

Informatik 2 Programmieren in Delphi: Stringgrids und Menüs

Duale Hochschule
Baden-Württemberg
University of
Cooperative Education

Programmieren in Pascal und Delphi

Theorieteil

Das Stringgrid

Für eine tabellarische Auswertung von Messdaten sehr nützlich ist das Stringgrid. Es ist ein zweidimensionales Gitter, das Strings aufnimmt. Es ähnelt ein bisschen einem Excel Sheet, nur kann man nicht mit ihm rechnen. Diese Komponente finden sie unter der Rubrik „Zusätzlich“.

Die wichtigste Eigenschaft des Stringgrids ist **cells**. Cells nimmt den Inhalt auf der dargestellt wird. **cells** ist ein zweidimensionales Array. Die erste Dimension gibt die Spalte an, die Zweite die Zeile. Beide Dimensionen beginnen bei 0. So ist:

```
Stringgrid1.Cells[0,0] // links oben  
Stringgrid1.Cells[1,0] // zweite Spalte, erste Reihe  
Stringgrid1.Cells[0,1] // erste Spalte, zweite Reihe  
Stringgrid1.Cells[3,7] // vierte Spalte, achte Reihe
```

In jede Zelle kann ein **String** gespeichert werden. Zahlen müssen vorher in Strings umgewandelt werden:

```
Stringgrid1.Cells[3,7]:='Hallo';  
Stringgrid1.Cells[1,2,]:=FloatToStr(98,3);
```

Das Umgekehrte gilt beim Auslesen des Grids:

```
x:=StrToFloat(Stringgrid1.Cells[1,2,]);
```

Strings zu Zahlen konvertieren – Fehler abfangen

Dabei müssen wie bei jeder Stringkonvertierung Fehler abgefangen werden. Leere Strings oder Buchstaben können nicht in Zahlen umgewandelt werden.

Neben dem **try ... except end** Block gibt es dazu auch zwei Routinengruppen:

StrToXXXXdef(<String>, >Vorgabe>) : Typ XXXX

Diese Funktion liefert als Rückgabewert entweder den konvertierten Wert des übergebenen Strings, wenn es keinen Fehler beim Konvertieren gab oder den Vorgabewert, wenn es einen Fehler gab.

TryStrToXXXX(<String>, <Variable vom Typ XXXX>) : boolean;

Diese Funktion hat den Rückgabewert **true**, wenn eine Konvertierung möglich war. Dann enthält die Variable den konvertierten Wert. Wenn dies nicht möglich war, so gibt die Funktion **false** zurück.

In beiden Gruppen steht XXXX für eines der Schlüsselwörter **Date**, **Int** oder **Float** also Datums, Integer und Fließkommavariablen. Eine Variable aus dieser Gruppe muss auch übergeben werden.

Informatik 2 Programmieren in Delphi: Stringgrids und Menüs

Beispiel: folgender Block

```
var x: Double;

Begin
  try
    x:=StrToFloat (Edit1.Text);
  except;
  End;
End;
```

Kann ersetzt werden durch:

```
const Fehler = -1000;

var x: Double;
    Temp: Double;

Begin
  Temp:=StrToIntDef (Edit1.Text, Fehler);
  if Temp<>Fehler then x:=Temp;
end;
```

Oder

```
var x: Double;

Begin
  TryStrToFloat (Edit1.Text, x);
end;
```

Die Benutzung von Stringgrids – Wichtige Eigenschaften

Die Anzahl der Zeilen im Stringgrid kann durch die Eigenschaft **RowCount** bestimmt und verändert werden. Umgekehrt enthält die Eigenschaft **Colcount** die Anzahl der Spalten:

```
Stringgrid1.RowCount:=8; // Die Tabelle hat 8 Zeilen
Stringgrid1.Colcount:=Stringgrid1.Colcount+1; // erweitert es um eine Spalte rechts.
```

Wie bei Count bei Stringlisten ist RowCount immer um 1 höher als der Index der letzten Zeile und Colcount um 1 höher als der Index der letzten Spalte.

Es ist möglich, die ersten Spalte(n) und Zeile(n) dunkler einzufärben und zu fixieren. (Sie scrollen dann nicht bei der Bewegung der Tabelle mit).

Dazu dienen die Eigenschaften

- **FixedColor**: die Farbe der fixierten Spalten und Zeilen
- **FixedRows**: die Anzahl der fixierten Zeilen
- **FixedCols**: die Anzahl der fixierten Spalten

Alle drei Eigenschaften sind vom Typ Integer. Es muss gelten: FixedRows<RowCount und FixedCols < Colcount, sonst erhält man eine Fehlermeldung. Während das Programm

läuft kann die Zahl dynamisch verändert werden.

Optionen

Das Aussehen einer Zelle kann mit **DefaultColWidth**, **DefaultRowHeight** und **Gridlinewidth** verändert werden. Alle drei Eigenschaften sind vom Typ Integer. Sie geben die Höhe, Breite und die Breite des Gitters in Pixels an.

Insbesondere für das Editieren der Zellen ist die Eigenschaft **Options** wichtig. Options enthält folgende Optionen:

- **goFixedVertLine** die fixierten Spalten des Gitters werden durch vertikale Linien getrennt.
- **goFixedHorzLine** die fixierten Zeilen des Gitters werden durch horizontale Linien getrennt.
- **goVertLine** die verschiebbaren Spalten des Gitters werden durch vertikale Linien getrennt.
- **goHorzLine** die verschiebbaren Zeilen des Gitters werden durch horizontale Linien getrennt.
- **goRangeSelect** der Benutzer kann Zellbereiche auswählen. Falls Options den Wert
- **goEditing** enthält, wird goRangeSelect ignoriert. Dies gilt nicht, wenn goRowSelect in Options enthalten ist.
- **goDrawFocusSelected** die Zellen mit dem Eingabefokus werden ebenso wie die ausgewählten Zellen ohne Eingabefokus in einer bestimmten Farbe angezeigt. Ist der Wert
- **goDrawFocusSelected** nicht enthalten, wird die Zelle mit dem Eingabefokus durch ein Fokusrechteck und nicht durch eine spezielle Hintergrundfarbe gekennzeichnet. Dies gilt nicht, wenn goRowSelect in Options enthalten ist.
- **goRowSizing** die verschiebbaren Zeilen können einzeln vergrößert bzw. verkleinert werden.
- **goColSizing** Die verschiebbaren Spalten können einzeln vergrößert bzw. verkleinert werden.
- **goRowMoving** die verschiebbaren Zeilen können mit der Maus verschoben werden.
- **goColMoving** die verschiebbaren Spalten können mit der Maus verschoben werden.
- **goEditing** der Inhalt der Zellen kann bearbeitet werden. Ist goEditing in Options enthalten, hat goRangeSelect keine Auswirkung.

Informatik 2 Programmieren in Delphi: Stringgrids und Menüs

- **goTabs** der Benutzer kann die Zellen des Gitters mit TAB und UMSCHALT+TAB aufrufen.
- **goRowSelect** es werden gesamte Zeilen ausgewählt und keine einzelnen Zellen. Ist goRowSelect in Options enthalten, hat goAlwaysShowEditor keine Auswirkung.
- goAlwaysShowEditor der Bearbeitungsmodus kann nicht beendet werden. EditorMode muss nicht mit der Eingabetaste oder F2 aktiviert werden. Ist goEditing in Options nicht enthalten, hat goAlwaysShowEditor keine Auswirkung. Ist goRowSelect in Options enthalten, hat goAlwaysShowEditor ebenfalls keine Auswirkung.
- **goThumbTracking** das Bild des Gitters wird aktualisiert, sobald die Positionsmarke in der Bildlaufleiste verschoben wird. Ist goThumbTracking nicht in Options enthalten, wird das Bild erst aktualisiert, nachdem die Maustaste an der neuen Position losgelassen wurde.

Da Options als ein Mengentyp (**SET**) definiert wurde, muss folgende Syntax im Programmtext verwendet werden.

```
Stringgrid1.options:=Stringgrid1.options+ [goEditing];
```

Um z.B. das Editieren einzuschalten.

```
Stringgrid1.options:=Stringgrid1.options-[goColSizing];
```

Um das Verändern der Spaltenbreite abzuschalten. Für ihr Projekt wird es insbesondere wichtig sein, **goEditing zu aktivieren, um Eingaben zu ermöglichen.**

Wenn der Benutzer eine andere Zelle anspringt, so wird das Ereignis **OnSelectCell** ausgelöst. Sie können hier auf eine abgeschlossene Eingabe in einer Zelle reagieren. Beachten sie, dass **OnSelectCell** nur beim Markieren einer anderen Zelle ausgelöst wird, nicht jedoch, wenn der Anwender von einer Zelle auf eine andere Komponente wechselt, z.B. das Hauptmenü. Tritt dies ein, so wird das **OnExit** Ereignis ausgelöst.

OnSelectCell hat als Übergabeparameter die Nummer der markierten Zeile und Spalte sowie die Variable Canselect mit der sie festlegen können, ob das Markieren möglich ist. Die Nummer der Zeile und Spalte kann man an dieser Stelle in eigene Variablen kopieren, da die OnClick und OnExit Ereignisse nicht die Nummer der Zelle als Parameter haben und diese bei Veränderungen benötigt wird.

Align und Scrollbars

Zwei wichtige Eigenschaften, die auch andere grafische Elemente haben, sollen bei dem Stringgrid auch besprochen werden:

Die Eigenschaft **Align** gibt an, ob sich die Komponente an einem der Ränder des Formulars oder Containers (TGroupBox) orientiert und auch bei Größenänderungen des

Informatik 2 Programmieren in Delphi: Stringgrids und Menüs

Formulars diesen folgt.

Folgende Werte kann Align annehmen:

AlNone: Standardverhalten: Sie legen Größe und Position fest.

Altop, Albottom: Die angegebenen Werte für die Breite (Width) werden ignoriert. Die Komponente ist immer so breit wie das Fenster. Sie befindet sich am oberen oder unteren Rand.

Alleft, Alright: Die angegebenen Werte für die Höhe (height) werden ignoriert. Die Komponente ist immer so hoch wie das Fenster. Sie befindet sich am linken oder rechten Rand.

Alclient: Die angegebenen Werte für Höhe und Breite werden ignoriert. Die Komponente nimmt den restlichen Platz ein, der nicht durch andere Komponenten eingenommen wird.

Mit Align=**Alclient** ist es so möglich das Stringgrid in das Fenster ganz einzupassen und auch bei Größenänderungen des Formulars folgt es diesen.

Scrollbars: Legt fest ob bei einer Komponente die ganz nicht in das Fenster passt Bildlaufleisten angezeigt werden. Mögliche Werte sind **ssboth** (beide Bildlaufleisten), **sshorizontal** und **ssvertical** (eine Bildlaufleiste in jeweils der angegebenen Richtung) und **ssnone** (keine Bildlaufleisten).

Menüs

Es gibt zwei Menüarten. Das eine ist das Hauptmenü und das zweite das Popopup-Menü. Das anlegen der Menüeinträge ist in beiden identisch. Nur ist das Hauptmenü immer oben sichtbar, während man das Popupmenü nur sieht wenn auf ein Element rechts geklickt wird.

Das Erstellen geht mit einem Menü Designer, den sie durch zweimaliges Klicken auf die Komponente aufrufen. Mit Peiltaste rechts können sie in Untermenüs wechseln oder wenn sie beim Hauptmenü in der obersten Zeile sind weitere Einträge im Hauptmenü anlegen. Mit der Taste **Einf** können sie an jeder Stelle neue Menüeinträge einfügen und mit der Taste **Entf** wieder löschen.

Das Erstellen von Menüs wird praktisch in der Vorlesung gezeigt. Wichtig ist, dass die Beschriftungen (Caption) zwei Sonderzeichen umfassen:

&: Wird nicht ausgegeben, markiert aber das folgende Zeichen als Abkürzung:

Ö&ffnen markiert das erste „F“. Tippt man, während das Menü offen ist nun ein „F“ ein, so wird dieser Menüeintrag aufgerufen, Das F ist unterstrichen um es zu markieren.

- Gibt eine Trennlinie aus

Eine zweite wichtige Eigenschaft ist **Shortcut**. Hier kann ein Tastenkürzel definiert werden, mit der ein Menüeintrag schnell aufgerufen werden kann. Erlaubt sind als Shortcuts alle Funktionstasten, sowie die Verbindungen von STRG alleine oder mit SHIFT mit den anderen Tasten.

Informatik 2 Programmieren in Delphi: Stringgrids und Menüs

Will man dies im Programm tun, so kann dies geschehen mit den Funktionen:

TextToShortCut (String) – wandelt eine Stringbeschreibung wie „STRG+F4“ in einen Shortcut um.

ShortCutToText (Shortcut) – erzeugt einen String aus einem Shortcut.

Welche Funktion beim Anklicken aufgerufen wird, erfolgt wie gehabt durch Zuweisung einer Routine an den OnClick Teil.

Es könne auch Untermenüs erstellt werden, Menüpunkte können ausgegraut werden (`enabled=false`) nicht angezeigt (`visible=false`) und es kann ein Haken angezeigt werden (`checked=true`). Diese Eigenschaften teilen sie mit anderen visuellen Elementen.

Jede Anwendung kann maximal ein Hauptmenü aufweisen. (Das Hauptmenü kann aus mehreren Untermenüs zusammengesetzt werden, aber es ist immer nur eines aktiv). Die Anwendung kann aber viele Popupmenüs gleichzeitig einsetzen und zwar je Komponente eine und diese können während der Laufzeit auch bequem ausgetauscht werden.

Das Popupmenü hat die gleichen Menüpunkte wie das Hauptmenü, es wird aber nicht automatisch angezeigt. Jede Komponente hat eine Eigenschaft `Popupmenü`. Ihr kann ein `Popupmenü` zugeordnet werden. Befindet sich die Maus dann über dieser Komponente und klickt man auf sie rechts, so wird das `Popupmenü` angezeigt. Befindet man sich auf einer anderen Komponente, so wird deren `Popupmenü` angezeigt oder es passiert nichts wenn keines zugeordnet ist.

So können sie ein Formular mit Editfeldern und einem Listenfeld haben und jedes hat eigene `Popupmenüs`, angepasst an die jeweils sinnvollen Aktionen.

Mehr zu Menüs weiter hinten im Skript bei benutzerfreundlichen Anwendungen.

Praxisteil

Die Anwendung die wir schreiben wollen, greift die Übungsaufgabe aus dem letzten Semester auf. Dort ging es um die Berechnung eines Bremswegs mit und ohne ABS.

Sie sind nun in einer Automobilfirma tätig und sollen die Bremscheiben für verschiedene ABS Systeme entwerfen und haben ein Programm geschrieben, dass die Berechnung in eine Funktion packt. Diese Funktion ist gegeben, und befindet sich als unit

Bremswertberechnung bei den Projekten. Sie enthält zwei Funktionen:

```
function Bremsweg(const masse, flaeche, geschwindigkeit, bremsmoment : double) : double;  
function Bremswegarray(const masse, flaeche, geschwindigkeit, bremsmoment : double) :  
bremsarray;
```

Für die heutige Aufgabe benötigen wir die erste. Sie berechnet den Bremsweg (in Metern). Übergeben wird ihr die Geschwindigkeit in km/h, die Masse des Autos (in kg), die Fläche (in m²) und das Bremsmoment der Scheiben pro Sekunde (in N). Die anderen Parameter sollen feststehen und nicht verändert werden können.

Wir schreiben nun ein Stringgrid bei dem in Spalte 1 die Masse, in Spalte 2, die Fläche, in Spalte 3, das Moment und in Spalte 4 die Geschwindigkeit steht.

Befinden sich in einer Zeile in allen 4 Spalten gültige Werte so soll in der fünften Spalte automatisch der Bremswert berechnet werden.

Über ein Menü und Popupmenü soll ein Anwender jederzeit neue Zeilen hinzufügen können oder löschen können. Am Ende des Stringgrids soll das Hinzufügen auch ohne Menü möglich sein.

Design

Fügen sie zuerst einmal ein Stringgrid ein. Der Name sollte „Grid“ lauten. Ändern sie die Ausrichtung Align auf AlClient, setzen sie die Spaltenzahl colcount auf 5. Setzen sie Fixedcols auf 0, aktivieren sie die Eigenschaft „goediting“ und tragen sie in die Formcreate Routine folgendes ein:

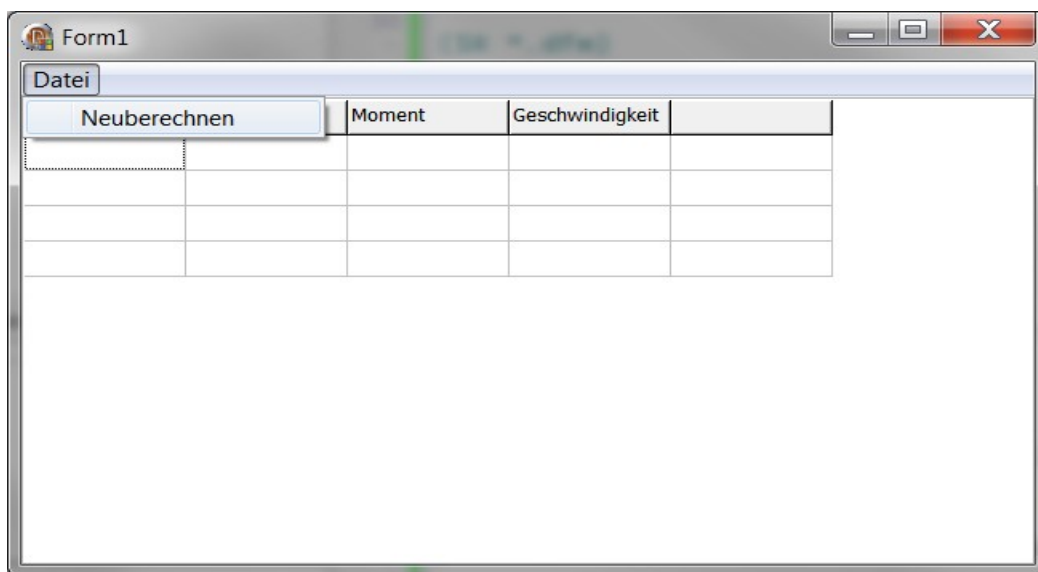
```
procedure TForm1.FormCreate(Sender: TObject);  
begin  
  grid.Cells[0,0] := 'Masse';  
  grid.Cells[1,0] := 'Fläche';  
  grid.Cells[2,0] := 'Moment';  
  grid.Cells[3,0] := 'Geschwindigkeit';  
  grid.Cells[4,0] := 'Bremsweg';  
  grid.DefaultColWidth := 100;  
end;
```

Informatik 2 Programmieren in Delphi: Stringgrids und Menüs

Es gibt keinen Editor, wo sie dies im Objektinspektor erledigen können, also sollten sie die Beschriftungen beim Erzeugen des Formulars anlegen.

Es bietet sich an, da sich die Struktur des Stringgrids laufend ändert (der Benutzer kann ja Zeilen löschen und anfügen) nicht auf die Änderung einer Zelle zu reagieren, sondern eine zentrale Routine zu schreiben, welche bei Aufruf die fünfte Spalte aller Zeilen ändert.

Um diese einmal zu testen fügen wir ein Menü ein, das als Eintrag unter „Datei“ einen Menüpunkt Neuberechnung hat, in dessen Onclickereignis wir die Berechnung einfügen. Das Formular sollte nun so aussehen:



Übung:

Schreiben sie die Routine welche durch die Zeilen des Stringgrids iteriert und jeweils in die fünfte Spalte das Rechenergebnis der Funktion `Bremswert()` einträgt, die sie mit den Werten aus den Spalten 1-4 füttern. Enthalten nicht alle Spalten Werte, so wird ein leerer String in die fünfte Spalte eingetragen. Verwenden sie zum Konvertieren eine der vorderen Stringkonvertierungsroutinen. Für das Eintragen des Ergebnisses können sie den Formatstring `'%.2f'` verwenden.

<ihr Code>

Weisen sie nun diese Neuberechnenroutine dem Onclickereignis der Zellen zu. Es sollte nun bei jedem Klick in eine Zelle die Berechnung erfolgen, wenn eine Zeile gültige Werte enthält.

Es ist sehr mühsam alle Werte nochmals einzutippen, wenn man in einer Zeile nur einen Wert verändern will, z.B. ein anderes Gewicht vorliegt.

Informatik 2 Programmieren in Delphi: Stringgrids und Menüs

Wir ergänzen zuerst das Hauptmenü um ein Menü „Bearbeiten“, dessen erster Menüpunkt nun „Zeile kopieren“ ist. In Die Onclickroutine für diesen Menüpunkt schreiben wir folgendes:

```
procedure TForm1.Zeilekopieren1Click(Sender: TObject);
var i,j : integer;
begin
  grid.RowCount:=grid.RowCount+1;
  for i:=grid.RowCount-2 downto grid.Selection.Top do
    for j:=0 to grid.ColCount-1 do
      begin
        grid.Cells[j,i+1]:=grid.Cells[j,i];
      end;
    end;
end;
```

Übung:

Grid.Selection.Top gibt die oberste selektierte Zeile an. Basierend auf diesem Codestück können sie nun die Routinen für das Einfügen einer leeren Zeile (ohne wie hier den Inhalt der vorherigen Zeile zu kopieren) schreiben.

<ihr Code>

Nicht viel schwieriger ist es eine Zeile zu löschen. Es ist eigentlich eine Umkehrung des Kopierens: Es wird die Kopierrichtung umgedreht und es wird die letzte Zeile gelöscht.

```
procedure TForm1.Zeilelschen1Click(Sender: TObject);
var i,j : integer;
begin
  for i:=grid.Selection.Top to grid.RowCount-2 do
    for j:=0 to grid.ColCount-1 do
      begin
        grid.Cells[j,i]:=grid.Cells[j,i+1];
      end;
    end;
  grid.RowCount:=grid.RowCount-1;
end;
```

Damit wir nun nicht jedes mal in das Menü gehen müssen, fügen wir ein Popupmenü hinzu. Ein Popupmenü hat Menüpunkte wie ein Hauptmenü, aber es gehört jeweils zu einer Komponente, daher muss man es mit der Komponente verbinden. Im Objektinspektor wählen wir daher den Punkt **Popupmenu** aus und wählen hier das Popupmenü. (es können mehrere Popupmenüs in einer Anwendung vorkommen). Danach können wir die Menüpunkte „Zeile löschen“, „Zeile kopieren“ und „Zeile anfügen“ hinzunehmen und die Onclickereignisse auf die schon existierenden Routinen setzen. Danach kann man das Popupmenü durch einen Rechtsklick im Grid auswählen.

Das letzte was unsere Anwendung benötigt, ist eine Sortierfunktion. Sie sollte universell sein, also nach jeder Spalte sortieren können. Damit es schnell geht verwenden wir den Bubblesort. Er geht als Algorithmus so:

```
for i:=Index erstes Element to Index letztes Element-1 do
  for j:=i+1 to Index letztes Element do
    if feld[i]>feld[j] then Tausche(i,J);
```

Das Tauschen führt einen Austausch durch. Das kann bei einfachen Datentypen nur der Austausch des Elementes sein. Hier steht aber der Schlüssel nach dem wir verglichen haben für eine ganze Zeile. Also wenn wir nach der ersten Spalte sortieren wollen:

```
procedure TForm1.Sortieren1Click(Sender: TObject);
var i,j,k : integer;
    temp : string;
begin
  for i:=1 to grid.RowCount-2 do
    for j:=i+1 to grid.RowCount-1 do
      if grid.Cells[1,i]>grid.Cells[1,j] then
        for k:=0 to grid.ColCount-1 do
          begin
            temp:=grid.Cells[k,i];
            grid.Cells[k,i]:=grid.Cells[k,j];
            grid.Cells[k,j]:=temp;
          end;
        end;
    end;
```

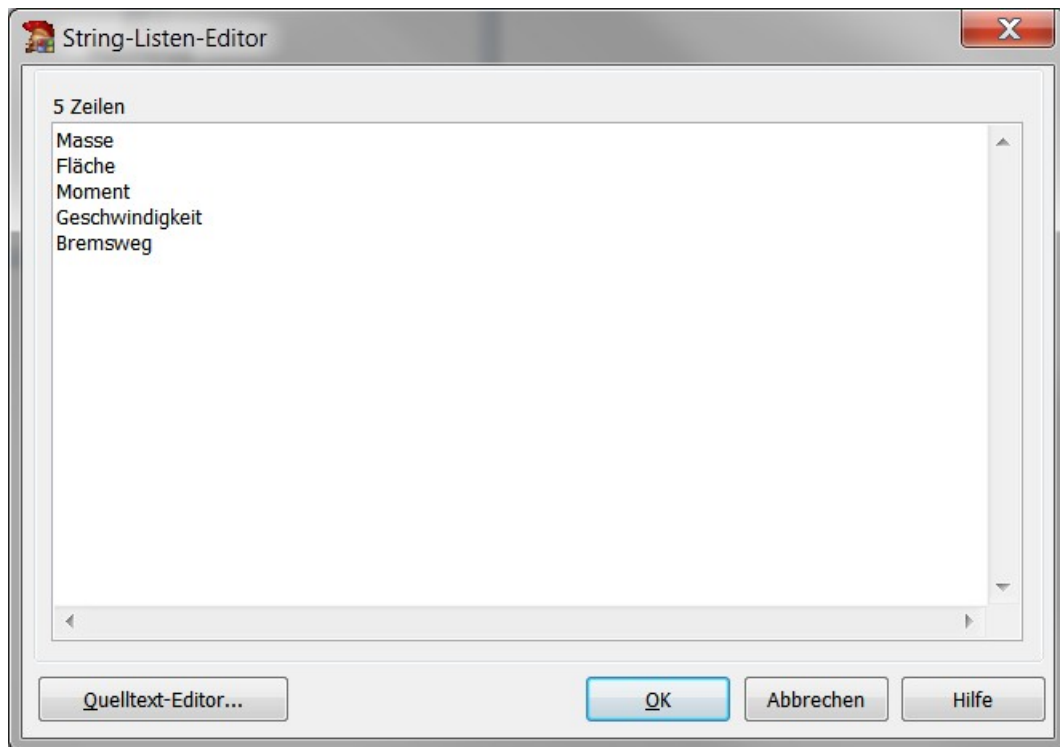
Übung:

Schreiben sie nun basierend auf diesem Code eine universelle Routine, welche die Spalte als Parameter erhält. (Werte: 0 ... Colcount-1).

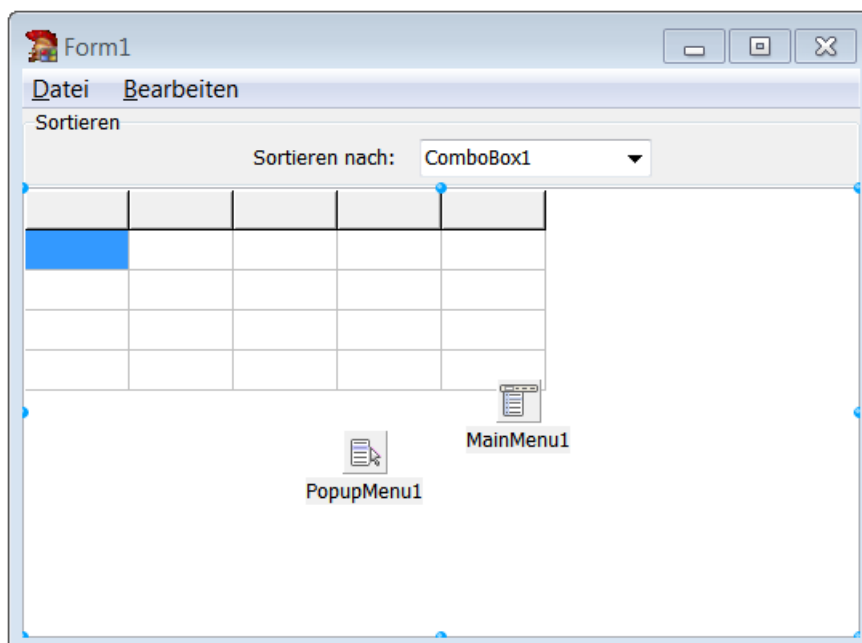
<ihr Code>

Zur Auswahl der Sortiermethode fügen wir oberhalb des Stringgrids eine **Groupbox** ein, die ein **Label** und eine **Combolistbox** aufnimmt. Als Auswahl für die Sortiermethoden fügen wir folgende Elemente hinzu:

Informatik 2 Programmieren in Delphi: Stringgrids und Menüs



Die Anwendung sollte nun so aussehen:



Eine Groupbox dient dazu Elemente zusammenzufassen und zu gruppieren. Neben optischen Gründen (es ist ein Rahmen und eine Beschriftung erkennbar) hat dies auch

programmiertechnische Gründe. So wird hier die Eigenschaft **align** der Groupbox auf **alTop** gesetzt. Damit sind alle Elemente oben fixiert. Innerhalb der Groupbox können sie aber zueinander angeordnet werden.

Eine **Combobox** ist noch vielseitiger als eine Listbox. Sie hat mehrere Darstellungsmöglichkeiten die man mit **Style** einstellen kann. Hier ist es **style=csDropDown** – mit dem Pfeil kann man Einträge auswählen und die Box macht sich klein. Ein Unterschied zu einer Listbox ist, das man oben in die Liste etwas eintippen kann. Dieser Text kann mit der Eigenschaft **Text** abgefragt werden, man kann aber dadurch in einer langen Liste auch schnell Einträge finden. Die Einträge selbst setzt man wie bei einer Listbox über die Eigenschaft **items** und einen markierten Eintrag findet man mit der Eigenschaft **Itemindex** heraus.

Übung:

Setzen sie beim Programmstart die Eigenschaft **Itemindex** auf eine Sortierung nach der Geschwindigkeit und implementieren sie das OnClickereignis für die Combobox, sodass nach der richtigen Spalte sortiert wird.

<ihr Code>

Abschlussarbeiten

Zuletzt gibt es noch an der Oberfläche etwas zu tun. Die Beschichtung des Formulars soll si gestaltet sein, dass sie etwas aussagt, z.b. Bremswertberechnung. Dies können sie im Objektinspektor ändern.

Weiterhin wäre es schön im Dateimenü einen Menüpunkt zu haben, der die Anwendung beendet.

Dann ist es unschön, dass die Spalten nicht die ganze Breite einnehmen. Ideal wäre es die spalten individuell zu setzen, doch wir begnügen und mit einer einfacheren Lösung. Unser Gitter hat 5 Spalten. Also setzen wir beim Ereignis **OnCanResize** einfach die Breite jeder Spalte auf ein Fünftel der neuen Spaltenbreite (Parameter **Newwidth**).

<ihr Code>

Informatik 2 Programmieren in Delphi: Stringgrids und Menüs

Lösungen:

eine mögliche Routine für das Neuberechnen:

```
procedure TForm1.Neuberechnen1Click(Sender: TObject);  
var i : integer;  
    f,m,v,mom : double;  
begin  
    for I:=0 to grid.RowCount-1 do  
        begin  
            grid.Cells[4,i]:= '';  
            if trystrtofloat(grid.Cells[0,i],m) then  
            if trystrtofloat(grid.Cells[1,i],f) then  
            if trystrtofloat(grid.Cells[2,i],mom) then  
            if trystrtofloat(grid.Cells[3,i],v) then  
                grid.Cells[4,i]:=format('%.2f',[Brensweg(m,f,v,mom)]);  
        end;  
    end;
```

Eine mögliche Routine für das Einfügen einer Leerzeile:

```
procedure TForm1.Leerzeileeinfügen1Click(Sender: TObject);  
var i : integer;  
  
begin  
    Zeilekopieren1Click(sender);  
    for i:=0 to grid.ColCount-1 do  
        grid.Cells[i,grid.Selection.Top+1]:= '';  
end;
```

Eine mögliche Routine für das Sortieren nach einer Spalte:

```
procedure TForm1.Sortieren(const spalte : integer);  
  
var i,j,k : integer;  
    temp : string;  
  
begin  
    for i:=1 to grid.RowCount-2 do  
        for j:=i+1 to grid.RowCount-1 do  
            if grid.Cells[spalte,i]>grid.Cells[spalte,j] then  
                for k:=0 to grid.ColCount-1 do  
                    begin  
                        temp:=grid.Cells[k,i];  
                        grid.Cells[k,i]:=grid.Cells[k,j];  
                        grid.Cells[k,j]:=temp;  
                    end;  
        end;  
end;
```

Eine mögliche Routine für das OnClickereignis der Combobox:

```
procedure TForm1.ComboBox1Click(Sender: TObject);  
  
begin  
    Sortieren(combobox1.ItemIndex);  
end;
```

Das Image Objekt

Man kann in Delphi zeichnen. Jedes Objekt hat eine Eigenschaft namens **Canvas**. Diese stellt zahlreiche Methoden zur Verfügung, um zu zeichnen. Es gibt Methoden für einfache geometrische Objekte wie Punkte, Linien, Kreise, Rechtecke.

Das Canvas Objekt heißt nicht nur „Canvas“ - englisch für Leinwand. Sie setzt auch das Konzept einer Leinwand um. So hat die Canvas zwei Sub-Komponenten den **Pen** – einen Zeichenstift für Linien und den **Brush**, einen Pinsel für Flächen. Beide haben Submethoden und Eigenschaften, mit denen man die Stärke, Farbe oder das Muster beeinflussen kann.

Für Zeichenprogramme, aber auch für das Ausgeben von Grafiken, zeichnet man nicht direkt auf das Formular. Stattdessen fügt man eine **TImage** Komponente ein und benutzt deren **Canvas**. Das hat eine Reihe von Vorteilen: Zum einen hat das Imageobjekt weitere Möglichkeiten wie z.B. den Inhalt als Bitmap zu speichern oder zu laden und zum anderen sorgt das Image-Objekt selbst für das Neuzeichnen. Das Neuzeichnen ist notwendig, wenn die Größe verändert wird, aber auch das Formular verdeckt oder wieder freigegeben wird.

Das Image Objekt hat eine Canvas als Subkomponente, aber auch andere nützliche Subkomponenten. So steckt in der Eigenschaft **Picture** das Bild in einem gängigen Grafikformat und kann als solches geladen oder gespeichert werden. Von den vielen Methoden von Timage können wir nur einige hier ansprechen. Eine gute Einführung finden sie auf der Seite

<http://noebis.pi-noe.ac.at/delphi/kapitel12.htm>

Die Konzeption von TCanvas ist diese:

Sie haben eine „Leinwand“ auf der sie mit einem Zeichenstift (**Pen**) oder einem Pinsel (**Brush**) zeichnen. Diese Zeichenwerkzeuge haben eigene Fähigkeiten, wie z.B. Farbe, Muster, Mischmodus mit dem Untergrund. Über die Eigenschaften der Werkzeuge wird das Aussehen der elementaren Operationen festgelegt.

Für unser Projekt brauchen wir folgende Methoden:

Von

TImage.Canvas.Pen

- Die Eigenschaft **Width**: Width ist die Pinselbreite in Pixeln.
- Die Eigenschaft **Color**: Color ist die Farbe. Die Farbe ist ein 24-Bit-RGB Wert, das bedeutet: In einer Zahl steckt der Rotanteil (untere 8 Bits), Grünanteil (Bits 8-15) und Blauanteil (Bits 16-23). Einfacher ist es oft Konstanten zu benutzen die vordefiniert sind, wie clblue für Blau oder clred für Rot.

Mit dem Pinsel (**Brush**) wird auf den Hintergrund gemalt. Auch er hat eine Farbe (**Color**). Wir brauchen den Pinsel um die Zeichenfläche jeweils zu leeren, wenn wir nach einer

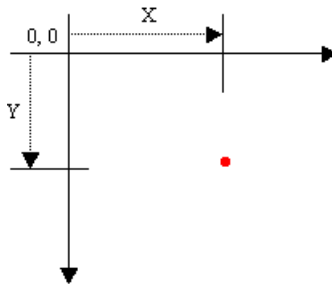
Informatik 2 Programmieren in Delphi: Stringgrids und Menüs

Änderung der Formel oder der Grenzen diese neu gezeichnet werden muss.

Dazu nutzen wir die Methode **Fillrect**. Sie nimmt einen Record vom Typ **TRect** entgegen. Der muss schon existieren. Am geschicktesten löscht man so die Zeichenfläche:

```
var Rect: TRect; // Koordinaten Record definieren  
  
.....  
Rect.Left:=0; // Von oben links  
Rect.Top:=0;  
Rect.Right:=Image1.Width; // Nach unten rechts  
Rect.Bottom:=Image1.Height;  
Image1.Canvas.Brush.Color:=ClWhite; //Pinselfarbe festlegen  
Image1.Canvas.FillRect(Rect);  
.....
```

Das leitet uns über, zu den Koordinatensystem von Windows. Es weicht in einem Punkt von dem üblichen kartesischen Koordinatensystem ab: Der Ursprung ist links oben und nicht links unten:



Die Breite der Grafik wird durch **Canvas.Width** angegeben und die Höhe durch **Canvas.Height**. Die Untergrenze ist immer (0,0).

Wann immer wir irgendwelche Messwerte zeichnen sollen, wie in dieser Anwendung den Bremsweg, so ist es notwendig, die Werte zu skalieren, also so anzupassen, dass sie das Fenster maximal ausfüllen. Wie wandelt man nun ein Koordinatensystem in ein anderes um, z.B. die vom Anwender gegebene?

Es sind zwei Schritte. Zuerst muss für jede Achse ein Skalierungsfaktor gewonnen werden. Er ist notwendig um den Platz vollständig auszufüllen. Nehmen wir an, wir haben gegeben:

Grafikfläche : 1000 Pixel breit, 500 Pixel hoch.

Gegebene Daten für den Bremsweg: Anfangsgeschwindigkeit 200 km/h, Bremsweg 2000 m.

Dann bilden wir den Bremswert auf die X-Achse ab, und die Geschwindigkeit auf die Y-Achse.

Der Skalierungsfaktor ist die Differenz des Größten X-wertes - kleinsten X-Wertes geteilt durch die Anzahl der Pixel. In unserem Fall ist der kleinste X-wert 0, sodass es sich reduziert zu:

Informatik 2 Programmieren in Delphi: Stringgrids und Menüs

Fakx = 2000 m / 1000 Pixel = 2 m/Pixel

analog für die Y-Achse:

Faky = 200 km/h / 500 Pixel = 0,4 km/Pixel*h

Der zweite Schritt ist eine Verschiebung des Ursprungs. Er ist notwendig, wenn wir auch negative X/Y Werte haben, z.B. wenn wir einen Funktionsplotter schreiben. Wir verschieben den Ursprung indem wir von jedem X/Y-Wert en kleinsten Wert abziehen, bevor wir ihn skalieren.

Das Skalieren geht nun so:

Bilde den Skalierungsfaktor Fakx/Faky

Für jedes Pixel bilde: (X-Wert-kleinster X-Wert)/Fakx

bzw.

(Y-Wert-kleinster Y-Wert)/Faky

Für den Y-wert müssen wir nun noch Berücksichtigen, dass „0“ oben links und der Maximalwert unten links ist. Wir bilden daher

Y-Koordinate:= Canvas.Height – Y-Koordinate.

Zeichenmethoden

Die beiden wesentlichen Methoden zum Zeichnen sind

MoveTo(X,Y) und

LineTo(X,Y).

Moveto bewegt den Zeichenstift an eine neue Position, ohne eine Linie zu zeichnen. LineTo zieht dann eine Linie zu dem angegebenen Punkt. Mit den Eigenschaften von Pen (Linienbreite = Wdith, Farbe=Color) erfolgt das Zeichnen. Will man also verschiedene Linien in verschiedenen Farben / Breiten haben, so muss man zwischendurch die Eigenschaften von Pen ändern.

Wenn sie Text ausgeben wollen, so machen sie dies mit

```
Canvas.TextOut(X,Y,'Text'); // Gibt den Text an der Position X,Y aus
```

Der Font kann festgelegt werden über die Font Eigenschaft von TCanvas:

```
Canvas.Font.Name:='Arial'; // Fontname  
Canvas.Font.size:=12; // Fontgröße
```

Zu weiteren Routinen zum Zeichnen konsultieren sie am besten die obige Website oder die Hilfe unter dem Stichwort Tcanvas.

Bilder Laden und Speichern

Die Eigenschaft Picture eines Image Objektes (nicht der Canvas) enthält das Bild in einem Metaformat das Windows speichern kann. Standardmäßig sind dies neben einigen Vektorformaten nur das Bitmap Format. Andere Formate können nachgerüstet werden.

Folgende Methoden sind in Tpicture für uns wichtig:

```
TImage.Picture.SaveToFile(Dateiname);  
TImage.Picture.LoadFromFile(Dateiname);
```

Damit wird ein Bild gespeichert. Es gibt zum Abfragen des Dateinamens auch einen TopenPicturedialog und einen TsavePicturedialog der nur Bilder als Dateien anzeigt und auch eine Vorschau des Bildes bietet. Wichtig ist auch:

```
Clipboard.Assign(TImage.Picture.Bitmap);
```

zum Kopieren in die Zwischenablage. Wobei natürlich TImage durch die Bezeichnung ihres Imageobjektes ersetzt werden muss und zum Benutzen der Zwischenablage in ihre **Uses** Liste die Unit **Clipbrd** aufgenommen werden muss:

```
implementation  
Uses Clipbrd;
```

Das Ausdrucken ist etwas komplizierter. Es basiert darauf, das Bild auf die Druckfläche des Papiers zu skalieren und skaliert auf den Drucker zu zeichnen. Eine einfache Routine zum Ausdrucken von **Image1** sieht so aus:

```
uses  
  Printers;  
  
procedure TForm1.Button1Click(Sender: TObject);  
var  
  ScaleX, ScaleY: Integer;  
  RR: TRect;  
begin  
  with Printer do  
  begin  
    BeginDoc;  
    // Mit BeginDoc wird ein Druckauftrag initiiert.  
    try  
      ScaleX := GetDeviceCaps(Handle, logPixelsX) div PixelsPerInch;  
      ScaleY := GetDeviceCaps(Handle, logPixelsY) div PixelsPerInch;  
      // Informationen über die Auflösung  
      // Retrieves information about the Pixels per Inch of the Printer.  
      Canvas.StretchDraw(RR, Image1.Picture.Graphic);  
      // An die Auflösung anpassen  
    finally  
      EndDoc; //Methode EndDoc beendet den aktuellen Druckauftrag  
    end;  
  end;  
end;
```

Praxisteil

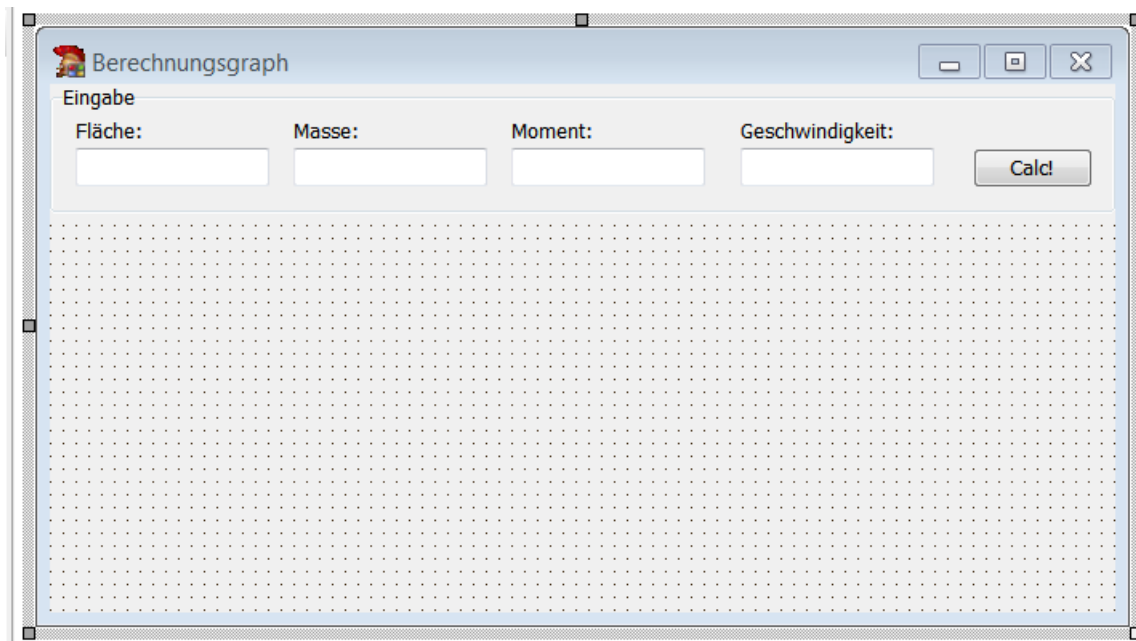
Wir knüpfen heute an die Bremswertberechnung aus der letzten Vorlesungsstunde an. Wir benötigen erneut die Unit Bremswertberechnung. Nur diesmal die Routine

```
function Bremswegarray(const masse, flaeche, geschwindigkeit, bremsmoment : double) :  
bremsarray;
```

Sie gibt einen Typ Bremsarray zurück, dessen Definition wir schließlich als ein dynamisches Array eines Records erkennen:

```
type bremsarray = array of bremsdaten;  
type bremsdaten = record  
  weg : double;  
  geschwindigkeit : double;  
  zeit : double;  
end;
```

Das erste was wir machen ist einmal eine Anwendung schreiben die in einer Groupbox Editfelder aufnimmt, In ihr geben wir die Eingabedaten (Masse, Fläche, Geschwindigkeit, Bremsmoment) ein. Die Daten prüfen wir mit derselben Routine die wir schon beim Stringgrid benutzten. So sollte unsere erste Ansicht aussehen:



Beim Klicken auf „Calc“ konvertieren wir die Werte und rufen die Berechnungsroutine auf. Den Rückgabewert speichern wir in einer lokalen Variable „erg“:

Informatik 2 Programmieren in Delphi: Stringgrids und Menüs

```
procedure TForm2.Button1Click(Sender: TObject);
var i : integer;
    f,m,v,mom : double;
    erg : bremsarray;

begin
  if trystrtofloat(labelededit1.Text,m) then
  if trystrtofloat(labelededit2.Text,f) then
  if trystrtofloat(labelededit3.Text,mom) then
  if trystrtofloat(labelededit4.Text,v) then
    erg:=Brenswegarray(m,f,v,mom);
end;
```

Fügen sie nun ein Image Objekt ein, und setzen sie die Eigenschaft Align auf alclient.

Nun geht es an das Erstellen der Grafik. Das geschieht in mehreren Schritten. Zuerst zeichnen wir nur einmal die Messkurve. Beim ersten Punkt müssen wir mit **Moveto** zu diesem gehen und bei allen folgenden ziehen wir vom vorherigen Punkt eine Linie mit **Lineto**. Dazu ist für jeden Punkt eine Koordinatentransformation nötig, die wir in einer eigenen Routine durchführen. Da der Skalierungsfaktor überall der gleiche ist, wird er separat berechnet. Wenn wir uns das Array ansehen, dann enthält es aber drei Werte pro Messpunkt: Zeit, Weg und Geschwindigkeit. Zeit steht immer für die X-Achse. Die Y-Achse kann der Weg oder die Geschwindigkeit sein (weg/Zeit oder Geschwindigkeit/Zeit Diagramm). Wir entscheiden uns im ersten Teil für ein Geschwindigkeit/Zeit Diagramm.

So sehe die Routine aus wenn wir uns auf die X-Koordinate beschränken und nur die Moveto Funktion für den ersten Bildpunkt durchführen:

```
procedure TForm2.Button1Click(Sender: TObject);
var i : integer;
    f,m,v,mom : double;
    erg : bremsarray;
    fakx,faky : double;
    x,y : integer;

Function ScaleX(Const X,X_untergrenze : double) : integer;

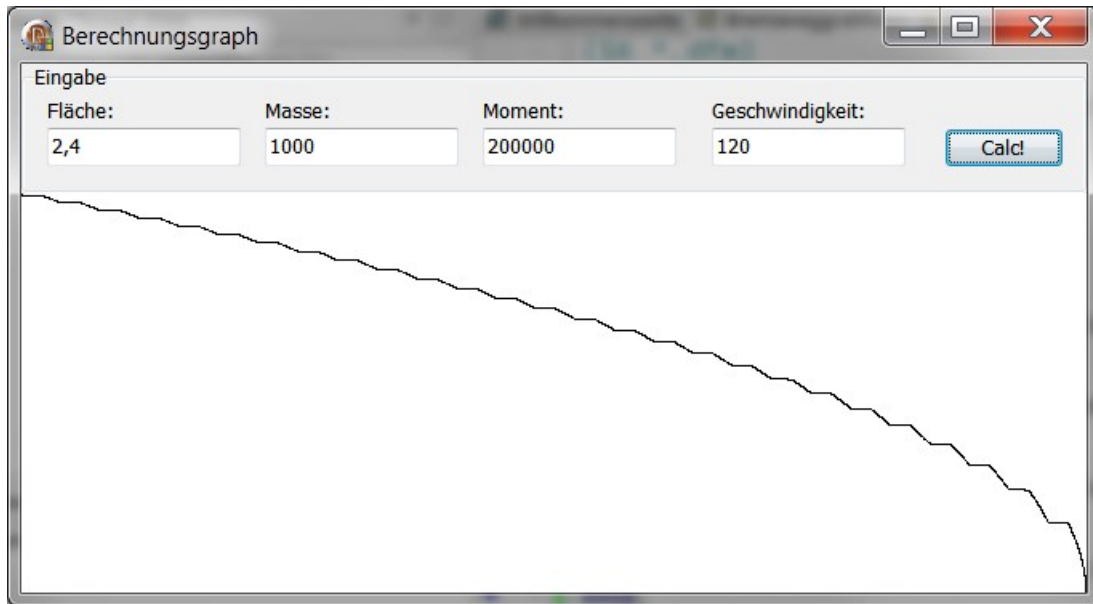
begin
  result:=Trunc((X-X_untergrenze)/Fakx*image1.width);
end;

begin
  if trystrtofloat(labelededit1.Text,m) then
  if trystrtofloat(labelededit2.Text,f) then
  if trystrtofloat(labelededit3.Text,mom) then
  if trystrtofloat(labelededit4.Text,v) then
    erg:=Brenswegarray(m,f,v,mom);
    fakx:=erg[high(erg)].zeit-erg[0].zeit;
    x:=scalex(erg[0].zeit,erg[0].zeit);
    image1.Canvas.MoveTo(x,y);
end;
```

Übung:

Ergänzen sie diese nun um die Berechnung von Faky, die ScaleY Funktion und die Berechnung der Bildschirmkoordinaten für Y. Als zweiten Schritt ziehen sie die Linien bis zum letzten Punkt in einer Schleife. Achten Sie darauf wann die Geschwindigkeit Minimal ist da dies die Untergrenze ist. Ihr Ergebnis sollte nun so aussehen:

Informatik 2 Programmieren in Delphi: Stringgrids und Menüs



Hinweis für die Praxis: Dort kann es sinnvoll sein das Fakx und Faky gleich groß sind um Verzerrungen zu vermeiden. Dann setzt man $Fakx := \text{Max}(fakx, Faky)$ (analog bei Faky).

Deutlich ist beim Betätigen des ABS, dass die Geschwindigkeit alle 100 ms rascher absinkt. Danach wird es für 100 ms gelöst um ein blockieren der Räder zu verhindern.

In einem zweiten Schritt wollen wir die Linie einmal in Dunkelblau ausgeben und mit einer breiteren Linie: Wir fügen vor der Moveto Routine ein:

```
Image1.Canvas.Pen.Width:=2;  
Image1.Canvas.Pen.Color:=clnavy;
```

Wenn Sie zweimal auf Calc klicken, werden sie feststellen, dass die zweite Linie die erste überlagert. Das ist unerwünscht, weil die erste natürlich andere Skalierungsfaktoren hat als die zweite, sie also nicht verglichen werden können.

Übung:

Löschen sie vor dem Zeichnen die Zeichenfläche. Testen sie sie durch zweimaliges klicken auf „Calc“

<ihr Code>

Das nächste ist es nun ein zwei Achsen einzuzeichnen. Eine Achse beim Zeitpunkt $t=0$ (Achse) und eine für $v=0$ (X-Achse). Diese soll in Schwarz sein und 1 Pixel breit. So sieht der Code für die X-Achse aus:

```
Procedure XAchse;
```

```
begin
```

```
Image1.Canvas.Pen.Width:=1;  
Image1.Canvas.Pen.Color:=clblack;  
x:=scalex(erg[0].zeit, erg[0].zeit);  
y:=scaley(0, erg[high(erg)].geschwindigkeit);  
Image1.Canvas.moveto(x,y);
```

Informatik 2 Programmieren in Delphi: Stringgrids und Menüs

```
x:=scalex(erg[high(erg)].zeit,erg[0].zeit);
Image1.canvas.lineto(x,y);
end;
```

Übung:

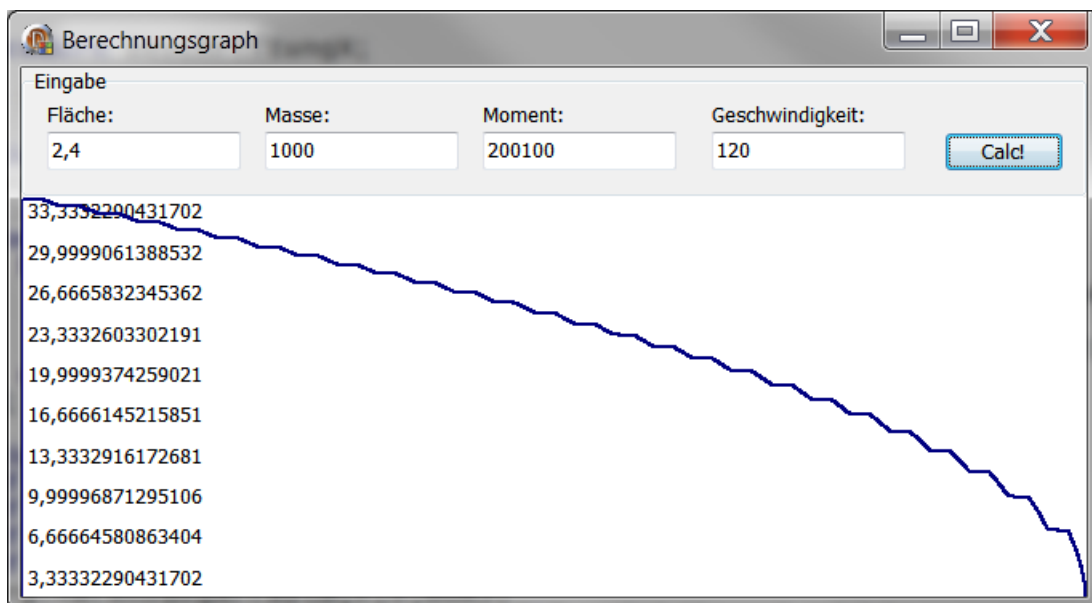
Schreiben sie nun den Code für die Y-Achse.

<ihr Code>

Die Achse alleine macht wenig Sinn. Sinnvollerweise sollten wir in regelmäßigen Abständen die X- und Y-Werte ausgeben. Sinnvoll wäre eine Unterteilung der Achsen in 10 Teile, z.B. für die Achse so:

```
procedure BeschriftungY;
var i : integer;
begin
  image1.Canvas.Pen.Color:=clblack;
  x:=scalex(erg[0].zeit,erg[0].zeit)+4;
  for i:=1 to 10 do
    begin
      y:=scaley(erg[0].geschwindigkeit/10*i,erg[high(erg)].geschwindigkeit);
      Image1.canvas.moveto(x,y);
      Image1.Canvas.TextOut(x,y,Floattostr(erg[0].geschwindigkeit/10*i));
    end;
end;
```

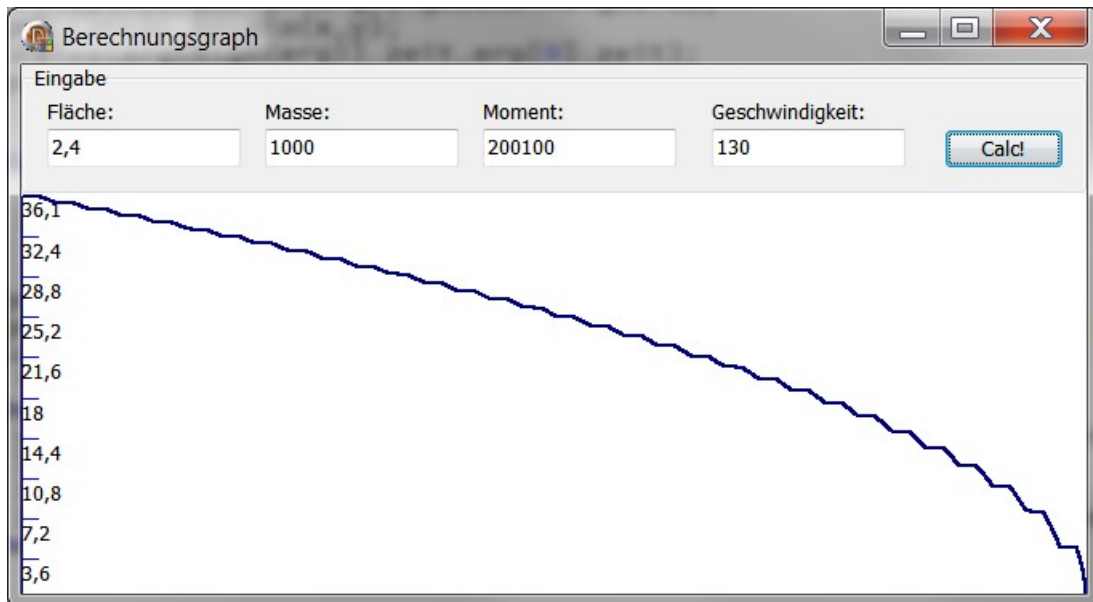
Die Beschriftung ist nun allerdings „suboptimal“ wie folgende Abbildung zeigt:



Sinnvoll ist es zu runden. Für diese Werte, die alle Vorkommastellen haben, reicht es den Wert mal 10 zu nehmen, davon den ganzzahligen Anteil zu nehmen und dieses Ergebnis durch 10 zu teilen. Dadurch entfallen alle Nachkommastellen außer der ersten.

Übung:

Passen sie die Ausgabe an. Ziehen sie auch eine 10 Pixel breite Linie von der X-Achse um zu zeugen wozu die Beschriftung gehört. Die Ausgabe sollte nun so aussehen:

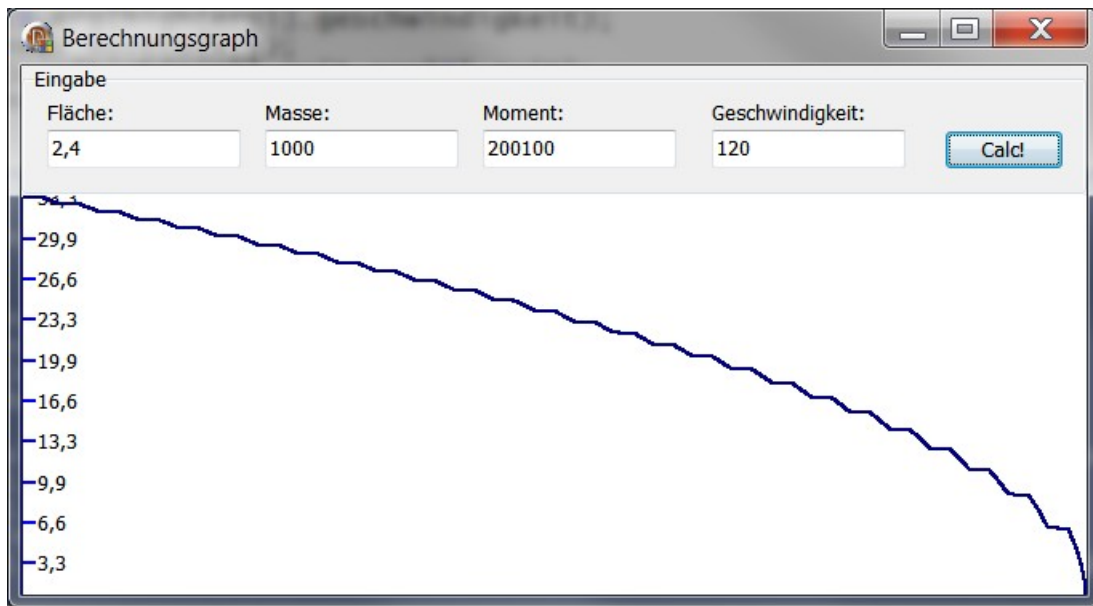


Die Beschriftung ist unterhalb der Linie, da der Textursprung oben links ist. Wir können das aber leicht korrigieren:

```
procedure BeschriftungY;  
var i : integer;  
    wert : double;  
begin  
    image1.Canvas.Pen.Color:=clblue;  
    image1.Canvas.Pen.Width:=2;  
    x:=scalex(erg[0].zeit,erg[0].zeit);  
    for i:=1 to 10 do  
        begin  
            wert:=Trunc(10*erg[0].geschwindigkeit/10*i)/10;  
            y:=scaley(wert,erg[high(erg)].geschwindigkeit);  
            Image1.Canvas.MoveTo(x,y);  
            Image1.Canvas.LineTo(x+10,y);  
            y:=y-Image1.Canvas.TextHeight('H') div 2;  
            Image1.Canvas.TextOut(x+10,y,Floattostr(wert));  
        end;  
end;
```

Nun sieht die Ausgabe so aus:

Informatik 2 Programmieren in Delphi: Stringgrids und Menüs

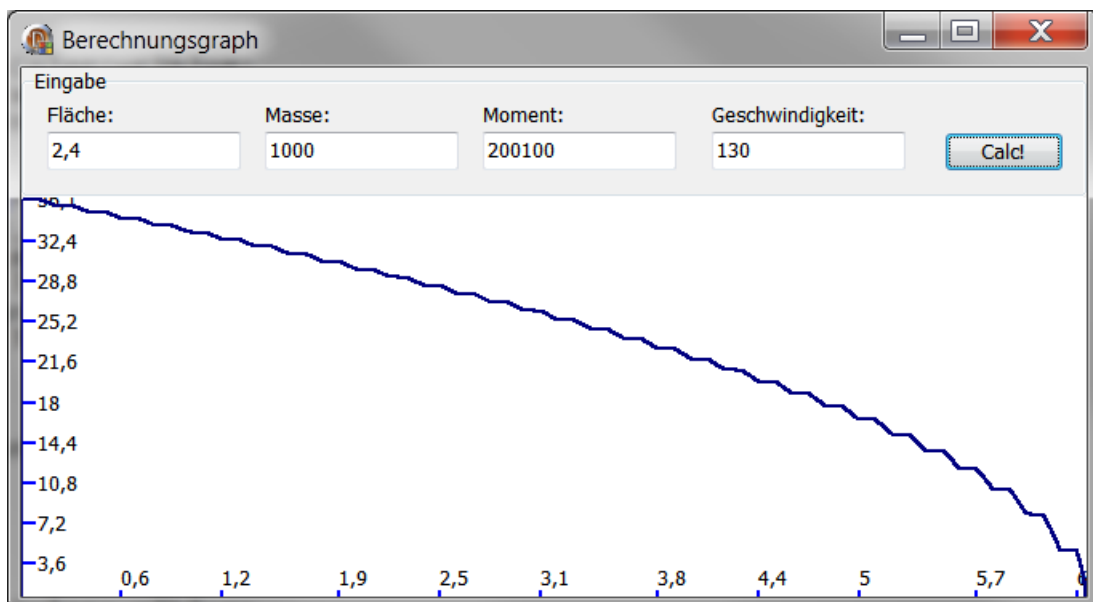


Übung:

Schreiben sie die entsprechende Routine für die Beschriftung der anderen Achse.

<ihr Code>

Es sollte nun so aussehen:



Informatik 2 Programmieren in Delphi: Stringgrids und Menüs

Das letzte ist es dass wir noch eine Möglichkeit einbauen das Bild zu speichern. Fügen sie einen Button, einen Picturesavedialog ein und schreiben sie in die OnClickroutine:

```
procedure TForm2.Button2Click(Sender: TObject);
begin
  if SavePictureDialog1.Execute then
  begin
    image1.picture.SaveToFile(SavePictureDialog1.filename);
  end;
end;
```

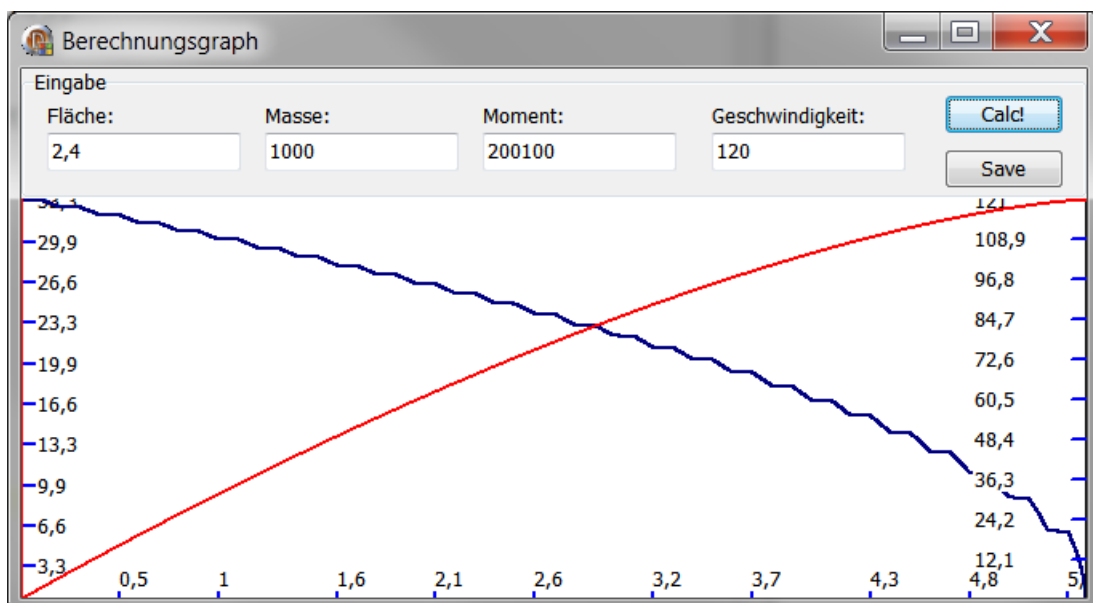
Der Weg

Bisher haben wir nur eine von zwei Größen, die Geschwindigkeit aufgezeichnet. Analog ist die Vorgehensweise bei dem Weg. Nur sollten wir, damit wir beide Grafiken eintragen können, nicht das Bild löschen. Die zweite Y-Achse können wir rechts eintragen und rot beschriften

Die Vorgehensweise für das Einzeichnen und die Y-achse ist analog zur Geschwindigkeit, nur muss ein anderer X-Wert für den Startpunkt der Textausgabe gewählt werden. Am besten setzen sie diese rechts. Beachten Sie das der Weg sich wie die Zeit verhält, also der kleinste Weg bei Erg[0] und der größte bei Erg[high(Erg)] steckt.

Frage: Wie kann man den jeweils kleinsten / größten Wert feststellen, ohne sich an einem bestimmten Index orientieren zu müssen?

Es sollte nun so aussehen:



Informatik 2 Programmieren in Delphi: Stringgrids und Menüs

Lösung: Eine vervollständigte Berechnungsroutine für den Graphen:

```
procedure TForm2.Button1Click(Sender: TObject);  
  
var i : integer;  
    f,m,v,mom : double;  
    erg : bremsarray;  
    fakx,faky : double;  
    x,y : integer;  
  
Function Scalex(Const X,X_untergrenze : double) : integer;  
  
begin  
    result:=Trunc((X-X_untergrenze)/Fakx);  
end;  
  
Function Scaley(Const y,y_untergrenze : double) : integer;  
  
begin  
    result:=image1.Height-trunc((y-y_untergrenze)/Faky);  
end;  
  
begin  
    if trystrtofloat(labelededit1.Text,f) then  
    if trystrtofloat(labelededit2.Text,m) then  
    if trystrtofloat(labelededit3.Text,mom) then  
    if trystrtofloat(labelededit4.Text,v) then  
        erg:=Brenswegarray(m,f,v,mom);  
        fakx:=(erg[high(erg)].zeit-erg[0].zeit)/image1.Width;  
        faky:=(erg[0].geschwindigkeit-erg[high(erg)].geschwindigkeit)/image1.Height;  
        x:=scalex(erg[0].zeit,erg[0].zeit);  
        y:=scaley(erg[0].geschwindigkeit,erg[high(erg)].geschwindigkeit);  
        image1.Canvas.MoveTo(x,y);  
        for i:=1 to high(erg) do  
            begin  
                x:=scalex(erg[i].zeit,erg[0].zeit);  
                y:=scaley(erg[i].geschwindigkeit,erg[high(erg)].geschwindigkeit);  
                image1.Canvas.LineTo(x,y);  
            end;  
        end;  
end;
```

Setzen der Farbe und Breite für den Graphen:

```
Image1.Canvas.Pen.Width:=2;  
Image1.Canvas.Pen.Color:=clnavy;
```

Mögliche Routine für das Löschen des Bildes:

```
image1.Canvas.FillRect(Rect(0,0,image1.Width,image1.Height));
```

Mögliche Y-Achsen:

```
Procedure YAchse;  
  
begin  
    image1.Canvas.Pen.Width:=1;  
    image1.Canvas.Pen.Color:=clblack;  
    x:=scalex(erg[0].zeit,erg[0].zeit);  
    y:=scaley(0,erg[high(erg)].geschwindigkeit);  
    Image1.Canvas.MoveTo(x,y);  
    y:=scalex(erg[0].geschwindigkeit,erg[high(erg)].geschwindigkeit);  
    Image1.Canvas.LineTo(x,y);  
end;
```

Informatik 2 Programmieren in Delphi: Stringgrids und Menüs

Mögliche X-Achsentextroutine:

```
procedure BeschriftungX;  
var i : integer;  
    wert : double;  
begin  
    image1.Canvas.Pen.Color:=clblue;  
    image1.Canvas.Pen.Width:=2;  
    Y:=scaley(erg[high(erg)].geschwindigkeit,erg[high(erg)].geschwindigkeit);  
    for i:=1 to 10 do  
        begin  
            wert:=Trunc(10*erg[high(erg)].zeit/10*i)/10;  
            x:=scalex(wert,erg[0].zeit);  
            Image1.Canvas.MoveTo(x,y);  
            Image1.Canvas.LineTo(x,y-20);  
            Image1.Canvas.TextOut(x,y-20,Floattostr(wert));  
        end;  
    end;
```

Gesamter Quelltext nach Eintrag des Weges:

```
unit Bremsweggrahform;  
interface  
uses  
    windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,  
    Dialogs, StdCtrls, ExtCtrls, ExtDlgs;  
type  
TForm2 = class(TForm)  
    GroupBox1: TGroupBox;  
    LabeledEdit1: TLabeledEdit;  
    LabeledEdit2: TLabeledEdit;  
    LabeledEdit3: TLabeledEdit;  
    LabeledEdit4: TLabeledEdit;  
    Button1: TButton;  
    Image1: TImage;  
    Button2: TButton;  
    SavePictureDialog1: TSavePictureDialog;  
    procedure Button1Click(Sender: TObject);  
    procedure Button2Click(Sender: TObject);  
    private  
        { Private-Deklarationen }  
    public  
        { Public-Deklarationen }  
end;  
var  
    Form2: TForm2;  
implementation  
uses Bremswertberechung;  
{ $R *.dfm }  
procedure TForm2.Button1Click(Sender: TObject);  
var i : integer;  
    f,m,v,mom : double;  
    erg : bremsarray;  
    fakx,faky : double;  
    x,y : integer;
```

Informatik 2 Programmieren in Delphi: Stringgrids und Menüs

```
Function Scalex(Const X,X_untergrenze : double) : integer;  
begin  
  result:=Trunc((X-X_untergrenze)/Fakx);  
end;  
Function Scaley(Const y,y_untergrenze : double) : integer;  
begin  
  result:=image1.Height-trunc((y-y_untergrenze)/Faky);  
end;  
Procedure XAchse;  
begin  
  image1.Canvas.Pen.Width:=1;  
  image1.Canvas.Pen.Color:=clblack;  
  x:=scalex(erg[0].zeit,erg[0].zeit);  
  y:=scaley(0,erg[high(erg)].geschwindigkeit);  
  Image1.canvas.moveto(x,y);  
  x:=scalex(erg[high(erg)].zeit,erg[0].zeit);  
  Image1.canvas.lineto(x,y);  
end;  
Procedure YAchse;  
begin  
  image1.Canvas.Pen.Width:=1;  
  image1.Canvas.Pen.Color:=clblack;  
  x:=scalex(erg[0].zeit,erg[0].zeit);  
  y:=scaley(0,erg[high(erg)].geschwindigkeit);  
  Image1.canvas.moveto(x,y);  
  y:=scalex(erg[0].geschwindigkeit,erg[high(erg)].geschwindigkeit);  
  Image1.canvas.lineto(x,y);  
end;  
Procedure YAchse2;  
begin  
  image1.Canvas.Pen.Width:=1;  
  image1.Canvas.Pen.Color:=clblack;  
  x:=scalex(erg[high(erg)].zeit,erg[0].zeit);  
  y:=scaley(0,erg[high(erg)].weg);  
  Image1.canvas.moveto(x,y);  
  y:=scalex(erg[0].weg,erg[high(erg)].weg);  
  Image1.canvas.lineto(x,y);  
end;  
procedure BeschriftungY;  
var i : integer;  
    wert : double;  
begin  
  image1.Canvas.Pen.Color:=clblue;  
  image1.Canvas.Pen.Width:=2;  
  x:=scalex(erg[0].zeit,erg[0].zeit);  
  for i:=1 to 10 do  
    begin  
      wert:=Trunc(10*erg[0].geschwindigkeit/10*i)/10;  
      y:=scaley(wert,erg[high(erg)].geschwindigkeit);  
      Image1.canvas.moveto(x,y);  
      Image1.canvas.lineto(x+10,y);  
      y:=y-image1.Canvas.TextHeight('H') div 2;  
      Image1.Canvas.TextOut(x+10,y,Floattostr(wert));  
    end;  
end;
```

Informatik 2 Programmieren in Delphi: Stringgrids und Menüs

```
procedure BeschriftungY2;
var i : integer;
    wert : double;

begin
  image1.Canvas.Pen.Color:=clblue;
  image1.Canvas.Pen.Width:=2;
  x:=image1.Width-10;
  for i:=1 to 10 do
  begin
    wert:=Trunc(10*erg[high(erg)].weg/10*i)/10;
    y:=scaley(wert,erg[0].weg);
    Image1.Canvas.MoveTo(x,y);
    Image1.Canvas.LineTo(x+10,y);
    y:=y-image1.Canvas.TextHeight('H') div 2;
    Image1.Canvas.TextOut(x-60,y,Floattostr(wert));
  end;
end;

procedure BeschriftungX;
var i : integer;
    wert : double;

begin
  image1.Canvas.Pen.Color:=clblue;
  image1.Canvas.Pen.Width:=2;
  Y:=scaley(erg[high(erg)].geschwindigkeit,erg[high(erg)].geschwindigkeit);
  for i:=1 to 10 do
  begin
    wert:=Trunc(10*erg[high(erg)].zeit/10*i)/10;
    x:=scalex(wert,erg[0].zeit);
    Image1.Canvas.MoveTo(x,y);
    Image1.Canvas.LineTo(x,y-20);
    Image1.Canvas.TextOut(x,y-20,Floattostr(wert));
  end;
end;

begin
  if trystrtofloat(labelededit1.Text,f) then
  if trystrtofloat(labelededit2.Text,m) then
  if trystrtofloat(labelededit3.Text,mom) then
  if trystrtofloat(labelededit4.Text,v) then
  erg:=Brenswegarray(m,f,v,mom);
  fakx:=(erg[high(erg)].zeit-erg[0].zeit)/image1.Width;
  faky:=(erg[0].geschwindigkeit-erg[high(erg)].geschwindigkeit)/image1.Height;
  image1.Canvas.FillRect(Rect(0,0,image1.Width,image1.Height));
  XAchse;
  YAchse;
  BeschriftungX;
  BeschriftungY;
  image1.Canvas.Pen.Width:=2;
  image1.Canvas.Pen.Color:=clnavy;
  x:=scalex(erg[0].zeit,erg[0].zeit);
  y:=scalex(erg[0].geschwindigkeit,erg[high(erg)].geschwindigkeit);
  image1.Canvas.MoveTo(x,y);
  for i:=1 to high(erg) do
  begin
    x:=scalex(erg[i].zeit,erg[0].zeit);
    y:=scaley(erg[i].geschwindigkeit,erg[high(erg)].geschwindigkeit);
    image1.Canvas.LineTo(x,y);
  end;
  // nun den weg
  faky:=(erg[high(erg)].weg-erg[0].weg)/image1.Height;
  YAchse2;
  BeschriftungY2;
```

Informatik 2 Programmieren in Delphi: Stringgrids und Menüs

```
Image1.Canvas.pen.width:=2;
Image1.Canvas.pen.color:=clred;
x:=scalex(erg[0].zeit,erg[0].zeit);
y:=scalex(erg[0].weg,erg[0].weg);
Image1.Canvas.MoveTo(x,y);
for i:=1 to high(erg) do
begin
  x:=scalex(erg[i].zeit,erg[0].zeit);
  y:=scaley(erg[i].weg,erg[0].weg);
  Image1.Canvas.lineTo(x,y);
end;
end;

procedure TForm2.Button2Click(Sender: TObject);
begin
  if SavePictureDialog1.Execute then
  begin
    Image1.picture.SaveToFile(SavePictureDialog1.filename);
  end;
end;
end.
```

Benutzerfreundliche Anwendungen

Eine Anwendung muss nicht nur ihre Funktion erfüllen, sie muss auch bestimmten Kriterien genügen, die nicht funktionell definiert werden können. Die Benutzerfreundlichkeit ist ein wichtiger Teil der Software Ergonomie. An dieser Stelle ist es nicht möglich alle Prinzipien zu verdeutlichen, doch die wichtigsten. Beachten sie beim Design von Anwendungen auf Folgendes:

Nutzen sie das Vorwissen des Anwenders, indem sie etablierte Standards nutzen:

- Verwenden sie die Standard-Menüstruktur von Windows (Datei-Menü, Bearbeiten Menü, Hilfe Menü etc.) wo Anwender zuerst nach bestimmten Menüpunkten suchen (wie Dateien öffnen und Druck im Datei Menü). Versuchen sie ihre Operationen in diese Menüs einzusortieren, also das Speichern in das Datei Menü.
- Verwenden sie gängige Shortcuts wie STRG-X für Ausschneiden (und nicht STRG-A oder ALT+Shift+STRG+F12).
- Verwenden Sie Kontextmenüs für Operationen, die nicht immer möglich sind, oder vom Inhalt abhängig sind. Kontextmenüs kann der Anwender mit der rechten Maustaste aufrufen.

Halten sie die Komplexität überschaubar:

- Wenn ihr Einstellungsdialog zu viele Optionen hat, so machen sie mehrere Dialoge daraus oder verwenden sie ein Reitersystem.
- Fassen sie zusammengehörige Dinge zusammen und trennen sie nicht zusammengehörige in separate Dialoge auf.
- Gliedern sie, wenn ein Menü zu lang wird, es in Untermenüs auf. Nicht benötigte Einträge können ausgeblendet werden.
- Lassen sie Dialoge und Fenster auf sich wirken: Sind die aufgeräumt, übersichtlich oder überfrachtet, unübersichtlich und unsymmetrisch?
- Wenn sie Toolbars verwenden – achten sie auf die Anzahl und blenden sie nur die ein die aktuell nötig sind.

Geben Sie Hilfestellungen!

- Verwenden sie die Hints von Delphi.
- Legen sie eine Hilfe an und Sprungzeile zu den Punkten, bei denen Hilfe nötig ist.
- Programmieren sie einen Assistenten, wenn ihr Programm zu komplex ist.

Informatik 2 Programmieren in Delphi: Stringgrids und Menüs

Vermeiden sie Fehleingaben!

- Prüfen sie Eingaben, so früh wie möglich.
- Akzeptieren sie keine Fehleingaben, wenn diese die Funktionalität des Programmes beeinträchtigen.
- Bieten sie eine einfache Möglichkeit in einen definierten Zustand zu kommen: z.B. über einen Eintrag „Standard wiederherstellen“

Verringern sie die nötigen Eingaben!

- Führen sie eine History der Eingaben, aus denen der Anwender auswählen kann
- Überlegen Sie, ob es Sinn macht, die aktuellen Eingaben eines Dialogs bei Programmende zu sichern und beim nächsten Start automatisch einzuladen.
- Macht es Sinn automatisch Vorgabewerte bereitzustellen (z.B. bei einer CD-Sammlung automatisch den Albumtitel des letzten Datensatzes, in den nächsten kopieren oder die Nummer hochzählen)

Braten Sie keine Extrawurst!

- Halten sie sich an Windows Farbschemas und Fonts.
- Setzen sie unbekannte grafische Elemente oder zweckentfremdete Elemente nur mit Bedacht ein.
- Nutzen sie, wo möglich, Standard Elemente, Standardgrößen und Farben.

Behindern sie den Anwender nicht!

- Eingabemasken sollten logisch aufgebaut sein, sodass der Anwender sie möglichst schnell ausfüllen kann.
- Menüpunkte und Felder, die momentan nicht zur Verfügung stehen werden, ausgegraut und nicht weggeblendet. (Ausgrauen: Eigenschaft `enabled:=false`, weggeblendet: Eigenschaft `Visual:=false`).
- Beschränken sie sich bei Meldungsdialogen, die der Anwender wegklicken muss, auf wichtige Hinweise, da diese den Arbeitsablauf stören. Andere Hinweise, die wichtig sein können, können sie über Hints anzeigen oder die Beschriftung von Statusfeldern ändern.

Schriftarten:

- Verwenden Sie Standard Schriftarten. Wenn sie nichts angeben, so übernehmen sie die normalen Windows Schriftarten. Eine Abweichung von diesen muss begründet sein, weil sie meist mit den Einstellungen kollidiert, die der Anwender in den Windows Einstellungen für die Anzeige machen kann.
- Grafische Anwendungen sollten serifenlose Schriftarten wie Arial oder Tahoma

Informatik 2 Programmieren in Delphi: Stringgrids und Menüs

verwenden. Meiden Sie serifenhaltige Schriften wie Times oder Helvetica. Diese sind für flüssiges Lesen in Büchern ausgelegt und wirken auf Formularen unsauber.

- **Absolutes No-Go:** Sie haben eine Schriftart gewählt, die der Anwender gar nicht installiert hat (aber z.B. Sie, weil sie eine Schriftartsammlung installiert haben - Windows sucht eine Ersatzschriftart und wählt diese dafür aus.

Robustheit, Fehlertoleranz

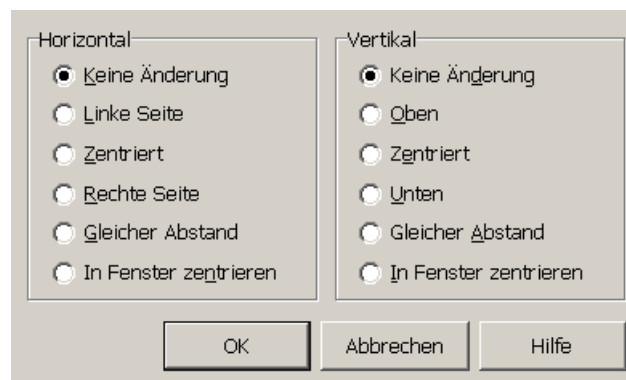
Zu der Benutzerfreundlichkeit gehört auch das Erkennen und Behandeln von Fehlern. Fehler des Benutzers sind meist falsche Eingaben, entweder weil er nicht korrekt informiert ist oder sich vertippt hat. Gute Programme reduzieren die Möglichkeiten für Fehleingaben (z.B. verwenden sie einen Kalender um Geburtsdaten einzugeben oder Eingabefelder die nur Zahlen akzeptieren für numerische Eingaben) und Sie überprüfen Fehler so schnell wie möglich: Ganz schlecht ist es einen Benutzer 20 Felder ausfüllen zu lassen, um ihm dann mitzuteilen, das er im Ersten einen Fehler gemacht hat und alles noch mal eingeben muss. Webshops, die diese Fehler machen, überleben meist nicht sehr lange.

Möglichkeiten, die Delphi für Benutzeroberflächen bietet.

Delphi kann einem das Design nicht abnehmen, aber es kann assistieren. Folgende Möglichkeiten bietet das System.

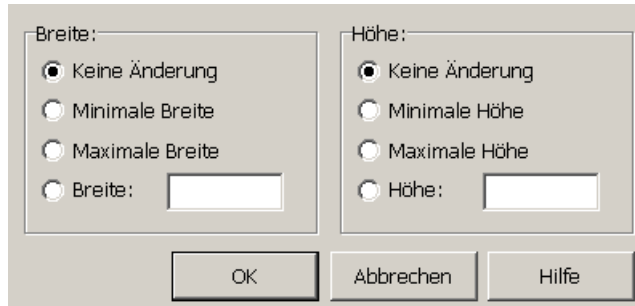
Unter dem Menüpunkt **Tools** → **Umgebungsoptionen** → **Designer** kann festgelegt werden, das neue Elemente an einem festen Raster ausgerichtet werden und wie groß dieses Raster in Pixeln ist. Damit kann man sehr einfach grafische Elemente untereinander ausrichten, ohne sie genau zu positionieren. Delphi bietet auch Hilfslinien beim Bewegen, die es einem erlauben, Komponenten zueinander auszurichten.

Mehrere Elemente kann man zueinander ausrichten, indem man sie mit der Maus markiert (entweder einen Rahmen um sie ziehen oder sie bei gedrückter Shift-Taste nacheinander anklicken) und dann auf die rechte Maustaste klickt. Im **Kontextmenü "Position"** gibt es den Menüpunkt **ausrichten**:



Informatik 2 Programmieren in Delphi: Stringgrids und Menüs

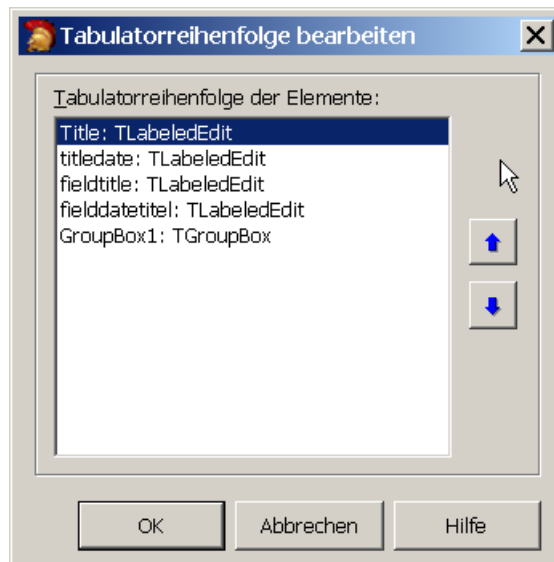
Damit kann die Position recht genau festgelegt werden. Die Höhe und Breite ist für mehrere Elemente festlegbar, indem man entweder die Elemente markiert und im **Objektinspektor** in den Feldern „**Width**“ und „**Height**“ von Hand neue Werte eingibt, oder indem man im selben Kontextmenü den Eintrag „**Größe**“ verwendet:



Bei grafischen Elementen kann der **Fokus** (welches Element gerade eine Eingabe entgegen nimmt) mittels der **Tabulatortaste** verschoben werden. Anders als mit der Maustaste geht dies aber nur in zwei Richtungen (vorwärts mit Tab, und rückwärts mit Shift+Tab). Trotzdem ist das Durchlaufen eines Eingabeformulars mit diesen Tasten schneller als mit der Maus und routinierte Anwender arbeiten fast ausschließlich so.

Die Tabulatorreihenfolge wird vorgegeben, von der Reihenfolge, in der Sie Elemente in das Formular hinzufügen. Wenn diese nicht die Eingabereihenfolge ist, oder sie neue Elemente hinzufügen, müssen sie diese selbst bearbeiten:

Klicken Sie dazu ins Formular und wählen sie im **Kontextmenü** den Menüpunkt „**Tabulatorreihenfolge**“. Sie erhalten folgenden Dialog:



Das erste Element erhält als Erstes den Fokus und danach in der Reihenfolge der Liste die folgenden.

Menüs

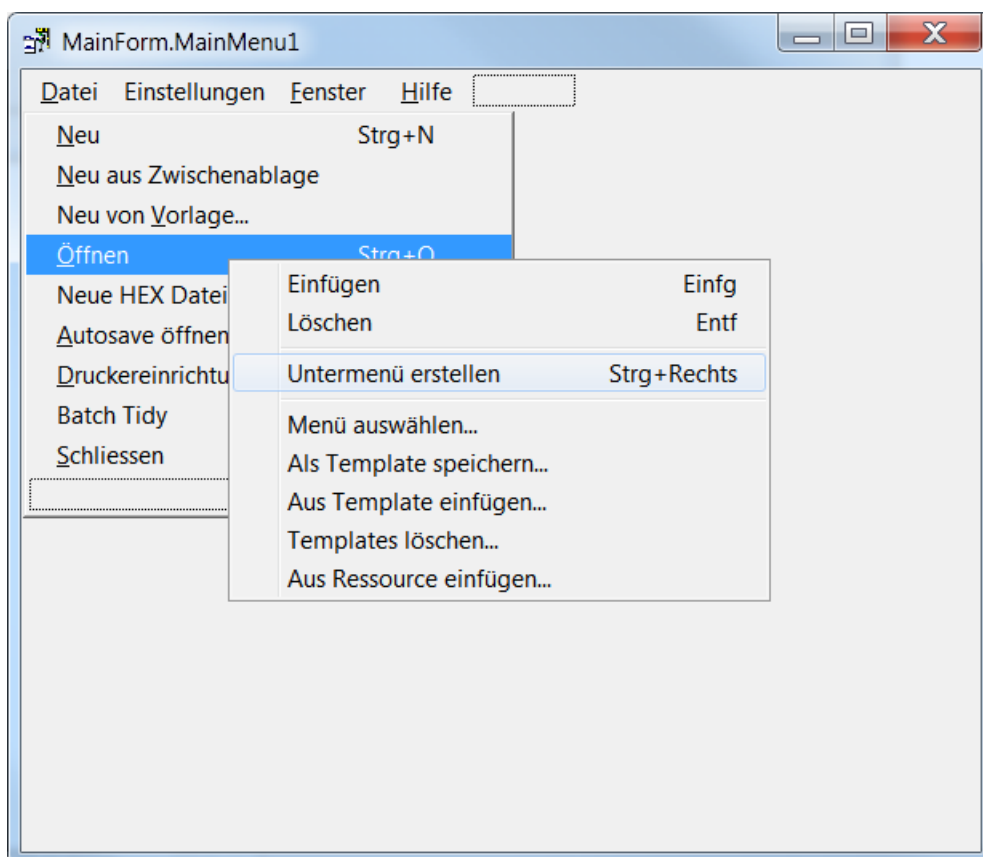
Menüs werden im Menüdesigner bearbeitet. Dazu wird eine **TMainMenu** Komponente zu einem Formular hinzugefügt und auf diese doppelt geklickt. Es erscheint eine visuelle Bearbeitungsmöglichkeit.

Innerhalb des Menübaums kann man mit den Pfeiltasten navigieren. Einen Menueintrag (vom Typ **TMenuItem**) fügt man durch die Einfügen-Taste hinzu und löscht ihn mit der ENTF-Taste. Die Beschriftung steht wie üblich bei Caption. Mit **OnClick** verknüpft man einen Klick auf den Eintrag mit einem Ereignis.

Bei der Beschriftung gibt es zwei Besonderheiten: Das Zeichen „&“ hebt einen Buchstaben hervor. Er erlaubt es mittel ALT+Buchstabe einen Menüpunkt aufzurufen. Besteht die Beschriftung nur aus dem Text „-“ so wird ein Trenner im Menü angezeigt.

Über den Menüdesigner kann man auch Untermenüs anlegen und so einen Menübaum aufbauen.

Den **Shortcut** eines Menüs (z.B. STRG-N für den Menüpunkt „Neu“ wird über die Eigenschaft Shortcut festgelegt.



Popupmenüs

Mit Popupmenüs kann man Komponenten eigene Kontextmenüs zuweisen. Kontextmenüs erscheinen nur, wenn man rechts klickt. Es gibt nur ein Hauptmenü. Es wird zugewiesen, indem man der Eigenschaft **Menu** des Formulars eine **TMainMenu** Komponente zuweist.

Dagegen hat jede Komponente eine Popupmenu Eigenschaft. So kann zum einen jeder Komponente (z.B. jedem Editfeld) ein Kontextmenü zugewiesen werden. Diese Eigenschaft wird vererbt – weist man einem Formular ein Popupmenü zu und legt nichts anderes fest, so wird dieses Popupmenü allen Komponenten zugewiesen. Popupmenüs sind vom Typ **Tpopupmenu** und werden wie das Hauptmenü mit dem Menüdesigner bearbeitet. Eine Anwendung kann mehrere Tpopupmenus besitzen.

Während der Programmausführung geschieht dies mit der Methode Assign:

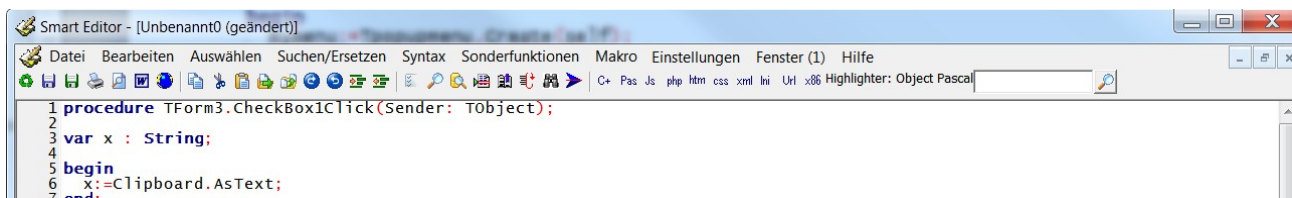
```
procedure TForm3.CheckBox1Click(Sender: TObject);
var x : String;
begin
  x:=Clipboard.AsText;
end;

procedure TForm3.Edit1Click(Sender: TObject);
Var mymenu: Tpopupmenu;
    menuitem : Tmenuitem;
begin
  mymenu:=Tpopupmenu.Create(self);
  menuitem:=Tmenuitem.Create(mymenu);
  menuitem.Caption:='Neues Menu';
  menuitem.OnClick:=CheckBox1Click;
  mymenu.Items.Add(menuitem);
  edit1.PopupMenu.Assign(mymenu);
end;
```

Dies zeigt auch, wie zur Laufzeit neue Menüpunkte erzeugt werden.

Toolbars

Eine Toolbar ist ein Feld unterhalb des Hauptmenüs, das grafische Buttons für die häufigsten Operationen aufnimmt:



Sie wird hinzugefügt indem eine **TToolbar** Komponente zur Anwendung hinzugenommen wird. Klickt man auf die Ttoolbar rechts, so erscheinen im Kontextmenu die Befehle „**Neuer Schalter**“ und „**neuer Trenner**“. Letzteres fügt einen Vertikalen Strich ein,

Informatik 2 Programmieren in Delphi: Stringgrids und Menüs

Ersteres einen Button.

Der Eigenschaft **MenuItemem** kann nun ein Menüpunkt zugewiesen werden. Das Bild stammt aus einer Liste von Bildern. Dazu fügt man der Anwendung eine **TImageList** Komponente hinzu.

Klickt man auf diese Imageliste doppelt, so kann man Bilder zu der Liste hinzufügen, dort verschieben, löschen etc. Dies sieht z.B. so aus:



Symbole findet man im Internet und im Ordner „Gemeinsame Dateien/Codegear Shared/Images“. Es sollten Bitmap Typen sein und sie sollten klein (maximal 32 Pixel) groß sein.

Für das Benutzen weist man der Toolbar Eigenschaft Imagelist die Liste zu und bei jedem Schalter weist man der Eigenschaft Imageindex eines Schalters eine Nummer aus der Liste zu. Bei der „2“ würde z.B. der Schalter das Infosymbol zeigen.

Analog kann man auch neben Menüeinträgen Bilder anzeigen.

Hilfen

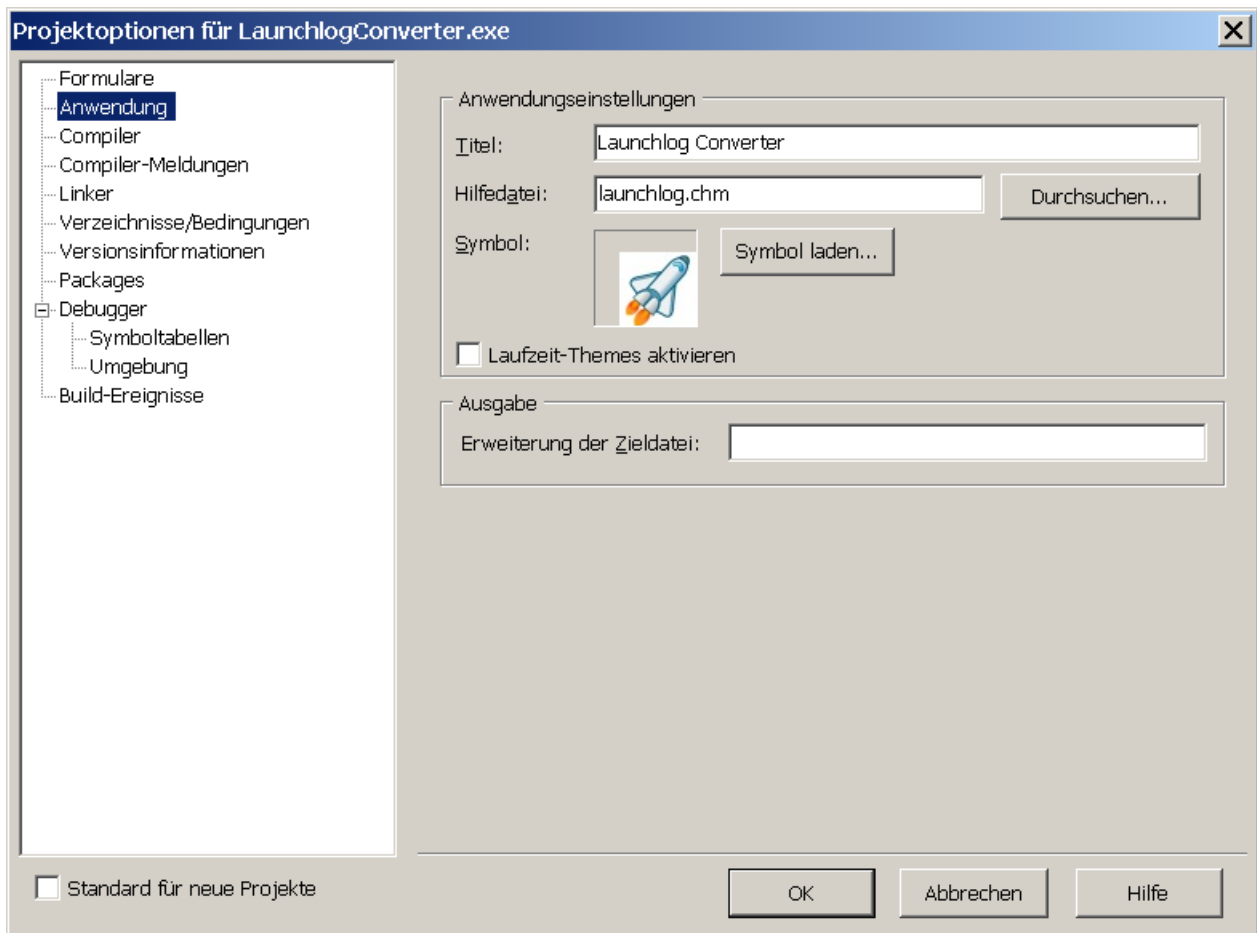
Für eine echte Helpdatei benötigen sie ein Hilfsprogramm um sie zu erstellen (die Möglichkeiten mit dem Helpcompiler der mit Delphi mitgeliefert wird, ermöglichen zwar die Erstellung einer Hilfe, bieten aber keinen Komfort bei der Erstellung). Doch das Grundsätzliche kann hier angesprochen werden.

Jede Hilfeseite (Bildschirmseite oder Eintrag) hat eine Context-ID. Dies ist eine Ganzzahl. Wenn sie ein grafisches Element markieren und im **Objektinspektor** bei **HelpContext** eine Zahl eingeben, so wird beim Klicken auf F1 (Hilfe aufrufen) genau die Hilfseite in der

Informatik 2 Programmieren in Delphi: Stringgrids und Menüs

Hilfe aufgerufen. Analog wird die Suche über Schlüsselwörter (**HelpKeyword**) durchgeführt.

Die Hilfsdatei kann über zwei Mechanismen bereitgestellt werden. Zum einen in den Projekteigenschaften. Klicken Sie dazu im Menü auf „**Ansicht** → **Projekteigenschaften**“. Wählen sie das Projekt aus und klicken sie dann rechts. Wählen sie hier den Menüpunkt „**Optionen**“. Sie sehen dann folgenden Dialog:



Der Titel ist die Bezeichnung des Programms. Sie sehen diese in der Taskleiste und wenn sie mit Alt+Tab über die Tasks wechseln. Sie können hier auch ein schönes Icon auswählen (Dateityp: ico, gibt es im Internet) und eben die Hilfsdatei auswählen. Es gibt zwei Typen: die alten Hilfsdateien (Endung .hlp) und die modernen HTML Hilfen (Endung .chm).

Im Programm ist dies aber auch möglich, z.B., wenn sie eine Hilfetaste implementieren:

```
procedure Ttitleform.Hilfe_ButtonClick(Sender: TObject);
begin
  Application.HelpFile:='Meine Hilfe.hlp';
  Application.HelpContext(1100);
end;
```

Die erste Zeile legt die Hilfedatei fest. Die Zweite öffnet sie, indem sie eine bestimmte Seite anhand ihrer Contextid öffnet.

Das Application Objekt steht für das Programm und ist jedem Formular übergeordnet.

Neben den Hilfen gibt es auch noch **Hints**. Hints (Hinweise) sind kurze Texte, die erscheinen, wenn sie mit der Maus eine Weile über einem Element ruhen. Den Text geben sie bei der Eigenschaft „**Hint**“ im **Objektinspektor** ein. Die Ausgabe muss aber dann noch eingeschaltet werden. Dies geschieht, indem "**ParentShowHint**" auf true gesetzt wird. Wichtig: Dies muss auch im Formular erfolgen (Per Standard ist bei ParentShowHint im Formular ausgeschaltet).

Eigene Hinweise und Meldungen

Das Anzeigen von Hints und Hilfen erfolgt normalerweise aktiv durch den Anwender. Hints erscheinen, wenn sie längere Zeit über einem Element verweilen, das einen Hint hat und die Hilfe fordern sie aktiv an.

Eigene Meldungen sollten unterteilt werden in Statusmeldungen (oder Informationen) und wichtige Meldungen, die den Arbeitsablauf aufhalten oder beeinflussen.

Statusmeldungen informieren den Anwender über Dinge, erfordern aber keine Aktion. Dies sind z.B.:

- Daten über die Position im Text, den man bearbeitet
- Voraussichtliche Dauer, die ein Scan eines Laufwerks auf Viren dauert
- Gerade bearbeitete Datei eines Batch Jobs.

Wichtige Meldungen darf der Anwender nicht ignorieren oder er muss agieren. Das sind z.B.

- Einlegen einer CD zur Datensicherung.
- Fehler beim Schreiben in eine Datei - Eingabe eines neuen Dateinamens?
- Herunterfahren des Computers in 30 Sekunden.

Statusmeldungen gibt man einfach aus. Möglichkeiten dazu sind:

- Titelleiste (**Caption**) des Formulars: z.B. bei einer Textverarbeitung dort den aktuellen Dateinamen
- **Label** mit einem Ergebnis
- **TStatusBar** am unteren Rand der Anwendung. Dieses hat mehrere Panels für verschiedene Abschnitte kann aber auch mit den Eigenschaften „Simplepanel:=true und Simpletext:="Statustext" nur einen Text ausgeben.
- Fortschrittsindikatoren bei längeren Operationen (Dateien verarbeiten etc.):

TProgressbar

Wichtige Meldungen erfordern ein Einschreiten des Anwenders. Sie kennen schon die ShowMessage Methode, die einen einfachen Dialog auf den Bildschirm zaubert:

Showmessage('Datei kann nicht geöffnet werden');



Etwas flexibler ist die MessageDlg Routine, die folgendes Format hat:

```
function MessageDlg(const Msg: string; DlgType: TMsgDlgType;  
  Buttons: TMsgDlgButtons; HelpCtx: Longint): Integer;
```

Der erste Parameter **Msg** ist ein String mit der Nachricht, die ausgegeben wird. Der zweite Parameter **DlgType** ist eine Integerkonstante, die den Typ angibt. Es gibt Folgende:

```
type  
  TMsgDlgType = (mtWarning, mtError, mtInformation, mtConfirmation, mtCustom);
```

Das erzeugt folgende Fenster:

mtinformation



mterror



Zuletzt kann noch angegeben werden, welche Buttons sichtbar sind. Es gibt Buttons mit

Informatik 2 Programmieren in Delphi: Stringgrids und Menüs

der Beschriftung "Ja", "Nein", "Ok", "Abbrechen", "Wiederholen", "Ignorieren", "Alle", "Nein zu allen", "Ja zu allen", "Hilfe".

Dies sind die folgenden Konstanten. Sie können mehr als einen Button angeben und müssen diese als Menge in einer eckigen Klammer angeben.

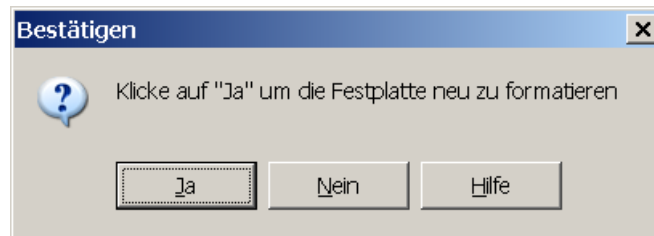
type

```
TMsgDlgType = (mtWarning, mtError, mtInformation, mtConfirmation, mtCustom);
TMsgDlgBtn = (mbYes, mbNo, mbOK, mbCancel, mbAbort, mbRetry, mbIgnore,
             mbAll, mbNoToAll, mbYesToAll, mbHelp);
TMsgDlgButtons = set of TMsgDlgBtn;
```

Beispiel:

```
MessageDlg('Klicke auf "Ja" um die Festplatte neu zu formatieren',
           mtconfirmation, [mbyes, mbno, mbhelp], 0);
```

ergibt folgenden Dialog.



Der letzte Parameter ist eine Contextid der Hilfe, die angesprungen wird, wenn der Anwender Hilfe anfordert. Geben sie hier eine Zahl oder 0 ein, wenn dies nicht erforderlich ist.

Der Rückgabewert der Funktion ist eine Integerkonstante und zwar eine die den Typ des vom Benutzer angeklickten Buttons enthält. Die Konstante unterscheidet sich von den Buttons nur dadurch, dass sie nicht mit "Mb" sondern, mit "Mr" anfängt. So könnte eine Reaktion auf folgenden Dialog so aussehen:

```
case MessageDlg('Klicke auf "Ja" um die Festplatte neu zu
formatieren', mtconfirmation, [mbyes, mbno, mbhelp], 0) of
  mryes:  Format('C');
  mrno:   exit;
  mrhelp: Application.HelpKeyword('Formatieren');
end;
```

Wenn sie den Rückgabewert nicht benötigen, so rufen sie den Dialog einfach als Prozedur auf:

```
MessageDlg('Klicke auf "Ja" um die Festplatte neu zu
formatieren', mtconfirmation, [mbyes, mbno, mbhelp], 0);
```

Die anderen Dialogtypen können sie durch Ausprobieren selbst feststellen. Es erscheint also jeweils ein Symbol und die Beschriftung ist angepasst "Information / Fehler")

Ausblenden von Elementen

Elemente die zeitweise nicht zur Verfügung stehen (z.b. der Menüpunkt „speichern“ wenn keine Datei geöffnet wird) kann man ausblenden oder abschalten. Ausblenden bedeutet: Das Element ist nicht sichtbar. Der Menüpunkt fehlt im Menü. Dies wird gemacht, indem man die Eigenschaft "**Visual**" auf false setzt. Da dies meist zur Laufzeit erfolgt, geschieht dies meist durch Programmcode:

```
procedure Ttitleform.Ausschalten;  
begin  
  Speichernunter_Menu.Visible:=false;  
end;
```

Das Zweite ist das Ausblenden. Das Element ist dann noch sichtbar. Aber es ist nicht aktiv. Es wird ausgegraut. Elemente die Eingaben entgegen nehmen könnten wie Editfelder, erlauben keine Veränderung mehr. Dies geht über die Eigenschaft **Enabled**.

In der Regel sollten Elemente nur ausgeblendet, aber nicht abgeschaltet werden. Anwender suchen nach Menüpunkten und wenn sie fehlen sind, sie verwirrt. Eine Ausnahme sind die Kontextmenüs. Sie sollten nur Befehle aufnehmen, die aktuell sinnvoll sind.

```
procedure Ttitleform.Ausschalten;  
begin  
  Speichernunter_Menu.Enabled:=false;  
end;
```

Programmtechnische Arbeiten

Viele Arbeiten können nur zur Laufzeit erledigt werden. Sie sind von den Eingaben abhängig oder was der Anwender tut. Hier einige Dinge, die zur Laufzeit durchgeführt werden sollten:

- Vorbelegung von Dialogen mit Standardwerten oder den aus einer Datei eingelesenen letzten Eingaben.
- Ausblenden / Deaktivieren von Menüpunkten abhängig von den aktuellen Eingaben – z.b. Deaktivieren von PasteFromClipboard, wenn die Zwischenablage leer ist.
- Zuweisen von passenden Kontextmenüs abhängig von den Eingaben
- Einblenden und Ausblenden von Toolfeldern.
- Anfügen von Eingaben zur Liste der letzten Ergebnisse, Durchnummerierung von Datensätzen. Vorbelegung von Standardfeldern des nächsten Satzes mit den Eingaben des letzten Satzes...

Diese können sie beim Klicken auf Bestätigen Buttons, beim Anzeigen eines Formulars (OnShow) oder beim Erzeugen (OnCreate) oder Zerstören (OnDestroy) durchgeführt werden.