ISBN 3-8273-1775-4

3 Hello World ...?

Wie in jedem Buch so gibt es auch hier ein Beispielprogramm, das übliche »Hello World«-Programm. Aus zwei Gründen habe ich mich allerdings gegen dieses Minimalprogramm entschieden. Zum einen wird es in jedem Fachbuch immer wieder aufgewärmt, zum zweiten ist die Applikationsentwicklung mit Delphi so einfach und schnell zu erledigen, dass wir ruhig zu einem etwas umfangreicheren Beispiel greifen können, um das Programmieren mit Delphi anhand eines Programms zu erklären.

Das Programm, das wir schreiben wollen, soll ein kleines Rechenprogramm sein, ein Brutto-Netto-Rechner. Aus einem Bruttowert und einem Prozentsatz soll der Nettowert errechnet werden.

3.1 Erstellen der Oberfläche

Normalerweise ist die Oberfläche das Letzte, was an einem Programm optimiert wird. In der Regel wird bei visuellen Programmiersystemen, zu denen auch Delphi gehört, die Konzentration zunächst auf die Funktionalität gelegt und erst danach die Oberfläche gestaltet. Für die Praxis bedeutet das, dass alle notwendigen Elemente zwar im Programm vorhanden sind (ansonsten könnte die Funktionalität ja nicht hergestellt werden), das endgültige Aussehen der Oberfläche sich aber erst später ergibt.

Für unser Minimalprogramm können wir die Oberfläche auch gleich richtig gestalten. Wir verwenden ein Formular, dessen Größe wir ein wenig verringern. Weiterhin benötigen wir eine Komponente vom Typ `TButton` oder `TBitBtn` zum Beenden des Programms, je eine Komponente vom Typ `TEdit` zur Eingabe des Bruttowerts und des Prozentsatzes und ein weiteres Eingabefeld vom Typ `TEdit`, das wir allerdings für die Ausgabe unseres Ergebnisses verwenden. Aus diesem Grund geben wir diesem Feld – im Beispiel ist es `Edit3` – eine andere Farbe. Setzen Sie die Eigenschaft `Color` des Eingabefelds auf den Wert `clBtnFace`.

Damit der Anwender weiß, wo er die Daten eingeben muss, beschriften wir die Eingabefelder mit Hilfe von drei Komponenten vom Typ `TLabel`. Der Wert der Eigenschaft `Caption` entspricht dem, was angezeigt wird. Ebenso setzen wir den Wert der Eigenschaft `Caption` des Formulars.

Unsere komplette Programmoberfläche, noch bar jeder Funktionalität, sieht danach aus wie in Bild 3.1:

Bild 3.1: Die Oberfläche des Beispielprogramms

3.2 Die Grundfunktionen hinzufügen

Überlegen wir uns, was wir an grundlegenden Funktionen benötigen. Als grundlegend bezeichnen wir in diesem Fall alle Funktionen, die für einen Windows-Anwender »normal« sind. Die eigentliche Berechnung des Ergebnisses ist die Hauptfunktion des Programms.

3.2.1 Das Programm beenden

Als Erstes sorgen wir dafür, dass unser `TButton` zum Beenden funktioniert. Die Funktionalität für die Schaltflächen in der Titelleiste des Formulars wird von Delphi automatisch zur Verfügung gestellt und entspricht dem üblichen Verhalten dieser Schaltflächen unter Windows.

Um unserer eigenen Schaltfläche die gleiche Funktionalität hinzuzufügen, müssen wir nun erstmals Programmcode eingeben. Wenn Sie auf die Schaltfläche doppelklicken, erstellt Delphi automatisch den Rumpf für die Standard-Ereignisbehandlungsroutine der Schaltfläche. In diesem Fall handelt es sich um das Ereignis `OnClick`, das dann auftritt, wenn der Anwender auf die Schaltfläche klickt.

Die Codezeile, die wir hinzufügen müssen, beschränkt sich eigentlich auf ein einziges Wort, abgeschlossen mit einem Semikolon:

```
close;
```

Durch diese Anweisung wird das Hauptfenster der Anwendung geschlossen und damit auch die Anwendung selbst.

3.2.2 Das Ergebnisfeld konfigurieren

Der Anwender empfindet es als normal, dass z. B. in das Ergebnisfeld nichts eingegeben werden kann – immerhin ist es ein Feld, in dem das Programm etwas ausgeben soll. Aus diesem Grund müssen wir verhindern, dass dieses Feld den Eingabefokus erhalten kann, denn das würde den Anwender irritieren. Wir können uns dabei allerdings auf die Anpassung der Eigenschaften im Objektinspektor beschränken und müssen nicht eine Zeile Programmcode schreiben.

Um zu verhindern, dass der Anwender eine Eingabe in unserem Eingabefeld `Edit3` tätigen kann, setzen wir einfach dessen Eigenschaft `ReadOnly` auf `true`. Damit kann zwar das Eingabefeld immer noch den Fokus erhalten, der Anwender kann jetzt aber zumindest nichts mehr eingeben.

3.2.3 Die Tabulatorreihenfolge

Alle Komponenten auf dem Formular unterliegen einer Reihenfolge, nach der sie angesprungen werden. Normalerweise ist diese Reihenfolge gleich der Erstellungsfolge, man kann sie jedoch ändern. Und man kann eine Komponente aus der Reihenfolge herausnehmen, so dass sie nicht mehr angesprungen wird. Dazu setzen wir die Eigenschaft `TabStop` der entsprechenden Komponente auf `false`, wodurch sie nicht mehr angesprungen werden kann.

Genau das tun wir jetzt mit unserem Ergebnisfeld `Edit3`. Indem wir es aus der Tabulatorreihenfolge herausnehmen, kann es nun zumindest nicht mehr mit Hilfe der Tabulatortaste angesprungen werden. Und schon haben wir auch diesen Teil der Funktionalität erledigt und somit eigentlich schon die gesamte Grundfunktionalität hergestellt. Wenn Sie wollen, können Sie das Programm jetzt schon einmal probeweise starten. Sie werden sehen, dass Sie bereits Eingaben täti-

gen können, dass die Tabulatortaste und die verschiedenen Schaltflächen funktionieren und dass sich auch das Fenster genau so wie alle anderen Windows-Fenster verhält. Und das alles mit nur einer Zeile Programmcode.

3.2.4 Der Text in den Eingabefeldern

Natürlich soll der Anwender seinen Text selbst eingeben, und vor allem wollen wir nicht, dass in den Eingabefeldern deren Name steht. Wir müssen diesen Text also löschen. Sie finden ihn in der Eigenschaft Text der jeweiligen Komponente. Löschen Sie einfach den Inhalt dieser Eigenschaft, und schon sind unsere Felder leer.

3.3 Die Hauptfunktion hinzufügen

Die Hauptfunktion ist unsere Berechnung. Da wir keine Schaltfläche für die Berechnung vorgesehen haben, müssen wir uns einen anderen Weg überlegen. Sinnvoll wäre es z. B., die Berechnung dann durchzuführen, wenn der Fokus von einem zum anderen Eingabefeld wechselt.

Delphi arbeitet ereignisorientiert. Auch beim Fokuswechsel tritt natürlich ein Ereignis auf, das wir dazu verwenden können, die Berechnung durchzuführen. Wir müssen uns lediglich entscheiden, ob wir beim Verlassen eines Eingabefelds oder beim Eintritt in ein Eingabefeld rechnen wollen. In diesem Fall habe ich mich für die Berechnung beim Verlassen des Eingabefelds entschieden, das entsprechende Ereignis lautet `OnExit`. In der Ereignisbehandlungsroutine zu diesem Ereignis werden wir die eigentliche Berechnung vornehmen.

Delphi erstellt wieder den Rumpf der Ereignisbehandlungsroutine, wenn wir im Objektinspektor auf das leere Feld neben dem Ereignisnamen doppelklicken. Wir müssen nur noch die Funktionalität hinzufügen.

Damit wir überhaupt rechnen können, benötigen wir einige Variablen. Für uns ist aber nur das Ergebnis der Berechnung interessant, wir müssen auf diese Variablen später nicht mehr zugreifen. Deshalb deklarieren wir sie lokal, also nur gültig für die Funktion, in der wir uns momentan befinden.

```
procedure TForm1.Edit1Exit(Sender: TObject);
var
  BruttoWert  : double;
  Prozentsatz : integer;
  Prozentwert,
  Nettowert   : double;
```

Jetzt überprüfen wir, ob auch wirklich beide Eingabefelder, das für den Bruttowert und das für den Nettowert, eine Eingabe enthalten. Wir kontrollieren dazu die Eigenschaft `Text` der Eingabefelder:

```
begin
  if (Edit1.Text='') or (Edit2.Text='') then
    exit;
```

Ist eines der Eiungabefelder leer, wird keine Berechnung durchgeführt und die Ereignisbehandlungsroutine wird ohne weiteres verlassen. Dazu dient der Befehl `exit`.

Wenn beide Felder Eingaben enthalten, können wir die Berechnung durchführen. Allerdings liegen die Daten zunächst noch als Werte des Datentyps `string` vor, wir benötigen aber `integer`- bzw. `double`-Werte. Delphi stellt dafür Umwandlungsfunktionen zur Verfügung, mit deren Hilfe wir die Eingaben in den Eingabefeldern zunächst den von uns lokal deklarierten Variablen zuweisen:

```
  BruttoWert  := StrToFloat(Edit1.Text);
  Prozentsatz := StrToInt(Edit2.Text);
  if (Prozentsatz>99) or (Prozentsatz<1) then
    exit;
```

An dieser Stelle habe ich noch eine kleine Kontrolle eingefügt, die die Routine dann ohne weitere Berechnung verlässt, wenn der Prozentsatz nicht zwischen 1 und 100 liegt. Ein Prozentsatz von 0% wäre ebenso unsinnig wie ein Prozentsatz von 100% oder höher. Wir brechen also ab, wenn dies der Fall ist.

Der nächste Schritt ist die Berechnung des Nettowertes, einfache Prozentrechnung, für uns und für Delphi sehr leicht zu bewerkstelligen:

```
ProzentWert := ((Bruttowert*Prozentsatz)/100);
NettoWert   := BruttoWert-ProzentWert;
```

Und zuletzt schreiben wir das Ergebnis in das entsprechende Feld `Edit3`, das wir dafür vorgesehen hatten. Hier müssen wir wieder eine Umwandlung vornehmen, denn unser Ergebnis hat ja den Datentyp `Real` und die Eigenschaft `Text` von `Edit3`, der wir das Ergebnis zuweisen, ist vom Typ `String`:

```
  Edit3.Text := FloatToStr(NettoWert);
end;
```

Hier nochmals die gesamte Funktion, wie sie sich in Delphi präsentiert. Die Kommentare wurden von mir hinzugefügt, damit Sie die einzelnen Teile unterscheiden können:

```
procedure TForm1.Edit1Exit(Sender: TObject);
var
  BruttoWert  : double;
  Prozentsatz : integer;
  Prozentwert,
  Nettowert   : double;
begin
  //Kontrollieren ob die Eingabefelder nicht leer sind
  if (Edit1.Text='') or (Edit2.Text='') then
    exit;

  //Umwandeln der Werte
  BruttoWert  := StrToFloat(Edit1.Text);
  Prozentsatz := StrToInt(Edit2.Text);
  if (Prozentsatz>99) or (Prozentsatz<1) then
    exit;

  //Berechnen
  ProzentWert := ((Bruttowert*Prozentsatz)/100);
  NettoWert   := BruttoWert-ProzentWert;

 //Ergebnis schreiben
  Edit3.Text := FloatToStr(NettoWert);
end;
```

Die Berechnung wird jetzt durchgeführt, wenn der Anwender das Feld `Edit1` verlässt. Das ist noch nicht ganz in unserem Sinne, denn wir wollen ja auch dann rechnen, wenn er `Edit2` verlässt. Andernfalls würde ja nach der Eingabe der Prozentzahl keine Berechnung durchgeführt.

Doch das ist eigentlich kein Problem. Wir benutzen die gleiche Ereignisbehandlungsroutine, diesmal aber für das Feld `Edit2`. Und statt alles nochmal zu programmieren, leiten wir diese Ereignisbehandlungsroutine um. Wir sagen Delphi also: Wenn das Ereignis `OnExit` von `Edit2` auftritt, führe die entsprechende Ereignisbehandlungsroutine von `Edit1` aus.

Auf der Seite EREIGNISSE im Objektinspektor sehen Sie, dass sich neben den Ereignissen aufklappbare Listenfelder befinden. Wählen Sie `Edit2` aus und klappen Sie das Listenfeld neben dem Ereignis `OnExit` auf. Sie werden den Eintrag Edit1Exit finden. Wählen Sie ihn aus und die Funktionalität ist ok.

Somit wäre die Hauptfunktion des Programms ebenfalls fertig. Was jetzt noch fehlt ist ein gewisser Teil »Usability«. Hauptsächlich gehört dazu, dass der Anwender keine Falscheingabe machen kann.

3.4 Falscheingaben verhindern

Es ist wichtig zu verhindern, dass der Anwender eine Falscheingabe tätigen kann. Wenn er z. B. einen Buchstaben oder einen Namen eingeben würde, würde unsere Berechnungsroutine feststellen, dass sich in dem entsprechenden Feld etwas befindet und mit der Berechnung beginnen. Bei der Umwandlung der Werte von `string` nach `integer` hätten wir bereits einen Fehler, die Berechnung funktioniert nicht.

Wir könnten natürlich diesen Fehler zur Laufzeit erkennen und abfangen, dem Anwender eine entsprechende Nachricht zukommen lassen und ihn die Eingabe erneut tätigen lassen. Das ist allerdings nicht elegant und entspricht nicht dem Verhalten heutiger Programme. Besser ist es, eine Falscheingabe erst gar nicht zuzulassen.

Wir können wieder ein Ereignis verwenden. Wenn der Anwender eine Taste drückt, treten zwei Ereignisse auf, die wir verwenden könnten: `OnKeyPress` und `OnKeyDown`, jeweils für das Eingabefeld, in dem er sich befindet. Wir verwenden das erste, da es uns den ASCII-Code des gedrückten Zeichens liefert.

Die Verhinderung der Eingabe für den Prozentsatz ist sehr einfach zu bewerkstelligen, da wir mit einem ganzzahligen Prozentsatz rechnen. Wir müssen also jedes Zeichen abfangen, das keine Zahl ist. Außerdem müssen wir die ← -Taste berücksichtigen. Anders als die Taste Entf, die eine Sondertaste ist, hat ← einen ASCII-Wert, nämlich dezimal 8. Die Zahlen finden wir unter den ASCII-Codes 48 bis 57, jeweils dezimal. Der Wert 0 teilt dem Programm mit, dass es sich um keinen Tastendruck handelt. Die Ereignisbehandlungsroutine sieht daher folgendermaßen aus:

```
procedure TForm1.Edit2KeyPress(Sender: TObject;
                               var Key: Char);
begin
 if not (Key in [#8,#48..#57]) then
    Key := #0;
end;
```

Der Parameter `Key`, der ja den ASCII-Code unseres Zeichens enthält, wird nach Behandlung der Routine wieder zurückgeliefert. Wenn wir also möchten, dass Windows nicht auf den Tastendruck reagiert, müssen wir den ASCII-Wert 0 zurückliefern, ausgedrückt durch das Rautenzeichen und den Wert. Wir tun es dann, wenn es sich bei der gedrückten Taste nicht um eine Ziffer oder die ← -Taste handelt.

Bei unserem Eingabefeld für den Bruttowert wird es nun ein wenig umfangreicher, denn hier haben wir eine Besonderheit: Der Anwender muss die Möglichkeit haben, auch ein Komma einzugeben, da es sich ja um einen Gleitkommawert handeln soll. Viele Anwender benutzen hierfür aber auch den Punkt. Das Trennzeichen für die Nachkommastellen ist im Deutschen aber das Komma. Außerdem darf er es nur ein einziges Mal eingeben, denn zwei Kommata sind in diesem Fall nicht erlaubt. Wir müssen uns also darum kümmern, dass die entsprechenden Kontrollen durchgeführt werden.

Zunächst kontrollieren wir wie gehabt die gedrückte Taste, diesmal sollen aber mehr Werte erlaubt sein. Der Wert #44 für das Komma und der Wert #46 für den Punkt kommen hinzu. Das ist Absicht, denn viele Anwender sind es gewohnt, zur Trennung den Punkt zu verwenden, wie das in manchen Anwendungen der Fall ist. Wir werden diesen Punkt dann automatisch in ein Komma konvertieren.

```
if not (Key in [#8,#44,#46,#48..#57]) then
  Key := #0;

//Wenn Punkt, dann nach Komma ändern
if Key=#46 then
  Key := #44;
```

Wenn nun ein Komma eingegeben wurde, müssen wir kontrollieren, ob nicht bereits eines enthalten ist, denn nur dann dürfen wir die Eingabe erlauben. Dazu verwenden wir die Funktion Pos, mit der wir das Vorkommen eines Zeichens oder einer Zeichenkette in einem String kontrollieren können. Wenn das Zeichen (oder die Zeichenkette) in dem entsprechenden String enthalten ist, wird der Index des ersten Zeichens zurückgegeben. Ist das Zeichen (die Zeichenkette) nicht enthalten, wird der Wert 0 zurückgegeben. In diesem Fall können wir die Eingabe zulassen, andernfalls setzen wir Key wieder auf #0.

```
if Key=#44 then
  if Pos(Key,Edit1.Text)<>0 then
    Key:=#0;
```

Damit wäre die Ereignisbehandlungsroutine komplett. Die gesamte Routine (und so kompliziert ist sie ja wirklich nicht) sieht folgendermaßen aus:

```
procedure TForm1.Edit1KeyPress(Sender: TObject;
                               var Key: Char);
begin
  if not (Key in [#8,#44,#46,#48..#57]) then
    Key := #0;

  //Wenn Punkt, dann nach Komma ändern
  if Key=#46 then
    Key := #44;
```

```
  //Kontrollieren, ob Komma schon vorhanden
  if Key=#44 then
    if Pos(Key,Edit1.Text)<>0 then
      Key:=#0;
end;
```

Damit wäre die Funktionalität komplett. Das Schöne an der Programmierung mit Delphi und seinen Ereignissen wird aber dann klar, wenn der Prozentwert auch als Gleitkommawert eingegeben werden müsste. In diesem Fall müssen Sie keine neue Ereignisbehandlungsroutine schreiben, die wie die oben angegebene die Eingabe eines Kommas kontrolliert. Statt dessen können Sie die bereits erstellte Ereignisbehandlungsroutine auch dem Ereignis `OnKeyPress` des anderen Eingabefelds zuweisen und hätten die Funktion wieder hergestellt.

Dieses kleine Beispiel sollte Ihnen ein wenig zeigen, wie die Programmierung mit Delphi vonstatten geht und Ihnen nebenbei schon ein paar Funktionen von Delphi aufzeigen. Alle Funktionen habe ich natürlich auch nicht im Kopf, daher benutze auch ich immer wieder die Online-Hilfe von Delphi (über die Taste F1). Aber auch Sie werden sich im Laufe der Zeit einige wichtige Funktionen aneignen, die Sie nie mehr vergessen, weil sie oft in Gebrauch sind. Aller Anfang ist natürlich schwer (das ist immer so), aber Sie werden sehen, dass Sie schon innerhalb kurzer Zeit zu ansprechenden und respektablen Ergebnissen kommen werden.

Zum Schluss der kleinen Exkursion noch eine Abbildung des fertigen Programms zur Laufzeit in Bild 3.2:

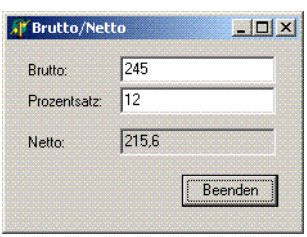

Bild 3.2: Das fertige Programm zur Laufzeit