printf("Tabelle 'messdaten' geöffnet\n");
                // kopiert das Feld mit dem Index 1 aus dem aktuellen Abfrageergebnis in
die Variable pDV
                RS.GetFieldValue(1,&DV);
                // erechnet beim ersten Durchlauf die Länge des Blobs
                sz=DV.GetBinarySize();
                // Anzahl der Messungen die sich im Blob befinden
                len=(sz-4)/8;
                pV=(char*)malloc(sz);
                pD=(double*)malloc(len*8);
                //Hole Kurvendaten von DB in ein Array
                // das Ergebnis liegt anschließend in pV
                DV.GetBinaryData(pV,sz);
                // hier werden alle Kurvendaten (Messdaten aus dem BLOB) durchgelaufen
                for (int i=0;i<len;i++)
                {
                        // aktueller Datenwert Niederwertigstes Byte und Höherwertiges Byte
werden vertauscht,
                        // da diese durch Labview in "falscher" Reinfolge in der DB
abgespeichert werden

                        pD[i]=LVDouble(pV+8*i+4);
                }

                paintL(pD,len,255,0,0,T->GetData()); // Zeichne die Kurve

                if (pD) free(pD);
                if (pV) free(pV);
                RS.Close();
```

```
        }
        else
                printf("Tabelle 'messdaten' nicht geöffnet\n");

        return len;
}
```

1.4.2.3.4 infoM

Diese Funktion gibt die Produktionsdaten zu der gezeichneten Kurve aus.

**Parameter**

| | |
|---|---|
| `CText *T` | Text Abfrage welche die Seriennummer der zu zeichnen Kurve enthält |
| `CDatabase *pDB` | Datenquelle |

```
void infoM(CText *T,CDatabase *pDB){
	// diese Funktion zählt die Anzahl der Messungen

	// Virtueller Datentyp für MFC ODBC Klasse
	CDBVariant DV;

	// Ergebnistyp  der ODBC Klasse (Abfrage Ergebnis)
	CRecordset RS(pDB);                          // Ergebnistyp  der ODBC Klasse
(Abfrage Ergebnis)

	// Rückgabewert der Methode
	int ret = 0;
	// Ausgabe Text
	char text[256];
	//Ergebnisse aus DB
	char hersteller[256],pruefer[255],mtime[255],platz[255]; //ergebnisse aus DB
	sprintf(hersteller,"keine Angabe");
	sprintf(pruefer,"keine Angabe");
	sprintf(mtime,"keine Angabe");
	sprintf(platz,"keine Angabe");


	// SQL String der Abfrage
	char qry[256];
	sprintf(qry,"SELECT hersteller,pruefer,time,platz "
			"FROM header "
			"WHERE seriennummer ='%s'",
			T->GetData());

	//mit pRS->Open wird die SQL anfrage an pRS und damit an die DB übergeben
	if(RS.Open(2,qry,0))
	{
		// Hole das Feld mit index 0 aus dem Abfrageergebnis heraus
		RS.GetFieldValue(0,hersteller,sizeof(hersteller));
		// Hole das Feld mit index 1 aus dem Abfrageergebnis heraus
		RS.GetFieldValue(1,pruefer,sizeof(pruefer));
		// Hole das Feld mit index 2 aus dem Abfrageergebnis heraus
		RS.GetFieldValue(2,mtime,sizeof(mtime));
		// Hole das Feld mit index 3 aus dem Abfrageergebnis heraus
		RS.GetFieldValue(3,platz,sizeof(platz));

		// Ausgabe des Ergebnisses
		char out[1000];
		sprintf(out,"Datenblatt :\n"
					"Ser. Nr.   :%s\n"
					"Hersteller :%s\n"
					"Mitarbeiter:%s\n"
					"Pr. Platz  :%s\n"
					"Pr. Zeit   :%s\n",T-
>GetData(),hersteller,pruefer,platz,mtime);
		info.SetText(out);


		RS.Close();
	}
	else printf("Fehler konnte DB in Funktion countM() nicht öffnen");
```

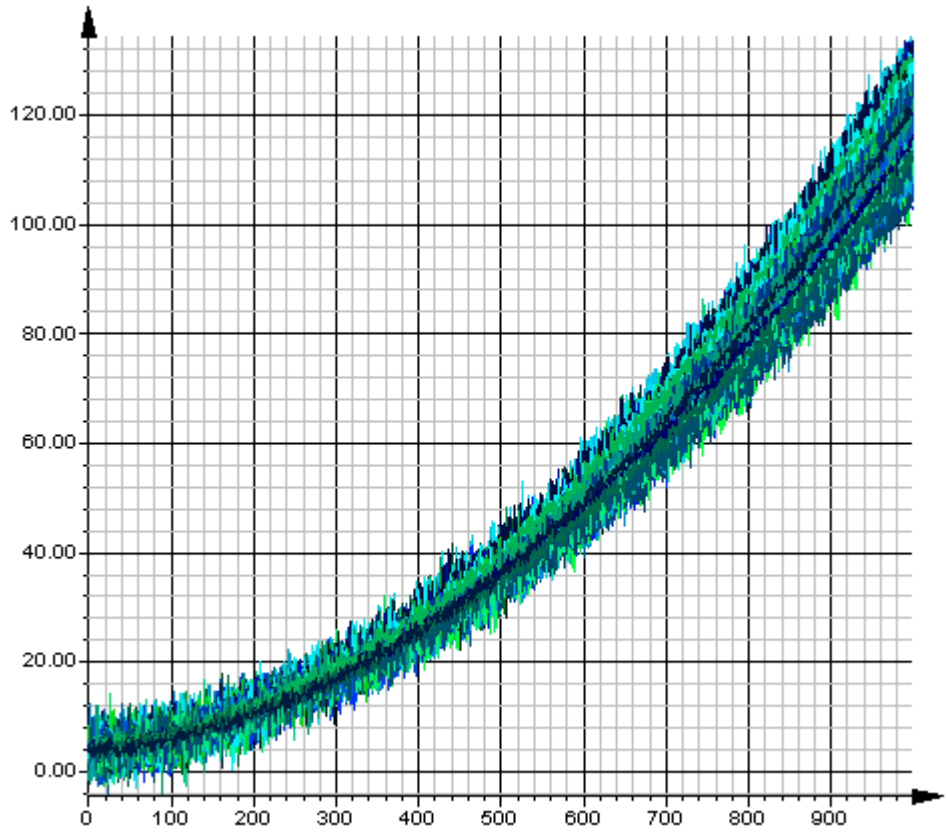
```
        return;
        }
```

1.4.2.3.5 OnRun

Hierbei handelt es sich um den Einsprungpunkt für das X1.

```
int method::OnRun()
{
        printf("START\n");

        PUB.Clear();          // Diagramm leeren

        CDatabase DB();               // Datenbank Objekt

        if (DB.Open("BLOB2",0,1,"ODBC;",1))   // Datenbank öffnen
                {
                printf("Datenbank 'BLOB2' geöffnet\n");

                CText T;
                // Textdialog
                T.SetData("x-10000");
                T.Edit("Bitte geben Sie das gesuchte Bauteil ein");
                // Aswertung der einzelnen Seriennummer
                oneKurven(&T,&DB); // Zeichnen der Kurve
                infoM(&T,&DB);     // Ausgabe der Produktionsdaten
                DB.Close();// Datenbank schließen

                }


        GetDocument()->Invalidate();
        return 1;
}
```

## 1.4.3 moreLine




### 1.4.3.1 Kurzbeschreibung

Dieses Beispiel zeichnet mehrere Messkurven die jeweils aus einem BLOB (**B**inary **L**arge **Ob**ject) gelesen wierden. Die *BLOB*s werden aus einer ODBC- Datenquelle eingelesen.

### 1.4.3.2 Bedienungsanleitung

Sie starten dieses Beispiel mit der Run- Taste.

Nach dem Start des Beispiels werden Sie aufgefordert die Seriennummer der zu zeichnenden Kurven einzugeben.
Anschließend werden diese Kurve gezeichnet.

Hierbei wird ein so genanntes *wildcard*- Zeichen verwendet. Das funktioniert wirkt ähnlich wie das "*" in Dos oder Unix. Das *wildcard*- Zeichen ist das % Zeichen. Das % steht für eine beliebige Anzahl von Zeichen.

Mögliche Seriennummern:
- x-10000 bis x-10099
- y-10000 bis y-10099

Beispiele für die Eingabe:

- Alle Seriennummern die mit x beginnen: x%
- Alle Seriennummern die mit 9 aufhören: %9
- Alle Kurven: %

Anschließend werden alle ausgewählten Kurven nacheinander in das Diagramm gezeichnet.

### 1.4.3.3 Skriptbeschreibung

Um das Skript zu bearbeiten müßen Sie sich in der Betriebsart Bearbeiten befinden. Drücken Sie nun mit der Rechten Maustaste auf **ROOTO** und wählen den Menue Punkt Skript an. Der Einsprungpunkt für das X1 ist die Methode OnRun.

1.4.3.3.1 paintL

Diese Funktion zeichnet einen Graph.

**Parameter:**

| | |
|---|---|
| `double* x` | *double* Array mit Messdaten |
| `int len` | Länge des Array x |
| `int r` | Farbe des Graphen Rot Anteil |
| `int g` | Farbe des Graphen Grün Anteil |
| `int b` | Farbe des Graphen Blau Anteil |
| `char* name` | Name des Graphen, der in der Legende erscheinen soll |

```
void paintL(double* x, int len, int r, int g, int b, char* name)
{       // diese Funktion zeichnet einen Graph
        // x Messwerte
        // r,g,b Farbe des Graphen
        // len Anzahl der Stützstellen
        // name Name des Graphen in der Legende

        CKurve2DTrace *mK=new CKurve2DTrace();
        mK->SetTraceAtt(1150,3);                    // Dieses Attribut gibt an, mit
welcher Funktion die Stützpunkte einer Kurve interpoliert werden
        mK->SetTraceAtt(1152,300);                  // Größe des Symbols zur
Punktmarkierung in [1/100 mm]
        mK->SetTraceAtt(4150,"AX");                 // legt die X Achse fest
        mK->SetTraceAtt(4151,"AY");                 // legt die Y Achse fest
        mK->SetTraceAtt(4154,name);                 // Name des Graphen in der Legende
        mK->SetTraceAtt(10021,x,len);         //Y-Werte explizit
        mK->SetTraceAtt(10020,0.0,1.0,len);   //X-Werte implizit
        mK->SetTraceAtt(9220,"X1-Sym",0);     // Font und Symbol für die Markierung
        mK->SetTraceAtt(7220,0,0,r,g,b);      // Füllmuster für die Markierungssymbole
        mK->SetTraceAtt(6220,1,0,0,0,255);    // Linienart für die Umrandung der
Markierungssymbole
        mK->SetTraceAtt(6221,1,0,r,g,b);      // Linienart und Farbe für die Kurve
        PUB.AddTrace(mK);                                  // fügt den Graphen in die
Legende ein
        return;
}

void paintL(double* x, int len, int r, int g, int b){
        paintL(x,len,r,g,b,"XXX");
        return;
}
```

1.4.3.3.2 LVDouble

Diese Funktion wandelt eine UNIX-Gleitkommazahl in das unter Windows gebräuchliche format um. Sie ändert die Byte- Reinfolge des *doubles*. Dies ist notwendig da die Bytes im Blob in umgekehrter Reihnfolge vorliegen (UNIX- Format).

**Parameter:**

`char* pC` double Wert im Unix Format

**Rückgabewert:**

`double` double Wert im Windows Format

```
double LVDouble(char* pC)
{
        char buf[8];
        buf[0]=pC[7];
        buf[1]=pC[6];
        buf[2]=pC[5];
        buf[3]=pC[4];
        buf[4]=pC[3];
        buf[5]=pC[2];
        buf[6]=pC[1];
        buf[7]=pC[0];
        void *pV=buf;

        return *(double*)pV;
}
```

1.4.3.3.3 Kurven

Diese Methode zeichnet alle Kurven die im SQL String (der sich im CText Objekt befindet) spezifiziert sind.

**Parameter**

| | |
|---|---|
| `CText *T` | Text Abfrage welche die Seriennummer der zu zeichnen Kurven enthält |
| `CDatabase *pDB` | Datenquelle |

Rückgabewert

| | |
|---|---|
| `int` | Anzahl der Messungen in den BLOB's (Stützstellen) |

```
int Kurven(CText *T,CDatabase *pDB)
{       // Diese Methode zeichnet alle Messungen
        // Rückgabewert Anzahl der Stützstellen
        CDBVariant DV();        // Virtueller Datentyp für MFC ODBC Klasse
        CRecordset RS(pDB);     // Ergebnistyp  der ODBC Klasse (Abfrage Ergebnis)
        char qry[256];          // SQL Abfrage String
        sprintf(qry,"SELECT * FROM messdaten where seriennummer like '%s'",T->GetData());
        int len = 0;            // Anzahl der Stützstellen

        if (RS.Open(2,qry,0) && !RS.IsBOF() && !RS.IsEOF())
        //mit pRS.Open wird die SQL anfrage an pRS und damit an die DB übergeben
        //pRS wird geöffnet
        //pRS.IsBOF() damit wird geprüft ob die Aktuelle Position des Ergebniszeigers vor
dem ersten
        //Ergebnis steht (Beginn of File)
        //pRS.IsBOF() damit wird geprüft ob die Aktuelle Position des Ergebniszeigers
hinter dem ersten
        //Ergebnis steht (End of File)
        //!pRS->IsBOF() && !pRS->IsEOF() hiermit wird überprüft ob der Ergebniszeiger nicht
        //vor und nach den Ergebnissen gleichzeitig steht,
        //dieser Fall das ist nur möglich wenn ein Ergebnis der Länge 0 zurückkam
        {
                int sz=0,num = -1;      // Laufvariable für Datensätze

                char   *pV=0;           // Aktueller Blob
                double* pD=0;           // Aktuelle Stützstellen aus Blob

                while (!RS.IsEOF())     // Wiederhole solange bis letzter Datensatz ereicht
                {
                        num++;
                        printf("Tabelle 'messdaten' geöffnet\n");
                        // kopiert das Feld mit dem Index 1 aus dem aktuellen
Abfrageergebnis in die Variable pDV
                        RS.GetFieldValue(1,&DV);
                        if (num==0)
                                {
                                // erechnet beim ersten Durchlauf die Länge des Blobs
                                sz=DV.GetBinarySize();
                                }
                        //wenn ein Datensatz mit falscher Länge in der DB sich befindet wird
dieser ausgelassen
                        if (sz=DV.GetBinarySize())
                        {
                                if (num==0)
                                {// hier werden die Arrays beim ersten Durchlauf reserviert

                                        // Anzahl der Messungen die sich im Blob befinden
                                        len=(sz-4)/8;
                                        pV=(char*)malloc(sz);
                                        pD=(double*)malloc(len*8);
                                }
                                //Hole Kurvendaten von DB in ein Array

                                // das ergebnis liegt anschließend in pV
                                DV.GetBinaryData(pV,sz);
```

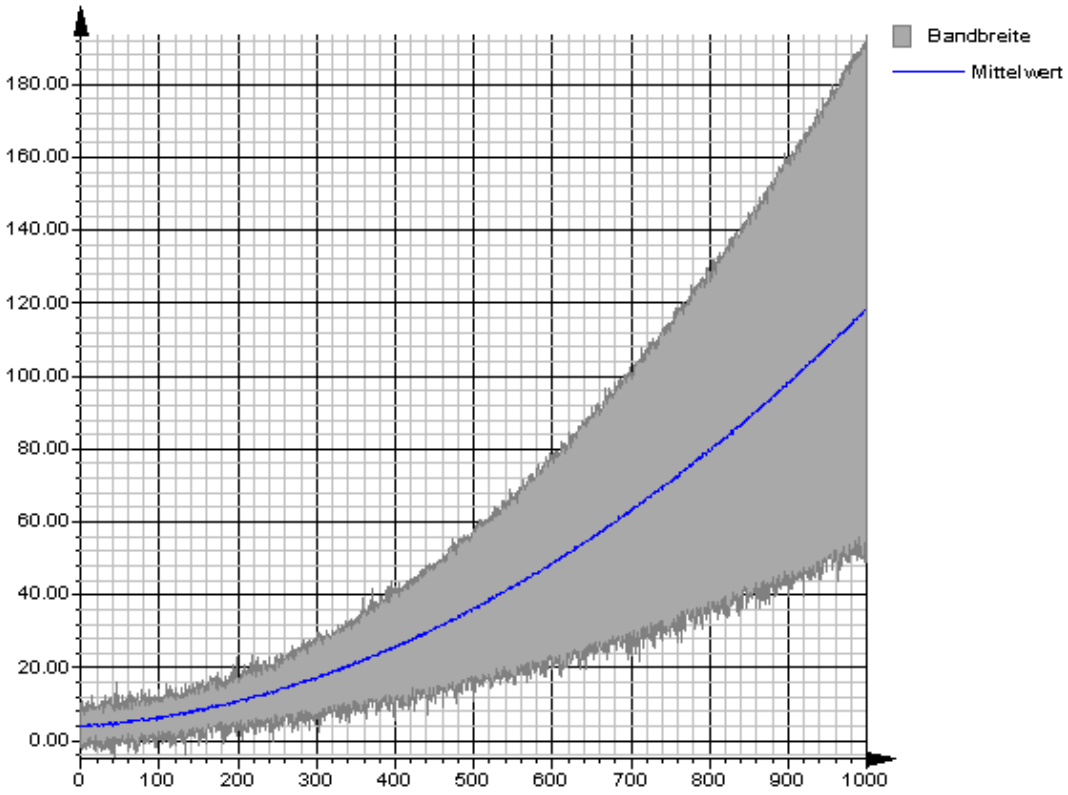
```
                                // hier werden alle Kurvendaten (Messdaten aus dem BLOB)
durchgelaufen
                                for (int i=0;i<len;i++)
                                {
                                // aktueller Datenwert Niederwertigstes Byte und
Höherwertiges Byte werden vertauscht,
                                // da diese im Blob in "falscher" Reinfolge in der DB
abgespeichert werden

                                        pD[i]=LVDouble(pV+8*i+4);

                                }
                                paintL(pD,len,((255/(num+7))%1)%255,255-
((num*77)%255),((128-num)*11)%255);
                        }
                        else
                        {
                                printf("Fehler!!!   Element zu klein oder zu groß Fehler im
Datenbestand\n");
                                printf("num=%i, sz=%i len=%i\n",num,sz,len);
                        }
                        RS.MoveNext(); // springe zur nächsten Ergebniszeile im
Abfrageergebnis
                }

                if (pD) free(pD);
                if (pV) free(pV);
                RS.Close();
        }
        else
                printf("Tabelle 'messdaten' nicht geöffnet\n");

        return len;
}
```

1.4.3.3.4 OnRun

Hierbei handelt es sich um den Einsprungpunkt für das X1.

```
int method::OnRun()
{
        printf("START\n");

        PUB.Clear();            // Diagramm leeren

        CDatabase DB();                 // Datenbank Objekt

        if (DB.Open("BLOB2",0,1,"ODBC;",1))   // Datenbank öffnen
                {
                printf("Datenbank 'BLOB2' geöffnet\n");

                // Textdialog am Programmanfang
                CText T;
                T.SetData("x-%");
                T.Edit("Bitte geben Sie die Suchabfrage ein");

                Kurven(&T,&DB);  // Zeichnet den Mittelwert, und das Band der Messungen
                DB.Close();     // Datenbank schließen
                }


        GetDocument()->Invalidate();
        return 1;
}
```

## 1.4.4 line




### 1.4.4.1 Kurzbeschreibung

Dieses Beispiel liest aus einer ODBC Datenquelle BLOBs (**B**inary **L**arge **Ob**jects) aus. Diese werden als Kurven interpretiert. Das gesamte Spektrum der Messwerte (Fläche zwischen Maximalwerten und Minimalwerte) wird grau markiert. Der Mittelwert aller Messungen wird ebenfalls gezeichnet.

### 1.4.4.2 Bedienungsanleitung

Sie starten dieses Beispiel mit der Run- Taste.

Nach dem Start des Beispiels werden Sie aufgefordert die Seriennummer der zu berücksichtigen Kurven einzugeben.

Hierbei wird ein so genanntes *wildcard*- Zeichen verwendet. Das funktioniert ähnlich wie das Symbol `"*"` in Dos oder Unix. Das *wildcard*- Zeichen ist das % Zeichen. Das % steht für eine beliebige Anzahl von Zeichen.

Mögliche Seriennummern:

- x-10000 bis x-10099
- y-10000 bis y-10099

Beispiele für die Eingabe:

- Alle Seriennummern die mit x beginnen: x%
- Alle Seriennummern die mit 9 aufhören: %9

- Alle Kurven: %

### 1.4.4.3 Skriptbeschreibung

Um das Skript zu bearbeiten müßen Sie sich in der Betriebsart Bearbeiten befinden. Drücken Sie nun mit der Rechten Maustaste auf **ROOTO** und wählen den Menue Punkt Skript an. Der Einsprungpunkt für das X1 ist die Methode OnRun.

#### 1.4.4.3.1 paintP

Diese Funktion zeichnet ein Polygon bestehend aus der "min" und "max" Kurve. Geschloßen wird dieses Polygon durch die Y-Achsen am Rande des Diagramms.

**Parameter**

| | |
|---|---|
| `double* min` | untere Begrenzungskurve |
| `double* max` | obere Begrenzungskurve |
| `int len` | Anzahl der Messwerte der Begrenzungskurven |
| `char* name` | Name des Polygons in der Legende |

```
void paintP(double* min, double* max,int len, char* name){

        double *poly=(double*)malloc(len*8*2);
        // hole Minwerte in das Polygon
        for (int m=0;m<len;m++)
                poly[m]=min[m];
        // drehe Maxwerte um und hänge sie an das Poligon an
        for (int j=0;j<len;j++)
                poly[j+len]=max[len-j-1];
        // erechne X werte
        double *x=(double*)malloc(len*8*2);
        for (int k=0;k<len;k++)
                x[k]=k;
        for (int l=len-1;l>=0;l--)
                x[l+(len-1)]=len-l;
        // zeichne das Polygon
        CPPolyTrace *pPP=new CPPolyTrace();
        pPP->SetTraceAtt(1240,3);                       //Layout Typ
        pPP->SetTraceAtt(4154,name);          // Namen des Polygons in der Legende
        pPP->SetTraceAtt(4150,"AX");          // legt die X Achse fest
        pPP->SetTraceAtt(4151,"AY");          // legt die Y Achse fest
        pPP->SetTraceAtt(6240,1,0,128,128,128);  // Linienart für die Umrandung des
Polygons
        pPP->SetTraceAtt(7240,0,0,169,169,169);  // Muster für die Füllung des Polygons
        pPP->SetTraceAtt(8240,169,169,169);   // Hintergrundfarbe für die Füllung
        pPP->SetTraceAtt(10040,x,2000);               // Anzahl Polygonkoordinaten in X-
Richtung
        pPP->SetTraceAtt(10041,poly,2000);            // Anzahl Polygonkoordinaten in Y-
Richtung
        pPP->SetTraceAtt(10042,2000.0,0.0,1L);        // Längen der einzelnen
Koordinatenvektoren bei PolyPolygonen
        PUB.AddTrace(pPP);                            // fügt das Polygon in die Legende ein

        free(x);
        free(poly);

        return;
}

void paintP(double* min, double* max,int len)
{
        paintP(min, max, len,"XXX");
        return;
}
```

#### 1.4.4.3.2 paintL

Diese Funktion zeichnet einen Graphen.

**Parameter:**

| | |
|---|---|
| `double* x` | *double* Array mit Messdaten (Graph) |
| `int len` | Länge des Array x |
| `int r` | Farbe des Graphen Rot Anteil |
| `int g` | Farbe des Graphen Grün Anteil |
| `int b` | Farbe des Graphen Blau Anteil |
| `char* name` | Name des Graphen, der in der Legende erscheinen soll |

```
void paintL(double* x, int len, int r, int g, int b, char* name){ // diese Funktion
zeichnet einen Graph

        CKurve2DTrace *mK=new CKurve2DTrace();
        mK->SetTraceAtt(1150,3);              // Dieses Attribut gibt an, mit welcher
Funktion die Stützpunkte einer Kurve interpoliert werden
        mK->SetTraceAtt(1152,300);            // Größe des Symbols zur Punktmarkierung in
[1/100 mm]
        mK->SetTraceAtt(4150,"AX");           // legt die X Achse fest
        mK->SetTraceAtt(4151,"AY");           // legt die Y Achse fest
        mK->SetTraceAtt(4154,name);           // Name des Graphen in der Legende
        mK->SetTraceAtt(10021,x,len);         //Y-Werte explizit
        mK->SetTraceAtt(10020,0.0,1.0,len);   //X-Werte implizit
        mK->SetTraceAtt(9220,"X1-Sym",0);     // Font und Symbol für die Markierung
        mK->SetTraceAtt(7220,0,0,r,g,b);      // Füllmuster für die Markierungssymbole
        mK->SetTraceAtt(6220,1,0,0,0,255);    // Linienart für die Umrandung der
Markierungssymbole
        mK->SetTraceAtt(6221,1,0,r,g,b);      // Linienart und Farbe für die Kurve
        PUB.AddTrace(mK);                                  // fügt den Graphen in die
Legende ein
        return;
}

void paintL(double* x, int len, int r, int g, int b){
        paintL(x,len,r,g,b,"XXX");
        return;
}
```

#### 1.4.4.3.3 LVDouble

Diese Funktion wandelt eine UNIX-Gleitkommazahl in das unter Windows gebräuchliche format um. Sie ändert die Byte- Reinfolge des *doubles*. Dies ist notwendig da die Bytes im Blob in umgekehrter Reihnfolge vorliegen (UNIX- Format).

**Parameter:**

`char* pC` double Wert im Unix Format

**Rückgabewert:**

`double` double Wert im Windows Format

```
double LVDouble(char* pC)
{
        char buf[8];
        buf[0]=pC[7];
        buf[1]=pC[6];
        buf[2]=pC[5];
        buf[3]=pC[4];
        buf[4]=pC[3];
        buf[5]=pC[2];
        buf[6]=pC[1];
        buf[7]=pC[0];
        void *pV=buf;

        return *(double*)pV;
}
```

#### 1.4.4.3.4 stdKurven

Dieses Beispiel liest aus einer ODBC Datenquelle BLOBs (**B**inary **L**arge **Ob**jects) aus. Diese werden als Kurven interpretiert. Die Maximalwerte und Minimalwerte dieser Kurven werden der Funktion paintP übergeben. Die Mittelwertskurvedieser Messkurven wird der Funktion paintL übergeben.

**Parameter**

| | |
|---|---|
| `CText *T` | Text Abfrage welche die Seriennummer der ausgewählten Kurven enthält |
| `CDatabase *pDB` | Datenquelle |

Rückgabewert

| | |
|---|---|
| `int` | Anzahl der Messungen in den BLOB's (Stützstellen) |

```
int stdKurven(CText *T,CDatabase *pDB)
{       // Diese Methode zeichnet Das Band der Messungen und den Mittelwert
        // Rückgabewert Anzahl der Stützstellen
        CDBVariant DV; // Virtueller Datentyp für MFC ODBC Klasse
        CRecordset RS(pDB);    // Ergebnistyp  der ODBC Klasse (Abfrage Ergebnis)

        char qry[256];         // SQL Abfrage String
        sprintf(qry,"SELECT * FROM messdaten where seriennummer like '%s'",T->GetData());
        int len = 0;           // Anzahl der Stützstellen

        if (RS.Open(2,qry,0) && !RS.IsBOF() && !RS.IsEOF())         //

/*
-mit pRS.Open wird die SQL anfrage an pRS und damit an die DB übergeben
-pRS wird geöffnet
-RS.IsBOF() damit wird geprüft ob die Aktuelle Position des Ergebniszeigers vor dem ersten
Ergebnis steht (Beginn of File)
-RS.IsBOF() damit wird geprüft ob die Aktuelle Position des Ergebniszeigers hinter dem
ersten Ergebnis steht (End of File)
!pRS.IsBOF() && !pRS.IsEOF() hiermit wird überprüft ob der Ergebniszeiger nicht vor und
```

```
nach den Ergebnissen gleichzeitig steht, dieser Fall das ist nur möglich wenn ein Ergebnis
der Länge 0 zurückkam
*/

        {
                int sz=0,num = -1;              // Laufvariable für Datensätze
                double *mittel=0;               // Array für spätere Mittelwertskurve
                double *min=0;          // Array für spätere Min Kurve
                double *max=0;          // Array für spätere Maxkurve
                char   *pV=0;                   // Aktueller Blob
                double* pD=0;                   // Aktuelle Stützstellen aus Blob

                while (!RS.IsEOF())     // Wiederhole solange bis letzter Datensatz ereicht
                {
                        num++;
                        printf("Tabelle 'messdaten' geöffnet\n");
                        // kopiert das Feld mit dem Index 1 aus dem aktuellen
Abfrageergebnis in die Variable pDV
                        RS.GetFieldValue(1,&DV);
                        if (num==0)
                                sz=DV.GetBinarySize(); // erechnet beim ersten Durchlauf die
Länge des Blobs
                        //wenn ein Datensatz mit falscher Länge in der DB sich befindet wird
dieser ausgelassen

                        if (sz=DV.GetBinarySize())
                        {
                                if (num==0)
                                {// hier werden die Arrays beim ersten Durchlauf reserviert

                                        // Anzahl der Messungen die sich im Blob befinden
                                        len=(sz-4)/8;
                                        pV=(char*)malloc(sz);
                                        min=(double*)malloc(len*8);           // Kurve der
Minimalwerte
                                        max=(double*)malloc(len*8);           // Kurve der
Max Werte
                                        mittel=(double*)malloc(len*8);        //
Mittelwertskurve
                                        pD=(double*)malloc(len*8);
                                }
                                //Hole Kurvendaten von DB in ein Array
                                DV.GetBinaryData(pV,sz);              // das ergebnis liegt
anschließend in pV

                                // hier werden alle Kurvendaten (Messdaten aus dem BLOB)
durchgelaufen
                                for (int i=0;i<len;i++)
                                {
                                        // aktueller Datenwert Niederwertigstes Byte und
Höherwertiges Byte werden vertauscht,
                                        // da diese im Blob in "falscher" Reinfolge in der DB
abgespeichert werden

                                        pD[i]=LVDouble(pV+8*i+4);

                                        // hier werden die Mittelwertskurve,  Min und Max
Kurve mit Daten gefüllt
                                        if (num==0)
                                        {
                                                min[i]=pD[i];
                                                max[i]=pD[i];
                                                mittel[i] = pD[i];
                                        }
                                        else
                                        {
                                                if (pD[i]<min[i]) // Minimal Kurve
                                                        min[i]=pD[i];
                                                if (pD[i]>max[i]) // Maximalkurve
                                                        max[i]=pD[i];
                                                mittel[i]=mittel[i]+pD[i];
                                        }
                                }
```

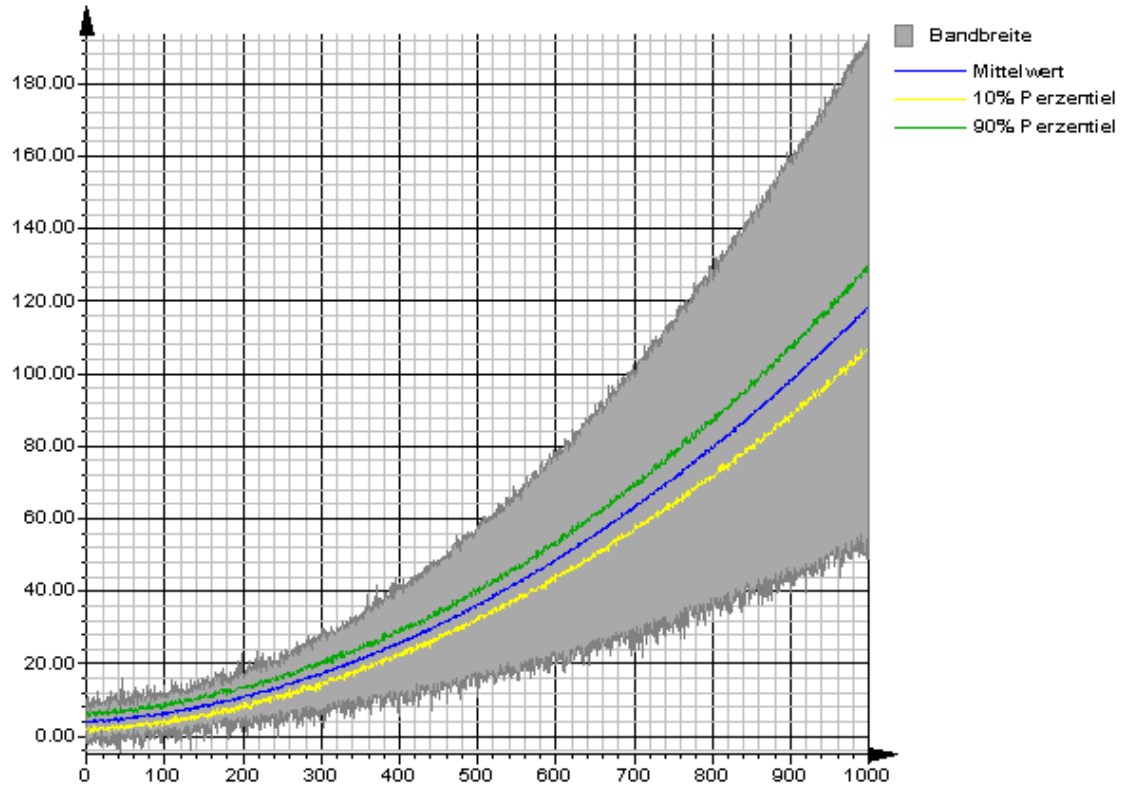
```
                        }
                        else
                        {
                                printf("Fehler!!!   Element zu klein oder zu groß Fehler im
Datenbestand\n");
                                printf("num=%i, sz=%i len=%i\n",num,sz,len);
                        }
                        RS.MoveNext(); // springe zur nächsten Ergebniszeile im
Abfrageergebnis
                }

                paintP(min,max, len,"Bandbreite");// Zeichne die Bandbreite der Messung

                for (int v=0;v<len;v++)// errechne den Mittelwert
                        mittel[v]=mittel[v]/(num+1);
                paintL(mittel,len,0,0,255,"Mittelwert"); // Zeichne den Mittelwert

                if (min) free(min);
                if (max) free(max);
                if (mittel) free(mittel);
                if (pD) free(pD);
                if (pV) free(pV);
                RS.Close();
        }
        else
                printf("Tabelle 'messdaten' nicht geöffnet\n");

        return len;
}
```

1.4.4.3.5 OnRun

Hierbei handelt es sich um den Einsprungpunkt für das X1.

```
int method::OnRun()
{
        printf("START\n");

        PUB.Clear();            // Diagramm leeren

        CDatabase DB;           // Datenbank Objekt

        if (DB.Open("BLOB2",0,1,"ODBC;",1))   // Datenbank öffnen
                {
                printf("Datenbank 'BLOB2' geöffnet\n");

                // Textdialog am Programmanfang
                CText T;
                T.SetData("x-%");
                T.Edit("Bitte geben Sie die Suchabfrage ein");

                stdKurven(&T,&DB);  // Zeichnet den Mittelwert, und das Band der Messungen
                DB.Close();
                }



        GetDocument()->Invalidate();
        return 1;
}
```

## 1.4.5 90p




### 1.4.5.1 Kurzbeschreibung

Dieses Beispiel liest aus einer ODBC Datenquelle BLOBs (**B**inary **L**arge **Ob**jects) aus. Diese werden als Kurven interpretiert. Das gesamte Spektrum der Messwerte (Fläche zwischen Maximalwerten und Minimalwerte) wird grau markiert. Der Mittelwert sowie eine 90% und eine 10% Perzentiele werden ebenfalls gezeichnet.

### 1.4.5.2 Bedienungsanleitung

Sie starten dieses Beispiel mit der Run- Taste.

Nach dem Start des Beispiels werden Sie aufgefordert die Seriennummer der zu berücksichtigen Kurven einzugeben.

Hierbei wird ein so genanntes *wildcard*- Zeichen verwendet. Das funktioniert wirkt ähnlich wie das "*" in Dos oder Unix. Das *wildcard*- Zeichen ist das % Zeichen. Das % steht für eine beliebige Anzahl von Zeichen.

Mögliche Seriennummern:

- x-10000 bis x-10099
- y-10000 bis y-10099

Beispiele für die Eingabe:

- Alle Seriennummern die mit x beginnen: x%

- Alle Seriennummern die mit 9 aufhören: %9
- Alle Kurven: %

### 1.4.5.3 Skriptbeschreibung

Um das Skript zu bearbeiten müßen Sie sich in der Betriebsart Bearbeiten befinden. Drücken Sie nun mit der Rechten Maustaste auf **ROOTO** und wählen den Menue Punkt Skript an. Der Einsprungpunkt für das X1 ist die Methode OnRun.

1.4.5.3.1 paintP

Diese Funktion zeichnet ein Polygon bestehend aus der "min" und "max" Kurve. Geschloßen wird dieses Polygon durch die Y-Achsen am Rande des Diagramms.

**Parameter**

| | |
|---|---|
| `double* min` | untere Begrenzungskurve |
| `double* max` | obere Begrenzungskurve |
| `int len` | Anzahl der Messwerte der Begrenzungskurven |
| `char* name` | Name des Polygons in der Legende |

```
Void paintP(double* min, double* max,int len, char* name){  //

        double *poly=(double*)malloc(len*8*2);
        // hole Minwerte in das Polygon
        for (int m=0;m<len;m++)
                poly[m]=min[m];
        // drehe Maxwerte um und hänge sie an das Poligon an
        for (int j=0;j<len;j++)
                poly[j+len]=max[len-j-1];
        // erechne X werte
        double *x=(double*)malloc(len*8*2);
        for (int k=0;k<len;k++)
                x[k]=k;
        for (int l=len-1;l>=0;l--)
                x[l+(len-1)]=len-l;
        // zeichne das Polygon
        CPPolyTrace *pPP=new CPPolyTrace();
        pPP->SetTraceAtt(1240,3);                       //Layout Typ
        pPP->SetTraceAtt(4154,name);          // Namen des Polygons in der Legende
        pPP->SetTraceAtt(4150,"AX");          // legt die X Achse fest
        pPP->SetTraceAtt(4151,"AY");          // legt die Y Achse fest
        pPP->SetTraceAtt(6240,1,0,128,128,128); // Linienart für die Umrandung des Polygons
        pPP->SetTraceAtt(7240,0,0,169,169,169); // Muster für die Füllung des Polygons
        pPP->SetTraceAtt(8240,169,169,169);   // Hintergrundfarbe für die Füllung
        pPP->SetTraceAtt(10040,x,2000);               // Anzahl Polygonkoordinaten in X-
Richtung
        pPP->SetTraceAtt(10041,poly,2000);            // Anzahl Polygonkoordinaten in Y-
Richtung
        pPP->SetTraceAtt(10042,2000.0,0.0,1L);        // Längen der einzelnen
Koordinatenvektoren bei PolyPolygonen
        PUB.AddTrace(pPP);                                      // fügt das Polygon in
die Legende ein

        free(x);
        free(poly);

        return;
}

void paintP(double* min, double* max,int len)
{
        paintP(min, max, len,"XXX");
        return;
}
```

1.4.5.3.2 paintL

Diese Funktion zeichnet einen Graphen.

**Parameter:**

| | |
|---|---|
| `double* x` | *double* Array mit Messdaten (Graph) |
| `int len` | Länge des Array x |
| `int r` | Farbe des Graphen Rot Anteil |
| `int g` | Farbe des Graphen Grün Anteil |
| `int b` | Farbe des Graphen Blau Anteil |
| `char* name` | Name des Graphen, der in der Legende erscheinen soll |

```
void paintL(double* x, int len, int r, int g, int b, char* name){ // diese Funktion
zeichnet einen Graph
        // x Messwerte
        // r,g,b Farbe des Graphen
        // len Anzahl der Stützstellen
        // name Name des Graphen in der Legende

        CKurve2DTrace *mK=new CKurve2DTrace();
        mK->SetTraceAtt(1150,3);                    // Dieses Attribut gibt an, mit
welcher Funktion die Stützpunkte einer Kurve interpoliert werden
        mK->SetTraceAtt(1152,300);                  // Größe des Symbols zur
Punktmarkierung in [1/100 mm]
        mK->SetTraceAtt(4150,"AX");                 // legt die X Achse fest
        mK->SetTraceAtt(4151,"AY");                 // legt die Y Achse fest
        mK->SetTraceAtt(4154,name);                 // Name des Graphen in der Legende
        mK->SetTraceAtt(10021,x,len);         //Y-Werte explizit
        mK->SetTraceAtt(10020,0.0,1.0,len);   //X-Werte implizit
        mK->SetTraceAtt(9220,"X1-Sym",0);     // Font und Symbol für die Markierung
        mK->SetTraceAtt(7220,0,0,r,g,b);      // Füllmuster für die Markierungssymbole
        mK->SetTraceAtt(6220,1,0,0,0,255);    // Linienart für die Umrandung der
Markierungssymbole
        mK->SetTraceAtt(6221,1,0,r,g,b);      // Linienart und Farbe für die Kurve
        PUB.AddTrace(mK);                                  // fügt den Graphen in die
Legende ein
        return;
}

void paintL(double* x, int len, int r, int g, int b){
        paintL(x,len,r,g,b,"XXX");
        return;
}
```

#### 1.4.5.3.3 LVDouble

Diese Funktion wandelt eine UNIX-Gleitkommazahl in das unter Windows gebräuchliche format um. Sie ändert die Byte- Reinfolge des *doubles*. Dies ist notwendig da die Bytes im Blob in umgekehrter Reihnfolge vorliegen (UNIX- Format).

**Parameter:**

`char* pC` double Wert im Unix Format

**Rückgabewert:**

`double` double Wert im Windows Format

```
double LVDouble(char* pC)
{
        char buf[8];
        buf[0]=pC[7];
        buf[1]=pC[6];
        buf[2]=pC[5];
        buf[3]=pC[4];
        buf[4]=pC[3];
        buf[5]=pC[2];
        buf[6]=pC[1];
        buf[7]=pC[0];
        void *pV=buf;

        return *(double*)pV;
}
```

#### 1.4.5.3.4 stdKurven

Dieses Beispiel liest aus einer ODBC Datenquelle BLOBs (**B**inary **L**arge **Ob**jects) aus. Diese werden als Kurven interpretiert. Die Maximalwerte und Minimalwerte dieser Kurven werden der Funktion paintP übergeben. Die Mittelwertskurvedieser Messkurven wird der Funktion paintL übergeben.

**Parameter**

| | |
|---|---|
| `CText *T` | Text Abfrage welche die Seriennummer der ausgewählten Kurven enthält |
| `CDatabase *pDB` | Datenquelle |

Rückgabewert

| | |
|---|---|
| `int` | Anzahl der Messungen in den BLOB's (Stützstellen) |

```
int stdKurven(CText *T,CDatabase *pDB)
{       // Diese Methode zeichnet Das Band der Messungen und den Mittelwert
        // Rückgabewert Anzahl der Stützstellen
        CDBVariant DV; // Virtueller Datentyp für MFC ODBC Klasse
        CRecordset RS(pDB);    // Ergebnistyp  der ODBC Klasse (Abfrage Ergebnis)

        char qry[256];         // SQL Abfrage String
        sprintf(qry,"SELECT * FROM messdaten where seriennummer like '%s'",T->GetData());
        int len = 0;           // Anzahl der Stützstellen

        if (RS.Open(2,qry,0) && !RS.IsBOF() && !RS.IsEOF())         //

/*
-mit pRS.Open wird die SQL anfrage an pRS und damit an die DB übergeben
-pRS wird geöffnet
-RS.IsBOF() damit wird geprüft ob die Aktuelle Position des Ergebniszeigers vor dem ersten
Ergebnis steht (Beginn of File)
-RS.IsBOF() damit wird geprüft ob die Aktuelle Position des Ergebniszeigers hinter dem
ersten Ergebnis steht (End of File)
!pRS.IsBOF() && !pRS.IsEOF() hiermit wird überprüft ob der Ergebniszeiger nicht vor und
nach den Ergebnissen gleichzeitig steht, dieser Fall das ist nur möglich wenn ein Ergebnis
```

```
der Länge 0 zurückkam
*/

	{
		int sz=0,num = -1;		// Laufvariable für Datensätze
		double *mittel=0;		// Array für spätere Mittelwertskurve
		double *min=0;		// Array für spätere Min Kurve
		double *max=0;		// Array für spätere Maxkurve
		char   *pV=0;			// Aktueller Blob
		double* pD=0;			// Aktuelle Stützstellen aus Blob

		while (!RS.IsEOF())	// Wiederhole solange bis letzter Datensatz ereicht
		{
			num++;
			printf("Tabelle 'messdaten' geöffnet\n");
			// kopiert das Feld mit dem Index 1 aus dem aktuellen
Abfrageergebnis in die Variable pDV
			RS.GetFieldValue(1,&DV);
			if (num==0)
				sz=DV.GetBinarySize(); // erechnet beim ersten Durchlauf die
Länge des Blobs
			//wenn ein Datensatz mit falscher Länge in der DB sich befindet wird
dieser ausgelassen

			if (sz=DV.GetBinarySize())
			{
				if (num==0)
				{// hier werden die Arrays beim ersten Durchlauf reserviert

					// Anzahl der Messungen die sich im Blob befinden
					len=(sz-4)/8;
					pV=(char*)malloc(sz);
					min=(double*)malloc(len*8);		// Kurve der
Minimalwerte
					max=(double*)malloc(len*8);		// Kurve der
Max Werte
					mittel=(double*)malloc(len*8);		//
Mittelwertskurve
					pD=(double*)malloc(len*8);
				}
				//Hole Kurvendaten von DB in ein Array
				DV.GetBinaryData(pV,sz);		// das ergebnis liegt
anschließend in pV

				// hier werden alle Kurvendaten (Messdaten aus dem BLOB)
durchgelaufen
				for (int i=0;i<len;i++)
				{
					// aktueller Datenwert Niederwertigstes Byte und
Höherwertiges Byte werden vertauscht,
					// da diese im Blob in "falscher" Reinfolge in der DB
abgespeichert werden

					pD[i]=LVDouble(pV+8*i+4);

					// hier werden die Mittelwertskurve, Min und Max
Kurve mit Daten gefüllt
					if (num==0)
					{
						min[i]=pD[i];
						max[i]=pD[i];
						mittel[i] = pD[i];
					}
					else
					{
						if (pD[i]<min[i]) // Minimal Kurve
							min[i]=pD[i];
						if (pD[i]>max[i]) // Maximalkurve
							max[i]=pD[i];
						mittel[i]=mittel[i]+pD[i];
					}
				}
			}
```

```
                        else
                        {
                                printf("Fehler!!!   Element zu klein oder zu groß Fehler im
Datenbestand\n");
                                printf("num=%i, sz=%i len=%i\n",num,sz,len);
                        }
                        RS.MoveNext(); // springe zur nächsten Ergebniszeile im
Abfrageergebnis
                }

                paintP(min,max, len,"Bandbreite");// Zeichne die Bandbreite der Messung

                for (int v=0;v<len;v++)// errechne den Mittelwert
                        mittel[v]=mittel[v]/(num+1);
                paintL(mittel,len,0,0,255,"Mittelwert"); // Zeichne den Mittelwert

                if (min) free(min);
                if (max) free(max);
                if (mittel) free(mittel);
                if (pD) free(pD);
                if (pV) free(pV);
                RS.Close();
        }
        else
                printf("Tabelle 'messdaten' nicht geöffnet\n");

        return len;
}
```

1.4.5.3.5 countM

Diese Funktion berechnet die Anzahl der Messungen (BLOBs).
Mittels einer SQL Abfrage wird das DBMS (Datenbank Management System) aufgefordert die Anzahl der Datensätze zu zählen. Auf den Rückgabe Wert kann mittels `m_lVal` direkt zugegriffen werden, da das DBMS nur einen *int* wert zurück gibt.

**Parameter**

| | |
|---|---|
| `CText *T` | Text Abfrage welche die Seriennummer der ausgewählten Kurven enthält |
| `CDatabase *pDB` | Datenquelle |

Rückgabewert

| | |
|---|---|
| `int` | Anzahl der Blobs |

```
int countM(CText *T,CDatabase *pDB){  // diese Funktion zählt die Anzahl der Messungen

        CDBVariant DV;                  // Virtueller Datentyp für MFC ODBC Klasse
        CRecordset RS(pDB);                     // Ergebnistyp  der ODBC Klasse (Abfrage
Ergebnis)
        int ret = 0;                            // Rückgabewert der Methode
        char qry[256];                  // SQL String der Abfrage
        sprintf(qry,"SELECT Count(messdaten.seriennummer) "
                                "FROM messdaten "
                                "WHERE seriennummer like '%s'",
                                T->GetData());
        if(RS.Open(2,qry,0))            //mit pRS->Open wird die SQL anfrage an pRS und damit
an die DB übergeben
        {
                RS.GetFieldValue(0,&DV);        // Hole das Feld mit index 0 aus dem
Abfrageergebnis heraus
                printf("Es gibt: %i Messungen\n",DV.m_lVal);
                ret=DV.m_lVal;                  // auf den Primitiv type Int kann mit m:lVal
direkt zugegriffen werden
                RS.Close();
        }
        else printf("Fehler konnte DB in Funktion countM() nicht öffnen");
```

```
return ret;
}
```

1.4.5.3.6 p90

Diese Funktion zeichnet die 90% und die 10% Perzentiele(Bereich in dem 90% bzw. 10% aller Messungen liegen) aller Messungen in das Diagramm ein.

**Parameter**

| | |
|---|---|
| `CText *T` | Text Abfrage welche die Seriennummer der ausgewählten Kurven enthält |
| `CDatabase *pDB` | Datenquelle |
| `int len` | Anzahl der Messwerte pro BLOB |

```
void p90(CText *T,CDatabase *pDB,int len)
{
	// diese Funktion zeichnet das 90% Perzentiel
	CDBVariant DV; // Virtueller Datentyp der ODBC Klasse
	CRecordset RS(pDB);    // Ergebnistyp  der ODBC Klasse (Abfrage Ergebnis)
	char qry[256];         // SQL String
	sprintf(qry,"SELECT * "
	                        "FROM messdaten "
	                        "WHERE seriennummer like '%s'",
	                        T->GetData());

	if (RS.Open(2,qry,0) &&
		!RS.IsBOF() &&
		!RS.IsEOF())
	{

		int num= -1; // Laufvariable der Messungen
		int anzahl=countM(T,pDB);     // Anzahl der Messungen
		double** d =(double**)malloc(len*sizeof(double*));//2 Dimensionales Array
zum sichern der Messdaten
		for (int i=0;i<len;i++)
		  d[i]=(double*)malloc(anzahl*sizeof(double));
		int sz = len*8+4;                    // Länge des Blobs
		char* pV=0;


		while(!RS.IsEOF()) // diese Schleife läuft alle Messungen in der DB durch
		{

			num ++;
			// hole die Daten aus dem Feld mit Index 1 und Kopiere Sie  in die
Variable pDV
			RS.GetFieldValue(1,&DV);

			if ((sz==DV.GetBinarySize()))  // fals ein Blob mit falscher Größe
kommt, wird dieser ausgelassen
			{
			// hole Kurvendaten in neues Array d

				pV=(char*)malloc(sz);
				DV.GetBinaryData(pV,sz);
					for (int i=0;i<len;i++){
						d[i][num]=LVDouble(pV+8*i+4); //aktueller
Datensatz
					}
				if (pV) free(pV);

			}
			else printf("Datenbank Zugriffsfehler in funktin p90");
			if (pV) free(pV);
			//pDV = aktueller Datensatz
			RS.MoveNext();

		}
		if (pV) free(pV);


		CVector V;
```

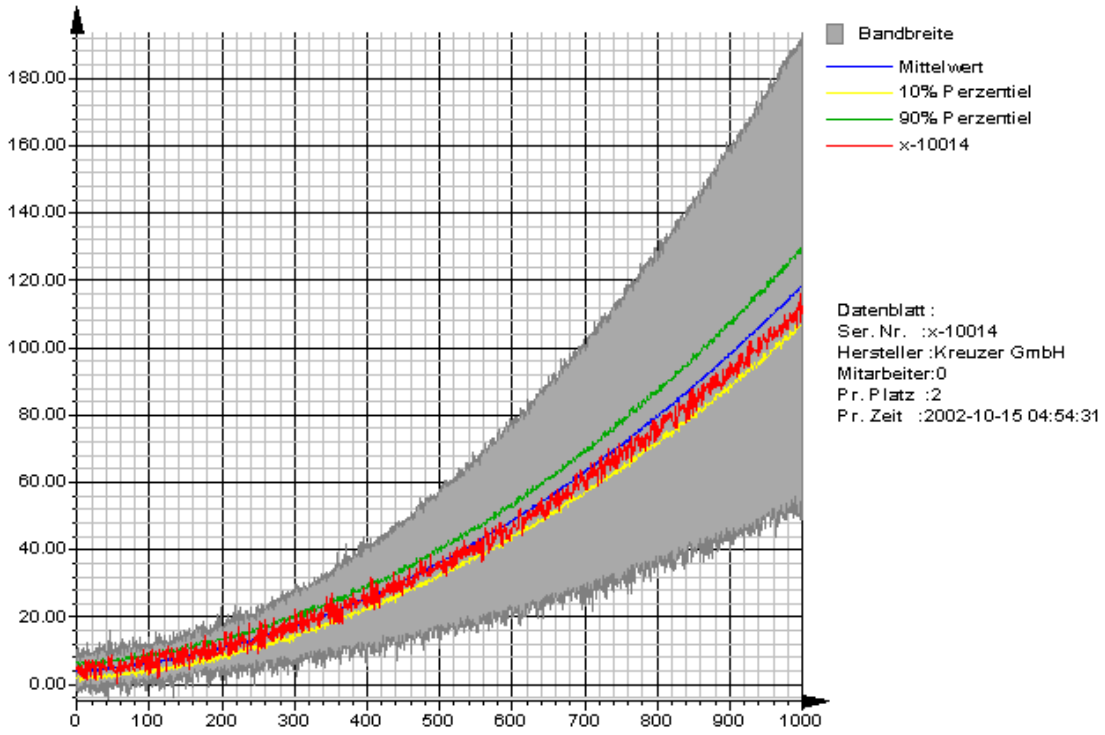
```
                double p10,p90;
                double *untereGrenze=(double*)malloc(len*8);  // Graph für die 10 % P.
                double *obereGrenze=(double*)malloc(len*8);    // Graph für die 90 % P.
                for (i=0;i<len;i++)  // Laufe alle Stützpunkte durch
                {
                        V.SetData(anzahl,d[i]);
                        V.Median(&p10,10.0,&p90,90.0,0L);
                        untereGrenze[i]=p10;
                        obereGrenze[i]=p90;
                }
                paintL(untereGrenze,len,255,255,0,"10% Perzentiel");
                paintL(obereGrenze,len,0,170,0,"90% Perzentiel");

                // aufräumen
                if (untereGrenze)free(untereGrenze);
                if (obereGrenze) free(obereGrenze);

                for (i=0;i<len;i++)
                {
                        if (d[i]!=0)
                        {
                                free(d[i]);
                        }
                }
                if (d)
                        free(d);
                RS.Close();
        }
        else printf("Fehler konnte DB in Funktion p90() nicht öffnen\n");

        return;
        }
```

#### 1.4.5.3.7 OnRun

Hierbei handelt es sich um den Einsprungpunkt für das X1.

```
int method::OnRun()
{
        printf("START\n");

        PUB.Clear();          // Diagramm leeren

        CDatabase DB;         // Datenbank Objekt

        if (DB.Open("BLOB2",0,1,"ODBC;",1))   // Datenbank öffnen
                {
                printf("Datenbank 'BLOB2' geöffnet\n");

                // Textdialog am Programmanfang
                CText T;
                T.SetData("x-%");
                T.Edit("Bitte geben Sie die Suchabfrage ein");

                int len=stdKurven(&T,&DB);  // Zeichnet den Mittelwert, und das Band der
Messungen
                p90(&T,&DB,len);              // zeichne 90% Perzentiele
                DB.Close();

                }


        GetDocument()->Invalidate();
        return 1;
}
```

## 1.4.6 artikel




### 1.4.6.1 Kurzbeschreibung

Dieses Beispiel liest aus einer ODBC Datenquelle BLOBs (**B**inary **L**arge **Ob**jects) aus. Diese werden als Kurven interpretiert. Das gesamte Spektrum der Messwerte (Fläche zwischen Maximalwerten und Minimalwerte) wird grau markiert. Der Mittelwert sowie eine 90% und eine 10% Perzentiele werden ebenfalls gezeichnet. Desweiteren wird eine speziell ausgewählte Kurve gezeichnet.

### 1.4.6.2 Bedienungsanleitung

Sie starten dieses Beispiel mit der Run- Taste.

Nach dem Start des Beispiels werden Sie aufgefordert die Seriennummer der zu berücksichtigen Kurven einzugeben.

Hierbei wird ein so genanntes *wildcard*- Zeichen verwendet. Das funktioniert wirkt ähnlich wie das "*" in Dos oder Unix. Das *wildcard*- Zeichen ist das % Zeichen. Das % steht für eine beliebige Anzahl von Zeichen.

Mögliche Seriennummern:

- x-10000 bis x-10099
- y-10000 bis y-10099

Beispiele für die Eingabe:

- Alle Seriennummern die mit x beginnen: x%
- Alle Seriennummern die mit 9 aufhören: %9
- Alle Kurven: %

Anschließend werden Sie aufgefordert eine weitere Seriennummer einzugeben. Hierbei muß eine vollständige und existierende Seriennummer eingegeben werden. Das verwenden von *wildcard*-Zeichen ist nicht möglich. Die Daten der hier ausgewählten Seriennummer werden zusätzlich zu den zuvor ausgewählten Daten ausgegeben.

### 1.4.6.3 Skriptbeschreibung

Um das Skript zu bearbeiten müßen Sie sich in der Betriebsart Bearbeiten befinden. Drücken Sie nun mit der Rechten Maustaste auf **ROOTO** und wählen den Menue Punkt Skript an. Der Einsprungpunkt für das X1 ist die Methode OnRun.

1.4.6.3.1 paintP

Diese Funktion zeichnet ein Polygon bestehend aus der "min" und "max" Kurve. Geschloßen wird dieses Polygon durch die Y-Achsen am Rande des Diagramms.

**Parameter**

| | |
|---|---|
| `double* min` | untere Begrenzungskurve |
| `double* max` | obere Begrenzungskurve |
| `int len` | Anzahl der Messwerte der Begrenzungskurven |
| `char* name` | Name des Polygons in der Legende |

```
Void paintP(double* min, double* max,int len, char* name){  //

        double *poly=(double*)malloc(len*8*2);
        // hole Minwerte in das Polygon
        for (int m=0;m<len;m++)
                poly[m]=min[m];
        // drehe Maxwerte um und hänge sie an das Poligon an
        for (int j=0;j<len;j++)
                poly[j+len]=max[len-j-1];
        // erechne X werte
        double *x=(double*)malloc(len*8*2);
        for (int k=0;k<len;k++)
                x[k]=k;
        for (int l=len-1;l>=0;l--)
                x[l+(len-1)]=len-l;
        // zeichne das Polygon
        CPPolyTrace *pPP=new CPPolyTrace();
        pPP->SetTraceAtt(1240,3);                       //Layout Typ
        pPP->SetTraceAtt(4154,name);          // Namen des Polygons in der Legende
        pPP->SetTraceAtt(4150,"AX");          // legt die X Achse fest
        pPP->SetTraceAtt(4151,"AY");          // legt die Y Achse fest
        pPP->SetTraceAtt(6240,1,0,128,128,128); // Linienart für die Umrandung des Polygons
        pPP->SetTraceAtt(7240,0,0,169,169,169); // Muster für die Füllung des Polygons
        pPP->SetTraceAtt(8240,169,169,169);   // Hintergrundfarbe für die Füllung
        pPP->SetTraceAtt(10040,x,2000);               // Anzahl Polygonkoordinaten in X-
Richtung
        pPP->SetTraceAtt(10041,poly,2000);            // Anzahl Polygonkoordinaten in Y-
Richtung
        pPP->SetTraceAtt(10042,2000.0,0.0,1L);        // Längen der einzelnen
Koordinatenvektoren bei PolyPolygonen
        PUB.AddTrace(pPP);                                      // fügt das Polygon in
die Legende ein

        free(x);
        free(poly);

        return;
}

void paintP(double* min, double* max,int len)
{
        paintP(min, max, len,"XXX");
        return;
}
```

1.4.6.3.2 paintL

Diese Funktion zeichnet einen Graphen.

**Parameter:**

| | |
|---|---|
| `double* x` | *double* Array mit Messdaten (Graph) |
| `int len` | Länge des Array x |
| `int r` | Farbe des Graphen Rot Anteil |
| `int g` | Farbe des Graphen Grün Anteil |
| `int b` | Farbe des Graphen Blau Anteil |
| `char* name` | Name des Graphen, der in der Legende erscheinen soll |

```
void paintL(double* x, int len, int r, int g, int b, char* name){ // diese Funktion
zeichnet einen Graph
        // x Messwerte
        // r,g,b Farbe des Graphen
        // len Anzahl der Stützstellen
        // name Name des Graphen in der Legende

        CKurve2DTrace *mK=new CKurve2DTrace();
        mK->SetTraceAtt(1150,3);                    // Dieses Attribut gibt an, mit
welcher Funktion die Stützpunkte einer Kurve interpoliert werden
        mK->SetTraceAtt(1152,300);                  // Größe des Symbols zur
Punktmarkierung in [1/100 mm]
        mK->SetTraceAtt(4150,"AX");                 // legt die X Achse fest
        mK->SetTraceAtt(4151,"AY");                 // legt die Y Achse fest
        mK->SetTraceAtt(4154,name);                 // Name des Graphen in der Legende
        mK->SetTraceAtt(10021,x,len);         //Y-Werte explizit
        mK->SetTraceAtt(10020,0.0,1.0,len);   //X-Werte implizit
        mK->SetTraceAtt(9220,"X1-Sym",0);     // Font und Symbol für die Markierung
        mK->SetTraceAtt(7220,0,0,r,g,b);      // Füllmuster für die Markierungssymbole
        mK->SetTraceAtt(6220,1,0,0,0,255);    // Linienart für die Umrandung der
Markierungssymbole
        mK->SetTraceAtt(6221,1,0,r,g,b);      // Linienart und Farbe für die Kurve
        PUB.AddTrace(mK);                                  // fügt den Graphen in die
Legende ein
        return;
}

void paintL(double* x, int len, int r, int g, int b){
        paintL(x,len,r,g,b,"XXX");
        return;
}
```

#### 1.4.6.3.3 LVDouble

Diese Funktion wandelt eine UNIX-Gleitkommazahl in das unter Windows gebräuchliche format um. Sie ändert die Byte- Reinfolge des *doubles*. Dies ist notwendig da die Bytes im Blob in umgekehrter Reihnfolge vorliegen (UNIX- Format).

**Parameter:**

`char* pC` double Wert im Unix Format

**Rückgabewert:**

`double` double Wert im Windows Format

```
double LVDouble(char* pC)
{
        char buf[8];
        buf[0]=pC[7];
        buf[1]=pC[6];
        buf[2]=pC[5];
        buf[3]=pC[4];
        buf[4]=pC[3];
        buf[5]=pC[2];
        buf[6]=pC[1];
        buf[7]=pC[0];
        void *pV=buf;

        return *(double*)pV;
}
```

#### 1.4.6.3.4 stdKurven

Dieses Beispiel liest aus einer ODBC Datenquelle BLOBs (**B**inary **L**arge **Ob**jects) aus. Diese werden als Kurven interpretiert. Die Maximalwerte und Minimalwerte dieser Kurven werden der Funktion paintP übergeben. Die Mittelwertskurvedieser Messkurven wird der Funktion paintL übergeben.

**Parameter**

| | |
|---|---|
| `CText *T` | Text Abfrage welche die Seriennummer der ausgewählten Kurven enthält |
| `CDatabase *pDB` | Datenquelle |

Rückgabewert

| | |
|---|---|
| `int` | Anzahl der Messungen in den BLOB's (Stützstellen) |

```
int stdKurven(CText *T,CDatabase *pDB)
{       // Diese Methode zeichnet Das Band der Messungen und den Mittelwert
        // Rückgabewert Anzahl der Stützstellen
        CDBVariant DV; // Virtueller Datentyp für MFC ODBC Klasse
        CRecordset RS(pDB);    // Ergebnistyp  der ODBC Klasse (Abfrage Ergebnis)

        char qry[256];         // SQL Abfrage String
        sprintf(qry,"SELECT * FROM messdaten where seriennummer like '%s'",T->GetData());
        int len = 0;           // Anzahl der Stützstellen

        if (RS.Open(2,qry,0) && !RS.IsBOF() && !RS.IsEOF())         //

/*
-mit pRS.Open wird die SQL anfrage an pRS und damit an die DB übergeben
-pRS wird geöffnet
-RS.IsBOF() damit wird geprüft ob die Aktuelle Position des Ergebniszeigers vor dem ersten
Ergebnis steht (Beginn of File)
-RS.IsBOF() damit wird geprüft ob die Aktuelle Position des Ergebniszeigers hinter dem
ersten Ergebnis steht (End of File)
!pRS.IsBOF() && !pRS.IsEOF() hiermit wird überprüft ob der Ergebniszeiger nicht vor und
nach den Ergebnissen gleichzeitig steht, dieser Fall das ist nur möglich wenn ein Ergebnis
```

```
der Länge 0 zurückkam
*/

	{
		int sz=0,num = -1;		// Laufvariable für Datensätze
		double *mittel=0;		// Array für spätere Mittelwertskurve
		double *min=0;		// Array für spätere Min Kurve
		double *max=0;		// Array für spätere Maxkurve
		char   *pV=0;			// Aktueller Blob
		double* pD=0;			// Aktuelle Stützstellen aus Blob

		while (!RS.IsEOF())	// Wiederhole solange bis letzter Datensatz ereicht
		{
			num++;
			printf("Tabelle 'messdaten' geöffnet\n");
			// kopiert das Feld mit dem Index 1 aus dem aktuellen
Abfrageergebnis in die Variable pDV
			RS.GetFieldValue(1,&DV);
			if (num==0)
				sz=DV.GetBinarySize(); // erechnet beim ersten Durchlauf die
Länge des Blobs
			//wenn ein Datensatz mit falscher Länge in der DB sich befindet wird
dieser ausgelassen

			if (sz=DV.GetBinarySize())
			{
				if (num==0)
				{// hier werden die Arrays beim ersten Durchlauf reserviert

					// Anzahl der Messungen die sich im Blob befinden
					len=(sz-4)/8;
					pV=(char*)malloc(sz);
					min=(double*)malloc(len*8);		// Kurve der
Minimalwerte
					max=(double*)malloc(len*8);		// Kurve der
Max Werte
					mittel=(double*)malloc(len*8);		//
Mittelwertskurve
					pD=(double*)malloc(len*8);
				}
				//Hole Kurvendaten von DB in ein Array
				DV.GetBinaryData(pV,sz);		// das ergebnis liegt
anschließend in pV

				// hier werden alle Kurvendaten (Messdaten aus dem BLOB)
durchgelaufen
				for (int i=0;i<len;i++)
				{
					// aktueller Datenwert Niederwertigstes Byte und
Höherwertiges Byte werden vertauscht,
					// da diese im Blob in "falscher" Reinfolge in der DB
abgespeichert werden

					pD[i]=LVDouble(pV+8*i+4);

					// hier werden die Mittelwertskurve, Min und Max
Kurve mit Daten gefüllt
					if (num==0)
					{
						min[i]=pD[i];
						max[i]=pD[i];
						mittel[i] = pD[i];
					}
					else
					{
						if (pD[i]<min[i]) // Minimal Kurve
							min[i]=pD[i];
						if (pD[i]>max[i]) // Maximalkurve
							max[i]=pD[i];
						mittel[i]=mittel[i]+pD[i];
					}
				}
			}
```

```
                        else
                        {
                                printf("Fehler!!!   Element zu klein oder zu groß Fehler im
Datenbestand\n");
                                printf("num=%i, sz=%i len=%i\n",num,sz,len);
                        }
                        RS.MoveNext(); // springe zur nächsten Ergebniszeile im
Abfrageergebnis
                }

                paintP(min,max, len,"Bandbreite");// Zeichne die Bandbreite der Messung

                for (int v=0;v<len;v++)// errechne den Mittelwert
                        mittel[v]=mittel[v]/(num+1);
                paintL(mittel,len,0,0,255,"Mittelwert"); // Zeichne den Mittelwert

                if (min) free(min);
                if (max) free(max);
                if (mittel) free(mittel);
                if (pD) free(pD);
                if (pV) free(pV);
                RS.Close();
        }
        else
                printf("Tabelle 'messdaten' nicht geöffnet\n");

        return len;
}
```

#### 1.4.6.3.5 oneKurven

Läd einen BLOB aus der ODBC Datenquelle. Ermittelt aus dem BLOB eine Kurve und zeichnet diese.

**Parameter**

| | |
|---|---|
| `CText *T` | Text Abfrage welche die Seriennummer der zu zeichnen Kurve enthält |
| `CDatabase *pDB` | Datenquelle |

**Rückgabewert**

| | |
|---|---|
| `int` | Anzahl der Messungen in den BLOB's (Stützstellen) |

```
int oneKurven(CText *T,CDatabase *pDB)
{
        CDBVariant DV;                  // Virtueller Datentyp für MFC ODBC Klasse
        CRecordset RS(pDB);                     // Ergebnistyp  der ODBC Klasse (Abfrage
Ergebnis)

        char qry[256];                          // SQL Abfrage String
        sprintf(qry,"SELECT * FROM messdaten where seriennummer='%s'",T->GetData());
        int len = 0; // Anzahl der Stützstellen
        int sz =0;

        if (RS.Open(2,qry,0) && !RS.IsBOF() && !RS.IsEOF())
/*
-mit pRS.Open wird die SQL anfrage an pRS und damit an die DB übergeben
-pRS wird geöffnet
-RS.IsBOF() damit wird geprüft ob die Aktuelle Position des Ergebniszeigers vor dem ersten
Ergebnis steht (Beginn of File)
-RS.IsBOF() damit wird geprüft ob die Aktuelle Position des Ergebniszeigers hinter dem
ersten Ergebnis steht (End of File)
!pRS.IsBOF() && !pRS.IsEOF() hiermit wird überprüft ob der Ergebniszeiger nicht vor und
nach den Ergebnissen gleichzeitig steht, dieser Fall das ist nur möglich wenn ein Ergebnis
der Länge 0 zurückkam
*/
```

```
        {

                char   *pV=0;                   // Aktueller Blob
                double* pD=0;                   // Aktuelle Stützstellen aus Blob


                        printf("Tabelle 'messdaten' geöffnet\n");
                        // kopiert das Feld mit dem Index 1 aus dem aktuellen
Abfrageergebnis in die Variable pDV

                        RS.GetFieldValue(1,&DV);
                        sz=DV.GetBinarySize();        // erechnet beim ersten Durchlauf die
Länge des Blobs
                        len=(sz-4)/8;                                   // Anzahl der Messungen
die sich im Blob befinden
                        pV=(char*)malloc(sz);
                        pD=(double*)malloc(len*8);
                        //Hole Kurvendaten von DB in ein Array
                        DV.GetBinaryData(pV,sz);              // das Ergebnis liegt
anschließend in pV

                        // hier werden alle Kurvendaten (Messdaten aus dem BLOB)
durchgelaufen
                        for (int i=0;i<len;i++)
                        {
                                        // aktueller Datenwert Niederwertigstes Byte und
Höherwertiges Byte werden vertauscht,
                                        // da diese im Blob in "falscher" Reinfolge in der DB
abgespeichert werden

                                pD[i]=LVDouble(pV+8*i+4);
                        }

                paintL(pD,len,255,0,0,T->GetData()); // Zeichne den Mittelwert

                if (pD) free(pD);
                if (pV) free(pV);
        RS.Close();
        }
        else
                printf("Tabelle 'messdaten' nicht geöffnet\n");


        return len;
}
```

1.4.6.3.6 countM

Diese Funktion berechnet die Anzahl der Messungen (BLOBs).
Mittels einer SQL Abfrage wird das DBMS (Datenbank Management System) aufgefordert die Anzahl der Datensätze zu zählen. Auf den Rückgabe Wert kann mittels `m_lval` direkt zugegriffen werden, da das DBMS nur einen *int* wert zurück gibt.

**Parameter**

| | |
|---|---|
| `CText *T` | Text Abfrage welche die Seriennummer der ausgewählten Kurven enthält |
| `CDatabase *pDB` | Datenquelle |

Rückgabewert

| | |
|---|---|
| `int` | Anzahl der Blobs |

```
int countM(CText *T,CDatabase *pDB){  // diese Funktion zählt die Anzahl der Messungen

        CDBVariant DV;                  // Virtueller Datentyp für MFC ODBC Klasse
        CRecordset RS(pDB);                     // Ergebnistyp  der ODBC Klasse (Abfrage
Ergebnis)
        int ret = 0;                            // Rückgabewert der Methode
        char qry[256];                  // SQL String der Abfrage
        sprintf(qry,"SELECT Count(messdaten.seriennummer) "
```

```
                                "FROM messdaten "
                                "WHERE seriennummer like '%s'",
                                T->GetData());
        if(RS.Open(2,qry,0))          //mit pRS->Open wird die SQL anfrage an pRS und damit
an die DB übergeben
        {
                RS.GetFieldValue(0,&DV);      // Hole das Feld mit index 0 aus dem
Abfrageergebnis heraus
                printf("Es gibt: %i Messungen\n",DV.m_lVal);
                ret=DV.m_lVal;                // auf den Primitiv type Int kann mit m:lVal
direkt zugegriffen werden
                RS.Close();
        }
        else printf("Fehler konnte DB in Funktion countM() nicht öffnen");


        return ret;
        }
```

#### 1.4.6.3.7 infoM

Die Funktion liest alle Informationen einer Messung aus einer ODBC Datenquelle und gib sie auf dem Bildschirm aus.

Anmerkung:
Auf die Daten kann mittels `GetFieldValue` direkt zugegriffen werden da es sich um Text Daten handelt.

**Parameter**

| | |
|---|---|
| `CText *T` | Text Abfrage welche die Seriennummer der zu zeichnen Kurve enthält |
| `CDatabase *pDB` | Datenquelle |

```
void infoM(CText *T,CDatabase *pDB){

        CDBVariant DV;                          // Virtueller Datentyp für MFC ODBC Klasse
        CRecordset RS(pDB);                             // Ergebnistyp  der ODBC Klasse
(Abfrage Ergebnis)
        int ret = 0;                                    // Rückgabewert der Methode
        char text[256];                                         // Ausgabe Text
        char hersteller[256],pruefer[255],mtime[255],platz[255]; //ergebnisse aus DB
        sprintf(hersteller,"keine Angabe");
        sprintf(pruefer,"keine Angabe");
        sprintf(mtime,"keine Angabe");
        sprintf(platz,"keine Angabe");



        char qry[256];                                  // SQL String der Abfrage
        sprintf(qry,"SELECT hersteller,pruefer,time,platz "
                            "FROM header "
                            "WHERE seriennummer ='%s'",
                            T->GetData());
        if(RS.Open(2,qry,0)) //mit pRS->Open wird die SQL anfrage an pRS und damit an die
DB übergeben
        {
                // Hole das Feld mit index 0 aus dem Abfrageergebnis heraus
                RS.GetFieldValue(0,hersteller,sizeof(hersteller));
                RS.GetFieldValue(1,pruefer,sizeof(pruefer)); // Hole das Feld mit index 1
aus dem Abfrageergebnis heraus
                RS.GetFieldValue(2,mtime,sizeof(mtime));     // Hole das Feld mit index 2
aus dem Abfrageergebnis heraus
                RS.GetFieldValue(3,platz,sizeof(platz));     // Hole das Feld mit index 3
aus dem Abfrageergebnis heraus
                char out[1000];
                sprintf(out,"Datenblatt :\n"
                                                "Ser. Nr.    :%s\n"
                                                "Hersteller  :%s\n"
                                                "Mitarbeiter:%s\n"
                                                "Pr. Platz   :%s\n"
                                                "Pr. Zeit    :%s\n",T-
>GetData(),hersteller,pruefer,platz,mtime);
                info.SetText(out);


                RS.Close();
        }
        else printf("Fehler konnte DB in Funktion countM() nicht öffnen");
}
```

#### 1.4.6.3.8 p90

Diese Funktion zeichnet die 90% und die 10% Perzentiele(Bereich in dem 90% bzw. 10% aller Messungen liegen) aller Messungen in das Diagramm ein.

**Parameter**

| | |
|---|---|
| `CText *T` | Text Abfrage welche die Seriennummer der ausgewählten Kurven enthält |
| `CDatabase *pDB` | Datenquelle |
| `int len` | Anzahl der Messwerte pro BLOB |

```
void p90(CText *T,CDatabase *pDB,int len)
{
	// diese Funktion zeichnet das 90% Perzentiel
	CDBVariant DV; // Virtueller Datentyp der ODBC Klasse
	CRecordset RS(pDB);    // Ergebnistyp  der ODBC Klasse (Abfrage Ergebnis)
	char qry[256];         // SQL String
	sprintf(qry,"SELECT * "
				"FROM messdaten "
				"WHERE seriennummer like '%s'",
				T->GetData());

	if (RS.Open(2,qry,0) &&
		!RS.IsBOF() &&
		!RS.IsEOF())
	{

		int num= -1; // Laufvariable der Messungen
		int anzahl=countM(T,pDB);     // Anzahl der Messungen
		double** d =(double**)malloc(len*sizeof(double*));//2 Dimensionales Array
zum sichern der Messdaten
		for (int i=0;i<len;i++)
		  d[i]=(double*)malloc(anzahl*sizeof(double));
		int sz = len*8+4;                    // Länge des Blobs
		char* pV=0;


		while(!RS.IsEOF()) // diese Schleife läuft alle Messungen in der DB durch
		{

			num ++;
			// hole die Daten aus dem Feld mit Index 1 und Kopiere Sie  in die
Variable pDV
			RS.GetFieldValue(1,&DV);

			if ((sz==DV.GetBinarySize()))  // fals ein Blob mit falscher Größe
kommt, wird dieser ausgelassen
			{
			// hole Kurvendaten in neues Array d

				pV=(char*)malloc(sz);
				DV.GetBinaryData(pV,sz);
					for (int i=0;i<len;i++){
						d[i][num]=LVDouble(pV+8*i+4); //aktueller
Datensatz
					}
				if (pV) free(pV);

			}
			else printf("Datenbank Zugriffsfehler in funktin p90");
			if (pV) free(pV);
			//pDV = aktueller Datensatz
			RS.MoveNext();

		}
		if (pV) free(pV);



		CVector V;
		double p10,p90;
		double *untereGrenze=(double*)malloc(len*8);  // Graph für die 10 % P.
		double *obereGrenze=(double*)malloc(len*8);    // Graph für die 90 % P.
		for (i=0;i<len;i++)  // Laufe alle Stützpunkte durch
		{
			V.SetData(anzahl,d[i]);
```

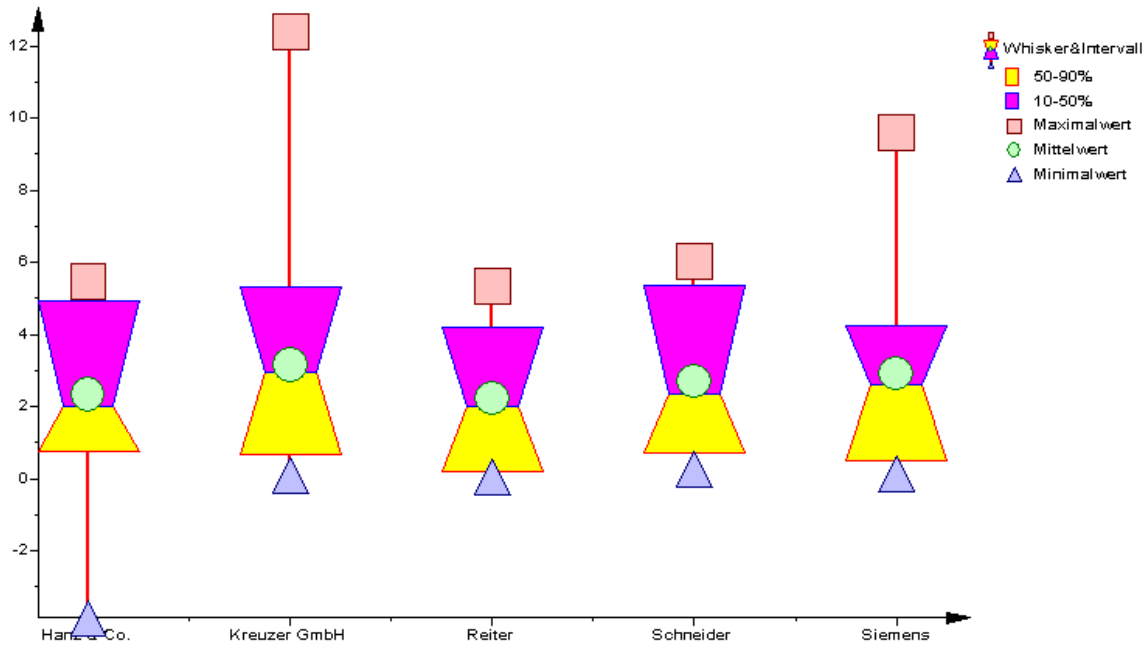
```
                V.Median(&p10,10.0,&p90,90.0,0L);
                untereGrenze[i]=p10;
                obereGrenze[i]=p90;
        }
        paintL(untereGrenze,len,255,255,0,"10% Perzentiel");
        paintL(obereGrenze,len,0,170,0,"90% Perzentiel");

        // aufräumen
        if (untereGrenze)free(untereGrenze);
        if (obereGrenze) free(obereGrenze);

        for (i=0;i<len;i++)
        {
                if (d[i]!=0)
                {
                        free(d[i]);
                }
        }
        if (d)
                free(d);
        RS.Close();
}
else printf("Fehler konnte DB in Funktion p90() nicht öffnen\n");

return;
}
```

1.4.6.3.9 OnRun

Hierbei handelt es sich um den Einsprungpunkt für das X1.

```
int method::OnRun()
{
        printf("START\n");

        PUB.Clear();           // Diagramm leeren

        CDatabase DB;          // Datenbank Objekt

        if (DB.Open("BLOB2",0,1,"ODBC;",1))   // Datenbank öffnen
                {
                printf("Datenbank 'BLOB2' geöffnet\n");

                // Textdialog am Programmanfang
                CText T;
                T.SetData("x-%");
                T.Edit("Bitte geben Sie die Suchabfrage ein");

                int len=stdKurven(&T,&DB);  // Zeichnet den Mittelwert, und das Band der
Messungen
                p90(&T,&DB,len);                        // zeichne 90% Perzentiele

                // Textdialog
                T.SetData("x-10000");
                T.Edit("Bitte geben Sie das gesuchte Bauteil ein");
                // Aswertung der einzelnen Seriennummer
                oneKurven(&T,&DB);
                infoM(&T,&DB);
                DB.Close();

                }


        GetDocument()->Invalidate();
        return 1;
}
```

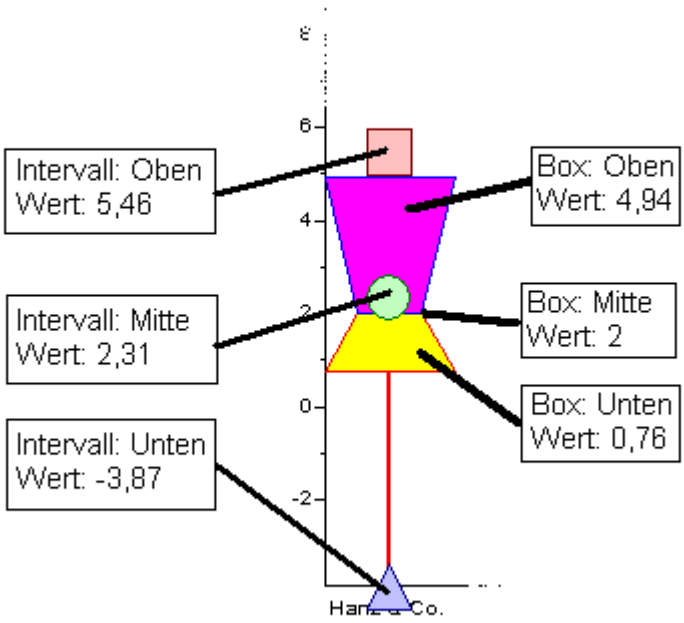
## 1.4.7 whisk



### 1.4.7.1 Kurzbeschreibung

Dieses Beispiel liest aus einer ODBC Datenquelle Messdaten und Hersteller Informationen aus und gibt diese in Form von Whiskers aus.

Ein Whisker hat 6 Anzeigewerte: *Intervall: Oben, Intervall: Mitte, Intervall: Unten, Box: Oben, Box: Unten, Box: Mitte*. Jeder dieser Werte kann mit einen beliebigen Wert belegt werden (wobei *Box: Mitte* zwischen *Box: Unten* und *Box: Oben* liegen muss). Die Werte der jeweiligen Anzeigewerte können auf der Y Achse Abgelesen werden.




### 1.4.7.2 Bedienungsanleitung

Sie starten dieses Beispiel mit der Run- Taste.

Zur Interpretation des Ergebnisses lesen Sie bitte die Kurzbeschreibung.

### 1.4.7.3 Skriptbeschreibung

Um das Skript zu bearbeiten müßen Sie sich in der Betriebsart Bearbeiten befinden. Drücken Sie nun mit der Rechten Maustaste auf **ROOTO** und wählen den Menue Punkt Skript an. Der Einsprungpunkt für das X1 ist die Methode OnRun.


#### 1.4.7.3.1 countM

Diese Funktion berechnet die Anzahl der Messungen.
Mittels einer SQL Abfrage wird das DBMS (Datenbank Management System) aufgefordert die Anzahl der Datensätze zu zählen. Auf den Rückgabe Wert kann mittels `m_lVal` direkt zugegriffen werden, da das DBMS nur einen *int* wert zurück gibt.

**Parameter**

| | |
|---|---|
| `char* hersteller` | char Array mit dem Herstellernamen, desen Messungen gezählt werden sollen |
| `CDatabase *pDB` | Datenquelle |

Rückgabewert

| | |
|---|---|
| `int` | Anzahl der Messungen |

```
int countM(CDatabase *pDB,char* hersteller)
{// diese Funktion zählt die Anzahl der Messungen
        CDBVariant DV;                          // Virtueller Datentyp für MFC ODBC Klasse
        CRecordset RS(pDB);                             // Ergebnistyp  der ODBC Klasse
(Abfrage Ergebnis)
        int ret = 0;                                    // Rückgabewert der Methode
        char qry[256];                                  // SQL String der Abfrage
        sprintf(qry,"SELECT Count(header.seriennummer) "
                                "FROM header where hersteller='%s'",hersteller);
        if(RS.Open(2,qry,0)) //mit pRS->Open wird die SQL anfrage an pRS und damit an die
DB übergeben
        {
                RS.GetFieldValue(0,&DV);// Hole das Feld mit index 0 aus dem
Abfrageergebnis heraus
                //printf("Es gibt: %i Messungen\n",pDV->m_lVal);
                ret=DV.m_lVal; // auf den Primitiv type Int kann mit m:lVal direkt
zugegriffen werden
                RS.Close();
        }
        else printf("Fehler konnte DB in Funktion countM() nicht öffnen");

        return ret;
}
```

#### 1.4.7.3.2 countH

Diese Funktion berechnet die Anzahl der Hersteller.
Mittels einer SQL Abfrage wird das DBMS (Datenbank Management System) aufgefordert die Anzahl der Hersteller zu zählen. Auf den Rückgabe Wert kann mittels `m_lVal` direkt zugegriffen werden, da das DBMS nur einen *int* wert zurück gibt.

**Parameter**

| | |
|---|---|
| `CDatabase *pDB` | Datenquelle |

Rückgabewert

| | |
|---|---|
| `int` | Anzahl der Hersteller |

```
int countH(CDatabase *pDB)
{// diese Funktion zählt die Anzahl der Hersteller
        CRecordset RS(pDB);    // Ergebnistyp  der ODBC Klasse (Abfrage Ergebnis)
        int ret = 0;   // Rückgabewert der Methode
        char qry[256];// SQL String der Abfrage
        sprintf(qry,"SELECT header.hersteller from header group by hersteller");
        if(RS.Open(2,qry,0))   //mit pRS->Open wird die SQL anfrage an pRS und damit an die
DB übergeben
        {

                while(!RS.IsEOF()){ // diese Schleife läuft alle Hersteller in der DB durch
                                ret++;
                                RS.MoveNext();
                }

                RS.Close();
        }
        else printf("Fehler konnte DB in Funktion countM() nicht öffnen");

        return ret;
}
```

### 1.4.7.3.3 getMData

Diese Methode liest Produktionsdaten aus einer ODBC Datenquelleund errechnet den Mittel-, Minimal-, Maximalwert, 10%- , 50%-, 90% Perzentiele aller Einschaltmimente eines Herstellers.

**Parameter**

| | |
|---|---|
| `char* hersteller` | Name der Herstellers für den die Werte berechnet werden sollen |
| `CDatabase *pDB` | ODBC Datenquelle |
| `double* ret` | Rückgabewert der Methode: in diesem Double Array werden die Rückgabewerte gesp |

```
void getMData(char* hersteller,CDatabase *pDB,double* ret)
{
        char qry2[256];                                         // SQL String der Abfrage
        CDBVariant DV; // Virtueller Datentyp für MFC ODBC Klasse
        CRecordset RS(pDB);                             // Ergebnistyp  der ODBC Klasse
(Abfrage Ergebnis)
        double min=0,max=0,mittel;
        int mAnzahl=countM(pDB,hersteller); // Anzahl der Messungen
        double* messungen=(double*)malloc(mAnzahl*sizeof(double)); // hier werden später
alle Einschaltmomente gespeichert
        int run2=0;
        double wert; // aktuelles Messergebnis

        sprintf(qry2,"SELECT schaltwerte.einschaltpunkt FROM header INNER JOIN schaltwerte
ON header.seriennummer =                    schaltwerte.seriennummer WHERE
(header.hersteller)='%s'",hersteller);

        if(RS.Open(2,qry2,0)) //mit pRS->Open wird die SQL anfrage an pRS und damit an die
DB übergeben
        {
                while(!RS.IsEOF())
                { // diese Schleife läuft alle Schwellwerte des Herstellers in der DB
durch
                        RS.GetFieldValue(0,&DV);      // Hole das Feld mit index 0 aus dem
Abfrageergebnis heraus
                        wert= DV.m_dblVal;
```

```
                        // hier werden min und maxwerte ermittelt
                        if (run2==0)
                        {
                                min=wert;
                                max=wert;
                                mittel=wert;
                        }
                        else
                        {
                                mittel+=wert;
                                if (min>wert) min = wert;
                                if (max<wert) max = wert;
                        }

                        messungen[run2]=wert;

                        RS.MoveNext();
                        run2++;
                }
        }
        RS.Close();
        // hier werden die Maeesungen ausgewertet und in das Returnarray gespeichert
        CVector V;
        double p10=0,p90=0,p50=0;
        V.SetData(mAnzahl,messungen);
        V.Median(&p10,10.0,&p90,90.0,&p50,50.0,0L);
        ret[0]= min;
        ret[1]= max;
        ret[2]= p10;
        ret[3]= p50;
        ret[4]= p90;
        ret[5]= mittel/run2;
        printf("Min:%g Max: %g P10:%g P50:%g P90:%g\n",ret[0],ret[1],ret[2],ret[3],ret[4]);

        free(messungen);


}
```

#### 1.4.7.3.4 OnRun

Hierbei handelt es sich um den Einsprungpunkt für das X1.

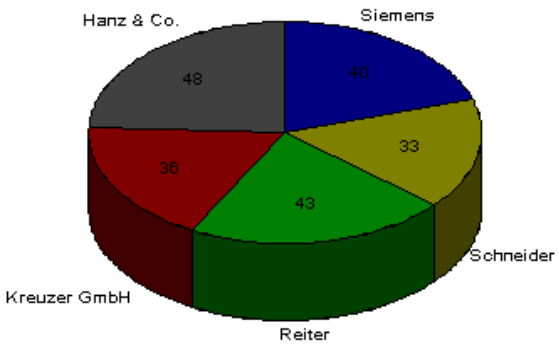
```
int method::OnRun()
{
	CDatabase DB;		// Datenbank Objekt
	double messung[6];

	if (DB.Open("BLOB2",0,1,"ODBC;",1))   // Datenbank öffnen
	{
		AX.EmptyTicks();				// lösche die Beschriftung der
X Achse

		int hAnzahl=countH(&DB);  // ermittle die Anzahl der Hersteller
		char hersteller[255];			// jeweils aktueller Hersteller
		int run1=0;// Laufndex für 1. while Schleife
		// Arrays zum Zeichnen der Whisker
		double* max=(double*)malloc(hAnzahl*sizeof(double));
		double* min=(double*)malloc(hAnzahl*sizeof(double));
		double* p10=(double*)malloc(hAnzahl*sizeof(double));
		double* p90=(double*)malloc(hAnzahl*sizeof(double));
		double* p50=(double*)malloc(hAnzahl*sizeof(double));
		double* mittel=(double*)malloc(hAnzahl*sizeof(double));

		// SQL Abfrage
		char qry[256];
		sprintf(qry,"SELECT header.hersteller from header group by hersteller");

		CRecordset RS(&DB);
		if(RS.Open(2,qry,0)) //mit pRS->Open wird die SQL anfrage an pRS und damit
an die DB übergeben
		{
			while(!RS.IsEOF())
			{  //diese Schleife läuft alle Hersteller in der DB durch
				RS.GetFieldValue(0,hersteller,sizeof(hersteller)); // hier
wird der Hersteller ausgelesen

				// Debug Ausgabe
				//printf("Hersteller: %s\n",hersteller);
				//printf("Messungen: %i\n",countM(&DB,hersteller));

				// errechne die Herstellerspezifischen Daten aus den
Messungen
				if (countM(&DB,hersteller)>0)
				{
					getMData(hersteller,&DB,messung);
					max[run1]=messung[1];
					min[run1]=messung[0];
					p10[run1]=messung[2];
					p50[run1]=messung[3];
					p90[run1]=messung[4];

					mittel[run1]=messung[5];
					AX.AddTick(hersteller,(run1*10.0)+10.0);
				}
				else
				{
					max[run1]=0;
					min[run1]=0;
					p10[run1]=0;
					p50[run1]=0;
					p90[run1]=0;
					mittel[run1]=0;
					AX.AddTick("Fehler in DB",(run1*10.0)+10.0);
				}

				RS.MoveNext();// springe zum nächsten Datensatz(hersteller)
				run1++;
			}
			RS.Close();
		}
		else
		  printf("Fehler konnte DB nicht öffnen");
```

```
                DB.Close();

                //Fügt die Whiskers in das Diagram ein
                CWhiskTrace* pWT=new CWhiskTrace();
                pWT->SetTraceAtt(1230,4);
                pWT->SetTraceAtt(1231,600);
                pWT->SetTraceAtt(3230,5.0);
                pWT->SetTraceAtt(3231,0.25);
                pWT->SetTraceAtt(4154,"Whisker&Intervall");
                //Diagrammbezug
                pWT->SetTraceAtt(4150,"AX");
                // X-Achse
                pWT->SetTraceAtt(4151,"AY");
                // Y-Achse
                pWT->SetTraceAtt(4230,"Maximalwert");
                pWT->SetTraceAtt(4231,"Mittelwert");
                pWT->SetTraceAtt(4232,"Minimalwert");
                pWT->SetTraceAtt(4233,"50-90%");
                pWT->SetTraceAtt(4234,"10-50%");
                //Daten übergeben
                pWT->SetTraceAtt(10030,10.0,10.0,5L);
                pWT->SetTraceAtt(10031,min,5L);
                pWT->SetTraceAtt(10032,mittel,5L);
                pWT->SetTraceAtt(10033,max,5L);
                pWT->SetTraceAtt(10034,p90,5L);
                pWT->SetTraceAtt(10035,p50,5L);
                pWT->SetTraceAtt(10036,p10,5L);
                //Layout von Intervall und Box
                pWT->SetTraceAtt(6234,1,0,255,0,0);
                pWT->SetTraceAtt(7234,0,0,255,255,0);
                pWT->SetTraceAtt(8234,255,255,255);
                pWT->SetTraceAtt(6235,1,0,0,0,255);
                pWT->SetTraceAtt(7235,0,1,255,0,255);
                pWT->SetTraceAtt(8235,255,255,255);
                pWT->SetTraceAtt(9230,"X1-Sym",39);
                pWT->SetTraceAtt(6230,1,0,128,0,0);
                pWT->SetTraceAtt(7230,0,0,255,192,192);
                pWT->SetTraceAtt(9231,"X1-Sym",44);
                pWT->SetTraceAtt(6231,1,0,0,128,0);
                pWT->SetTraceAtt(7231,0,0,192,255,192);
                pWT->SetTraceAtt(9232,"X1-Sym",38);
                pWT->SetTraceAtt(6232,1,0,0,0,128);
                pWT->SetTraceAtt(7232,0,0,192,192,255);
                pWT->SetTraceAtt(6233,1,50,255,0,0);
                //Kurve an Legende übergeben
                LEG.Clear();
                LEG.AddTrace(pWT);

                GetDocument()->Invalidate();

                free(min);
                free(max);
                free(p90);
                free(p10);
                free(p50);
        }
        return 1;
}
```

## 1.4.8 torte-DB

XOn Beispiel,
Dieses einfache Beispiel zeigt wie ein Tortendiagram über ein Skript angesteuert werden kann. Die Daten werden aus einer ODBC Datenquelle entnommen.
Das Skript befindet sich in ROOT0

Anzahl der verbauten Teile, aufgeschlüßelt nach Hersteller.




### 1.4.8.1 Kurzbeschreibung

Dieses Beispiel liest aus einer ODBC Datenquelle Hersteller-Informationen aus. Es wird errechnet welchen Anteile welcher Hersteller an den gelieferten Bauteile hat. Das Ergebnis wird als Torten-Diagramm ausgegeben.

### 1.4.8.2 Bedienungsanleitung

Sie starten dieses Beispiel mit der Run- Taste.

### 1.4.8.3 Skriptbeschreibung

Um das Skript zu bearbeiten müßen Sie sich in der Betriebsart Bearbeiten befinden. Drücken Sie nun mit der Rechten Maustaste auf **ROOTO** und wählen den Menue Punkt Skript an. Der Einsprungpunkt für das X1 ist die Methode OnRun.

1.4.8.3.1 countM

Diese Funktion berechnet die Anzahl der Messungen.
Mittels einer SQL Abfrage wird das DBMS (Datenbank Management System) aufgefordert die Anzahl der Datensätze zu zählen. Auf den Rückgabe Wert kann mittels `m_lVal` direkt zugegriffen werden, da das DBMS nur einen *int* wert zurück gibt.

**Parameter**

| | |
|---|---|
| `char* hersteller` | char Array mit dem Herstellernamen, desen Messungen gezählt werden sollen |
| `CDatabase *pDB` | Datenquelle |

Rückgabewert

| | |
|---|---|
| `int` | Anzahl der Messungen |

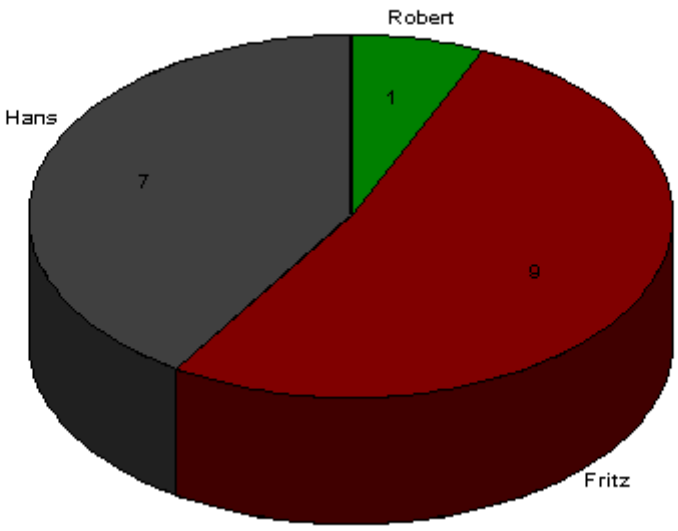
```
int countM(CDatabase *pDB,char *hersteller)
{// diese Funktion zählt die Anzahl der Messungen

        CDBVariant DV; // Virtueller Datentyp für MFC ODBC Klasse
        CRecordset RS(pDB);    // Ergebnistyp  der ODBC Klasse (Abfrage Ergebnis)

        int ret = 0;           // Rückgabewert der Methode
        char qry[256]; // SQL String der Abfrage

        sprintf(qry,"SELECT Count(header.seriennummer) "
                            "FROM header where hersteller='%s'",hersteller);

        if(RS.Open(2,qry,0)) //mit pRS->Open wird die SQL anfrage an pRS und damit an die
DB übergeben
        {
                RS.GetFieldValue(0,&DV);      // Hole das Feld mit index 0 aus dem
Abfrageergebnis heraus
                //printf("Es gibt: %i Messungen\n",pDV->m_lVal);
                ret=DV.m_lVal;// auf den Primitiv type Int kann mit m:lVal direkt
zugegriffen werden

                RS.Close();
        }
        else printf("Fehler konnte DB in Funktion countM() nicht öffnen");


        return ret;
}
```

#### 1.4.8.3.2 OnRun

Hierbei handelt es sich um den Einsprungpunkt für das X1.

```
int method::OnRun()
{

        torte.Clear();// löscht alle Tortenstücke


        CDatabase DB;          // Datenbank Objekt
        char hersteller[512];

        if (DB.Open("BLOB2",0,1,"ODBC;",1))   // Datenbank öffnen
        {
                CRecordset RS(&DB);

                // SQL Abfrage
                char qry[256];
                sprintf(qry,"SELECT header.hersteller from header group by hersteller");
                if(RS.Open(2,qry,0)) //mit pRS->Open wird die SQL anfrage an pRS und damit
an die DB übergeben
                {
                        for(int i=0;!RS.IsEOF();i++)
                        {  //diese Schleife läuft alle Hersteller in der DB durch
                                // hier wird der Hersteller ausgelesen
                                RS.GetFieldValue(0,hersteller,sizeof(hersteller));

                                // Debug Ausgabe
                                printf("Hersteller: %s\n",hersteller);
                                printf("Messungen: %i\n",countM(&DB,hersteller));


                                // hier wird ein neues Tortenstück eingefügt

                                // fügt neues Tortenstück mit der Größe 7,1 ein
                                torte.AddPie((double)countM(&DB,hersteller),-1);  /
                                torte.SetNodeAtt(1105,i);  // legt Tortenstück i als
aktuelles Tortenstück fest

                                //benennt das Tortenstück i mit dem Inhalt der var.
Hersteller
                                torte.SetNodeAtt(4150,hersteller);

                                RS.MoveNext();
                        }
                RS.Close();
                }
        DB.Close();
        }


        GetDocument()->Invalidate();// Zeichnet den Bildschirm neu
  printf("END\n");
        return 1;
}
```

#### 1.4.8.4 Das Tortendiagramm




Dieses Tortendiagramm wurde über das nachfolgende Skript angesteuert. Die dazugehörige Beispieldatei heißt torte.x1g.

```
int method::OnRun()
{
        while (torte.DelPie()==0);// löscht alle Tortenstücke

        // fügt Tortenstück Hans ein
        torte.AddPie(7.1,-1);  // fügt neues Tortenstück mit der Größe 7,1 ein
        torte.SetNodeAtt(1105,0);  // legt Tortenstück 0 als aktuelles Tortenstück fest
        torte.SetNodeAtt(4150,"Hans"); // fügt den Namen des Tortenstücks ein

        // fügt Tortenstück Hans ein
        torte.AddPie(9.1,-1);  // fügt neues Tortenstück mit der Größe 9,1 ein
        torte.SetNodeAtt(1105,1);  // legt Tortenstück 1 als aktuelles Tortenstück fest
        torte.SetNodeAtt(4150,"Fritz"); // fügt den Namen des Tortenstücks ein

// fügt Tortenstück Hans ein
        torte.AddPie(1.1,-1);  // fügt neues Tortenstück mit der Größe 1,1 ein
        torte.SetNodeAtt(1105,2);  // legt Tortenstück 2 als aktuelles Tortenstück fest
        torte.SetNodeAtt(4150,"Robert"); // fügt den Namen des Tortenstücks ein

        GetDocument()->Invalidate();
  return 1;
}
```

## 1.4.9 Datenbank installieren

Für die Datenbankbeispiele benötigen sie natürlich eine geeignete Datenquelle. Diese Datenquelle (`blob2.mdb`) muß

- auf ihrem Rechner vorhanden sein
- als Datenquelle im ODBC- Manager registriert sein
- ein geeigneter ODBC- Treiber muß auf ihrem Rechner installiert sein

Nachfolgend erhalten sie Informationen über diese Arbeitsschritte.

#### 1.4.9.1 ACCESS- ODBC- Treiber installieren

diese Beispiele gehen davon aus das auf dem Rechner ein ODBC Access Treiber installiert ist. Dies ist normalerweise der Fall wenn MS Access installiert ist. Sollte eine solche Accessunterstützung nicht installiert sein muß dies nachgeholt werden. Dies geht am einfachsten indem Sie den MDAC (Microsoft Data Acsess Components) Treiber installieren.

Dieser kann unter: http://www.microsoft.com/mdac kostenlos herunter geladen werden.

#### 1.4.9.2 ODBC- Datenquelle anlegen

Damit X1 auf die Daten zugreifen kann, muß die Datenquelle (blob2.mdb) im Windows registriert werden. Hierfür ist bei allen Windows Versionen (ab Win95 ) das Programm *odbcconf* vorhanden. Um die Datei blob2.mdb zu registrieren geben sie folgende Zeile in der Eingabeaufforderung ein:

```
odbcconf CONFIGDSN "Microsoft Access Driver (*.mdb)" "DSN=BLOB2|DBQ=E:\blob2.mdb"
```

wobei sie den Pfad *E:* ersetzen müssen. Verwenden Sie statt *E:* den Pfad in welchem die Datei *blob2.mdb* auf ihrem Rechner liegt.

Der X1- Installationssatz für die Demo- und Vollversion führt diese Schritte automatisch aus, wenn sie die Datenbank- Beisiele zur Installation anwählen.

## 1.5 ActiveX

Die Beispiele zeigen, wie man X1 über die ActiveX- Schnittstelle fernsteuert. Die Beispiele gliedern sich nach den Entwicklungsplattformen, die zur Fernsteuerung verwendet werden. Enthalten sind Beispiele mit

- LabVIEW
- CVI/Measurement Studio
- Microsoft Visual Studio

### 1.5.1 LabVIEW

In diesem Kapitel finden sie Beispiele zur Fernsteuerung von X1 mit LabVIEW.

#### 1.5.1.1 Easy

**Verzeichnis**
<X1>/samples/ActiveX/LabVIEW/Easy

**Systemvoraussetzungen**

- X1 ab Version 2.3
- LabVIEW 6.1

**Beschreibung**
Dieses Beispiel demonstriert die grundlegenden Mechanismen zur Fernsteuerung von X1 in LabVIEW:

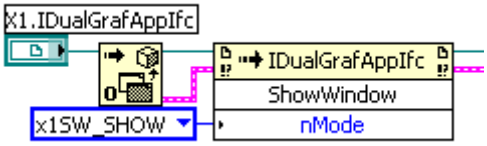
- X1 starten
- Dokument laden
- Dokument aktualisieren
- Dokument drucken
- X1 schließen

Das verwendete X1- Dokument `easy.x1g` besitzt ein Diagramm mit einer Kurve. Die Daten der Kurve (Y-Werte) sind mit einem Vektor vom Typ double im Datenpool verknüpft. Der Name des Datenpoolelementes lautet *Easy*. Das LabVIEW- Programm überträgt ein Array von Zufallszahlen an das Datenpoolelement *Easy* und läßt das Dokument neu zeichnen.

##### 1.5.1.1.1 X1 starten

Zuerst wird ein neues ActiveX-Objekt erzeugt und diesem die ActiveX-Klasse *X1.IDualGrafAppIfc* zugewiesen. Die Klassen können über Rechte Maustaste -> Select ActiveX-Class ->Browse... ausgewählt werden. Dabei ist darauf zu achten, dass es eine "Dual-Class" ist. Alle ActiveX-Nodes finden sie in LabVIEW in der Funktionspalette unter *Communication->ActiveX*.

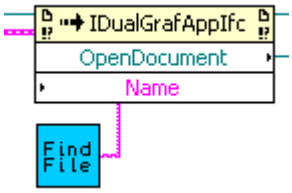
Mit der Invoke Node wird die Methode *ShowWindow* geöffnet (Rechte Maustaste -> Methods). Der Parameter wird auf *x1SW_SHOW* gesetzt. Damit wird das X1-Fenster angezeigt.




Mit diesem Schritt haben wir X1 gestartet und das Fenster angezeigt. Im nächsten Schritt *Dokument laden* wird die Diagrammvorlage geöffnet.

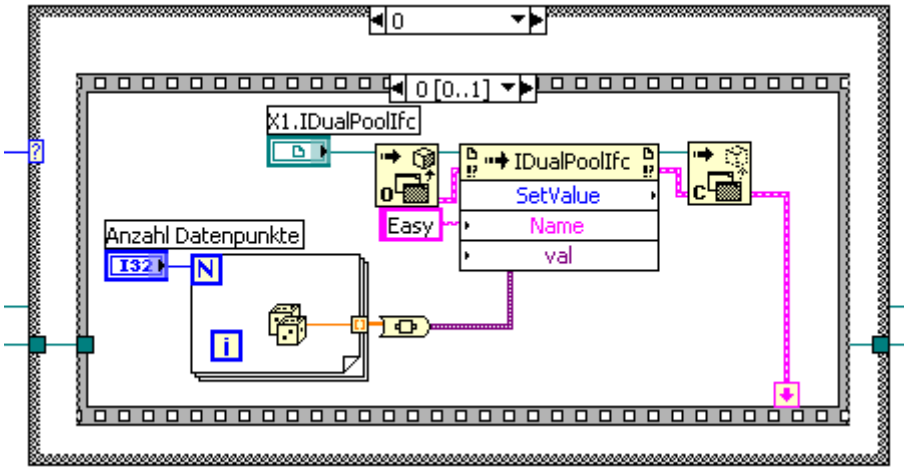
#### 1.5.1.1.2 Dokument laden

Mit der Methode *OpenDocument,* die über eine weitere Invoke Node ausgewählt wird, wird eine gespeicherte X1-Dokumentenvorlage geöffnet. Über den Parameter *Name* wird der Pfad eingegeben. Das SubVI *Find File* dient nur zur automatischen Ermittlung des Pfades in dem sich die Vorlage befindet.




#### 1.5.1.1.3 Dokument aktualisieren

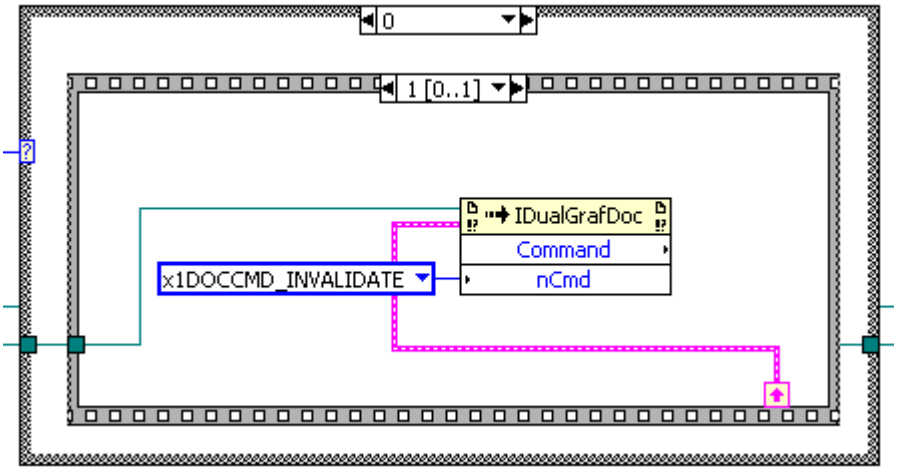
Wird der Button *Neue Daten erzeugen* gedrückt, wird folgender Diagrammteil ausgeführt:




Zuerst wird ein neuer Array mit Zufallszahlen erzeugt. Diese müssen dann in den Daten-Pool von X1 übertragen werden.
Dazu wird eine neue ActiveX-Klasse geöffnet mit dem Namen *X1.DualPoolIfc*.
Danach wird die Methode *SetValue* über eine Invoke Node ausgeführt. Über den Parameter *Name* wird der Name des Daten-Pools angegeben.
Bevor der erzeugte Array an den Parameter *val* übergeben werden kann, muss er in einen Variant-Datentyp konvertiert werden. Die entsprechende Node ist auch im ActiveX Menü der

Funktionspalette von LabVIEW zu finden.
Nach erfolgter Übergabe der Daten wird die Klasse wieder geschlossen.

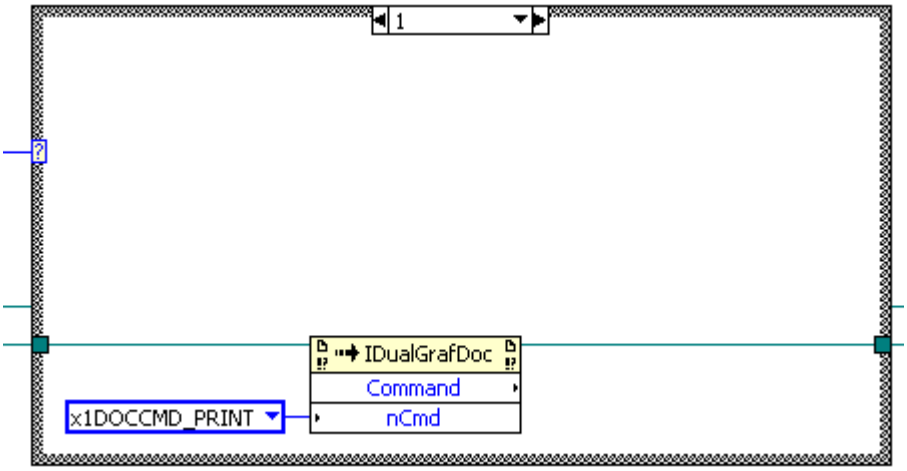
Im nächsten Schritt wird die das Diagramm mit den neu erzeugten Daten aktualisiert.




Dies geschieht mit der Klasse *IDualGrafDoc* über die Methode *Command*. Mit dem Parameter X1DOCCMD_INVALIDATE wird das Dokument aktualisiert.
Als Refnum für die Klasse wird der zuvor erzeugte Refnum beim Öffnen des Dokumentes verwendet.

#### 1.5.1.1.4 Dokument drucken

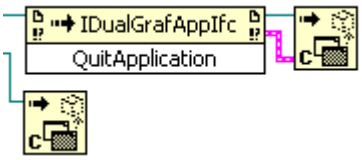
Wird der Button *Diagramm drucken* gedrückt, wird folgender Diagrammteil ausgeführt:




Wir verwenden erneut die Klasse *IDualGrafDoc* mit der Methode *Command*. Mit dem Parameter X1DOCCMD_PRINT wird das Dokument gedruckt.
Als Refnum für die Klasse wird der zuvor erzeugte Refnum beim Öffnen des Dokumentes benutzt.

1.5.1.1.5 X1 beenden

Wird der Button *Exit* gedrückt, soll X1 wieder geschlossen werden.




Dies geschieht mit der Klasse *IDualGrafAppIfc*, die wir zuvor schon zum Öffnen von X1 verwendet haben.
Über die Methode *QuitApplication* wird x1 beendet. Der Refnum muss logischerweise der gleiche wie beim Öffnen von X1 sein.
Danach werden die noch geöffneten ActiveX-Klassen geschlossen.

## 1.5.2 CVI/Measurement Studio

In diesem Kapitel finden sie Beispiele zur Fernsteuerung von X1 mit CVI/Measurement Studio.

### 1.5.2.1 Easy

**Verzeichnis**
<X1>/samples/ActiveX/CVI/Easy

**Systemvoraussetzungen**
- X1 ab Version 2.3
- Measurement Studio 6.0

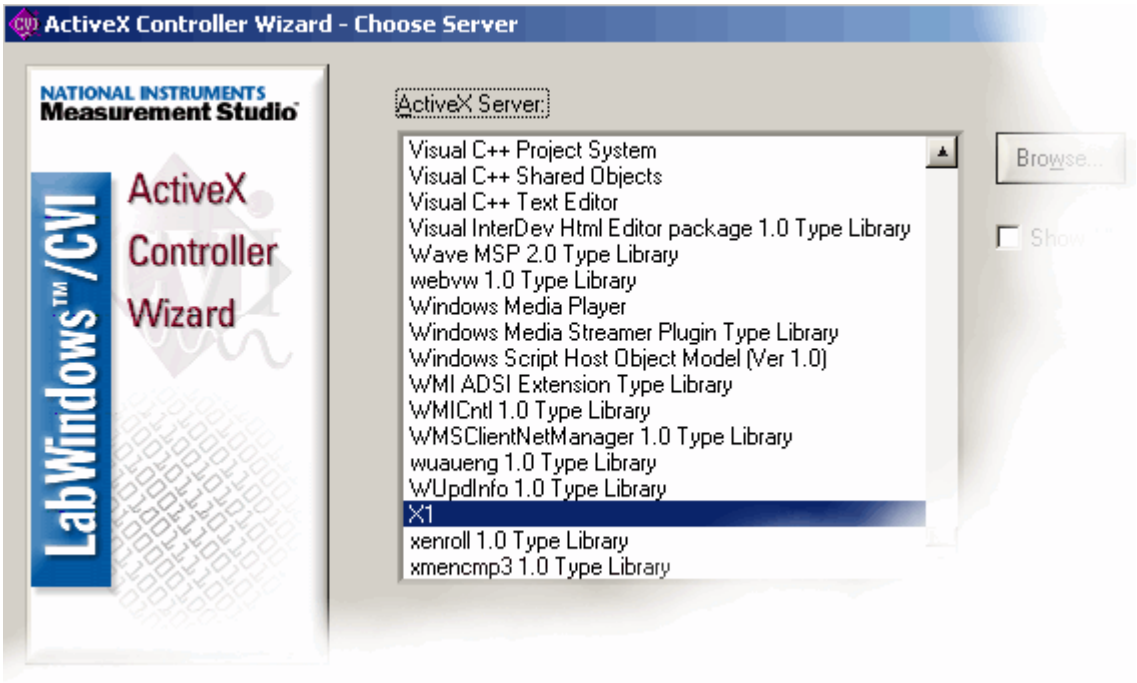
**Beschreibung**
Dieses Beispiel demonstriert die grundlegenden Mechanismen zur Fernsteuerung von X1 in CVI:

- ActiveX Controller erzeugen
- X1 starten
- Dokument laden
- Dokument aktualisieren
- Dokument drucken
- X1 schließen

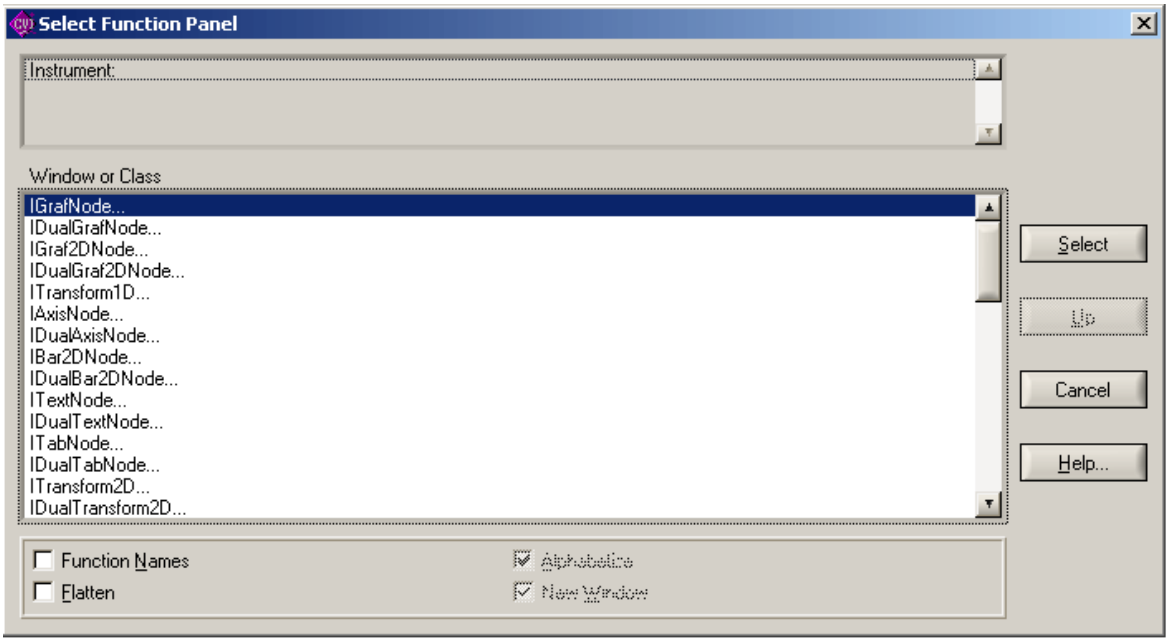
Das verwendete X1- Dokument `easy.x1g` besitzt einen Diagramm mit einer Kurve. Die Daten der Kurve (Y-Werte) sind mit einem Vektor vom Typ double im Datenpool verknüpft. Der Name des Datenpoolelementes lautet *Easy*. Das CVI- Programm überträgt ein Array von Zufallszahlen an das Datenpoolelement *Easy* und läßt das Dokument neu zeichnen.

1.5.2.1.1 ActiveX Controller erzeugen

Um X1 von einer CVI- Applikation fernzusteuern, benötigen sie einen ActiveX Controller von X1 in CVI. Diesen erzeugen sie über das Menü *Tools -> Create ActiveX Controller...*
Mit dem ActiveX Controller Wizard wählt man X1 als ActiveX Server aus.




Nach Angabe des Pfades und des Dateinamens wird der ActiveX Controller schliesslich erzeugt. Er ist jetzt unter CVI im Menü *Instrument* aufgeführt.




1.5.2.1.2 X1 starten

Zuerst wird ein neues ActiveX-Objekt erzeugt, dass den Handle *DualGrafAppIfcHandle* der Applikation zurückgibt. Das entsprechende Functionpanel findet man unter *Instrument -> 1... -> IDualGrafAppIfc... -> NewIDualGrafAppIfc*.
Danach wird mit der Funktion *X1_IDualGrafAppIfcShowWindow* das X1-Fenster angezeigt. Die Funktion findet man unter *Instrument -> 1... -> IDualGrafAppIfc... -> Show Window*. Es wird das Handle von dem zuvor erzeugten ActiveX-Objekt verwendet.

```
X1_NewIDualGrafAppIfc (NULL, 1, LOCALE_NEUTRAL, 0, &DualGrafAppIfcHandle);
X1_IDualGrafAppIfcShowWindow (DualGrafAppIfcHandle, NULL, X1Const_x1SW_SHOW);
```

#### 1.5.2.1.3 Dokument laden

Nach dem Starten von X1 muss die Dokumentenvorlage mit den Voreinstellungen für das Diagramm geladen werden. Dies geschieht mit der Funktion *X1_IDualGrafAppIfcOpenDocument*, zu finden unter *Instrument -> 1... -> IDualGrafAppIfc... -> Open Document*. Dabei wird das neue Handle *DualGrafAppIfcDocumentHandle* für das Dokument erzeugt.

```
X1_IDualGrafAppIfcOpenDocument(
        DualGrafAppIfcHandle,
        NULL,
        Path,
        &DualGrafAppIfcDocumentHandle);
```

#### 1.5.2.1.4 Dokument aktualisieren

Durch betätigen des Buttons *Neue Daten erzeugen* wird die Callback *Update* aufgerufen.
Zuerst wird ein Array mit Zufallszahlen erzeugt. Damit dieser von dem Active X-Objekt verarbeitet werden kann muss er in einen Variant umgewandelt werden. Dies erreicht man mit der der Funktion *CA_VariantSet1DArray* (*Library -> ActiveX... -> Variant related Functions... -> Assigning Values to Variants... -> Variant Set 1D Array*).
Der gewandelte Array mit den Zufallszahlen wird danach an den Datenpool von X1 übergeben. Dazu muss erst ein neues ActiveX-Objekt geöffnet werden mit *X1_NewIDualPoolIfc* (*Instrument -> 1... -> IDualPoolIfc... -> NewIDualPoolIfc*). Das zurückgegebene Handle *DualPoolIfcHandle* wird in der nächsten Funktion zum Schreiben des Arrays in den Datenpool, von der Funktion *X1_IDualPoolIfcSetValue* (*Instrument -> 1... -> IDualPoolIfc... -> Set Value*) benötigt.
Danach wird das Active X-Objekt des Daten-Pools wieder geschlossen mit *CA_DiscardObjHandle (Library -> ActiveX... -> Resource Managment... -> Discard Object Handle*).
Um das Diagrammfenster in X1 zu aktualisieren wird die Funktion *X1_IDualGrafDocCommand* (*Instrument -> 1... -> IDualGrafIDoc... -> Command*) mit dem Handle der Dokumentenvorlage *DualGrafAppIfcDocumentHandle* ausgeführt.

```
int i;
double daten[50];
CAObjHandle DualPoolIfcHandle;
VARIANT VDaten;

//Daten erzeugen
for (i=0; i<=50; i++)
  daten[i]=1.0e-4*rand();

//Array an Variant übergeben
CA_VariantSet1DArray (&VDaten, CAVT_DOUBLE, 50, daten);
//Datenpool öffnen
X1_NewIDualPoolIfc(NULL, 1, LOCALE_NEUTRAL, 0, &DualPoolIfcHandle);
//Pooldaten übergeben
X1_IDualPoolIfcSetValue (DualPoolIfcHandle, NULL, "Easy", VDaten, &ret);
//Poolobjekt freigeben
CA_DiscardObjHandle (DualPoolIfcHandle);
//Report neu zeichnen
X1_IDualGrafDocCommand (DualGrafAppIfcDocumentHandle, NULL,
                        X1Const_x1DOCCMD_INVALIDATE, &ret);
```

#### 1.5.2.1.5 Dokument drucken

Durch betätigen des Buttons *Diagramm drucken* wird die Callback *Print* aufgerufen.
Mit der Funktion *X1_IDualGrafDocCommand* (*Instrument -> 1... -> IDualGrafIDoc... -> Command*), dem Handle der Dokumentenvorlage *DualGrafAppIfcDocumentHandle* und dem Parameter zum drucken wird das geöffnete Dokument mit dem Diagramm ausgedruckt.

```
X1_IDualGrafDocCommand (DualGrafAppIfcDocumentHandle, NULL, X1Const_x1DOCCMD_PRINT, &ret);
```

#### 1.5.2.1.6 X1 beenden

Durch Drücken des Buttons *Beenden* wird mit der Funktion *X1_IDualGrafAppIfcQuitApplication* X1 beendet (*Instrument -> 1... -> IDualGrafAppIfc... -> Quit Application)*.
Danach müssen noch die bestehenden Object Handles mit der Funktion *CA_DiscardObjHandle* geschlossen werden (*Library -> ActiveX... -> Resource Managment... -> Discard Object Handle*).

```
//X1 Beenden
```

```
X1_IDualGrafAppIfcQuitApplication (DualGrafAppIfcHandle, NULL);
//Objekthandle freigeben
CA_DiscardObjHandle (DualGrafAppIfcDocumentHandle);
CA_DiscardObjHandle (DualGrafAppIfcHandle);
```

## 1.6 Werkzeugleiste

| Symbol | Shortcut | Bezeichnung | Beschreibung |
|---|---|---|---|
|  | <Ctrl> + <A> | Runtaste | Skript eines Reports ausführen |
|  | <Ctrl> + <B> | Betriebsart Bearbeiten | Wechselt in den *Bearbeiten- Modus* bzw. verlässt ihn wieder. Beim Verlassen des *Bearbeiten- Modus* werden die Skripte Compilie Objekte, die ein Fehlerhaftes Skript besitzen, werden im Projektfenster mit dem Symbol markiert. |

# Index

## - A -



## - B -



## - D -



## - E -



## - G -



## - L -



## - O -



## - W -