Editoren

Aleksander

Ein Editor besteht im wesentlichen aus zwei Komponenten: einem *TextBuffer*, der den ganzen Text aufnimmt und verwaltet und einem *View*-Objekt, das einen Ausschnitt des Textes auf dem Bildschirm anzeigt und mit dem Benutzer interagiert. Für den Anfang kann als TextBuffer auch eine einfache `TStringList` herhalten. Später darf es dann aber schon auch eine etwas komplexere Klasse sein.

Dem Wesen der zwei Komponenten entsprechend gibt es auch zwei *Koordinatensysteme*. Im ersten Koordinatensystem werden die Koordinaten in Buchstaben-Einheiten angegeben. Der Punkt $(3|2)$ befindet sich also in der vierten Spalte und dritten Zeile (natürlich beginnen wir immer bei $(0|0)$ für die erste Spalte in der ersten Zeile). Das zweite Koordinatensystem gibt Punkte in Pixeln relativ zum Fenster wieder. Dort wäre der Punkt $(3|2)$ vier Pixel rechts und drei Pixel unterhalb der linken oberen Fensterecke.

Während der Bearbeitung eines Dokumentes müssen wir Punkte oft zwischen den beiden Koordinatensystemen umrechnen. Mausereignisse werden von Pixel- in Buchstabenkoordinaten umgerechnet und die Position des Carets (d. i. der blinkende Cursor) muss von Buchstabenkoordinaten in Pixelkoordinaten umgerechnet werden, etc. Für Schriften mit fester Buchstabenweite ist die Umrechnung denkbar einfach: multipliziere die Buchstabenkoordinate einfach mit der Breite bzw. Höhe eines einzelnen Zeichens.

Da die Umrechnungen oft vorkommen, ist es sinnvoll, zwei Funktionen dafür zu schreiben. *Pos* ist vom Typ `TPoint` und bezeichnet die Buchstabenposition innerhalb des Textbuffers.

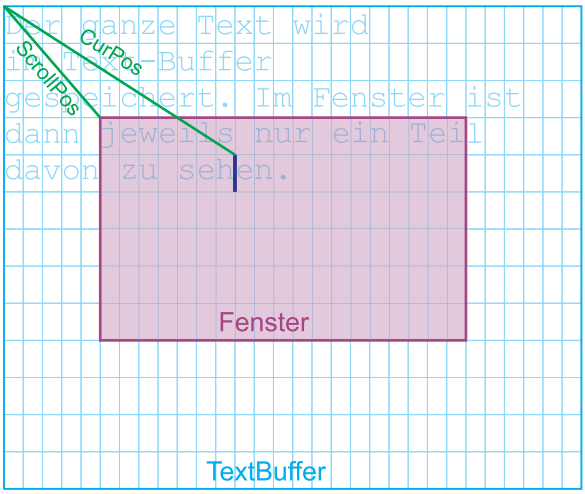
```
procedure CalcPosToCoord(Pos: TPoint; out X, Y: Integer);
begin
  X := Pos.X * CharWidth;
  Y := Pos.Y * CharHeight;
end;

procedure CalcCoordToPos(X, Y: Integer; out Pos: TPoint);
begin
  Pos.X := (X + (CharWidth div 2)) div CharWidth;
  Pos.Y := (Y + (CharHeight div 2)) div CharHeight;
end;
```

Durch das addieren der halben Buchstabenweite bzw. Höhe werden die Werte automatisch richtig gerundet.

Nun kann das Fenster ja nur einen kleinen Teil des ganzen Textes auf dem Bildschirm anzeigen. Um trotzdem den ganzen Text bearbeiten zu können, müssen wir das Fenster relativ zum darunterliegenden Text verschieben können: das führt zum *Scrollen*.

Nebst einer Variable `CurPos: TPoint`, die die Position des Carets innerhalb des Textbuffers festhält führen wir eine zweite Variable `ScrollPos: TPoint` ein. `ScrollPos` gibt dann die Position der linken oberen Ecke des Fensters in Buchstabeneinheiten an. Das folgende Bild zeigt die Situation für die Werte

```
ScrollPos.X := 5;
ScrollPos.Y := 3;
CurPos.X := 12;
CurPos.Y := 4;
```



Wenn wir die Variable `ScrollPos` bei der Umrechnung der Koordinaten berücksichtigen, werden die Funktionen zu:

```
procedure CalcPosToCoord(Pos: TPoint; out X, Y: Integer);
begin
  X := (Pos.X - ScrollPos.X) * CharWidth;
  Y := (Pos.Y - ScrollPos.Y) * CharHeight;
end;

procedure CalcCoordToPos(X, Y: Integer; out Pos: TPoint);
begin
  Pos.X := ScrollPos.X + (X + (CharWidth div 2)) div CharWidth;
  Pos.Y := ScrollPos.Y + (Y + (CharHeight div 2)) div CharHeight;
end;
```

Nun kann der Caret auch ausserhalb des Fensters zu liegen kommen. Dann müssen wir das Fenster entsprechend Scrollen, damit er wieder sichtbar wird. Dazu folgende

Idee: wenn die Variablen X und Y die Position des Carets in Pixels auf dem Bildschirm angeben, dann führt ein

```
DoScrollByPixels(X, Y);
```

im Prinzip dazu, dass der Caret genau in die linke obere Fensterecke zu liegen kommt. Wenn wir nun zuerst von X den halben `ClientWidth` abziehen, so liegt der Caret nachher in der oberen Mitte des Fensters. Das ganze in Buchstabenkoordinaten umgerechnet:

```
DoScrollBy((X - ClientWidth div 2) div CharWidth,
           Y div CharHeight);
```

Dieses Scrollverhalten können wir auch in eine eigene Funktion packen:

```
procedure ScrollToCaret;
var
  X, Y: Integer;
begin
  CalcPosToCoord(CurPos, X, Y);
  if (X < 0) or (X > ClientWidth) or
     (Y < 0) or (Y > ClientHeight) then
     DoScrollBy((X - ClientWidth div 2) div CharWidth,
                Y div CharHeight);
end;
```

Eine Alternative dazu wäre:

```
procedure ScrollToCaret;
var
  X, Y: Integer;
begin
  CalcPosToCoord(CurPos, X, Y);
  if (X < 0) or (X > ClientWidth) or
     (Y < 0) or (Y > ClientHeight) then
     DoScrollTo(CurPos.X - (ClientWidth div CharWidth),
                CurPos.Y);
end;
```